

## Abstract

We present the theory of Simpl, a sequential imperative programming language. We introduce its syntax, its semantics (big and small-step operational semantics) and Hoare logics for both partial as well as total correctness. We prove soundness and completeness of the Hoare logic. We integrate and automate the Hoare logic in Isabelle/HOL to obtain a practically usable verification environment for imperative programs.

Simpl is independent of a concrete programming language but expressive enough to cover all common language features: mutually recursive procedures, abrupt termination and exceptions, runtime faults, local and global variables, pointers and heap, expressions with side effects, pointers to procedures, partial application and closures, dynamic method invocation and also unbounded nondeterminism.

— **Simpl** —

A Sequential Imperative Programming Language  
Syntax, Semantics, Hoare Logics and Verification  
Environment

Norbert W. Schirmer

March 19, 2025

**Contents**

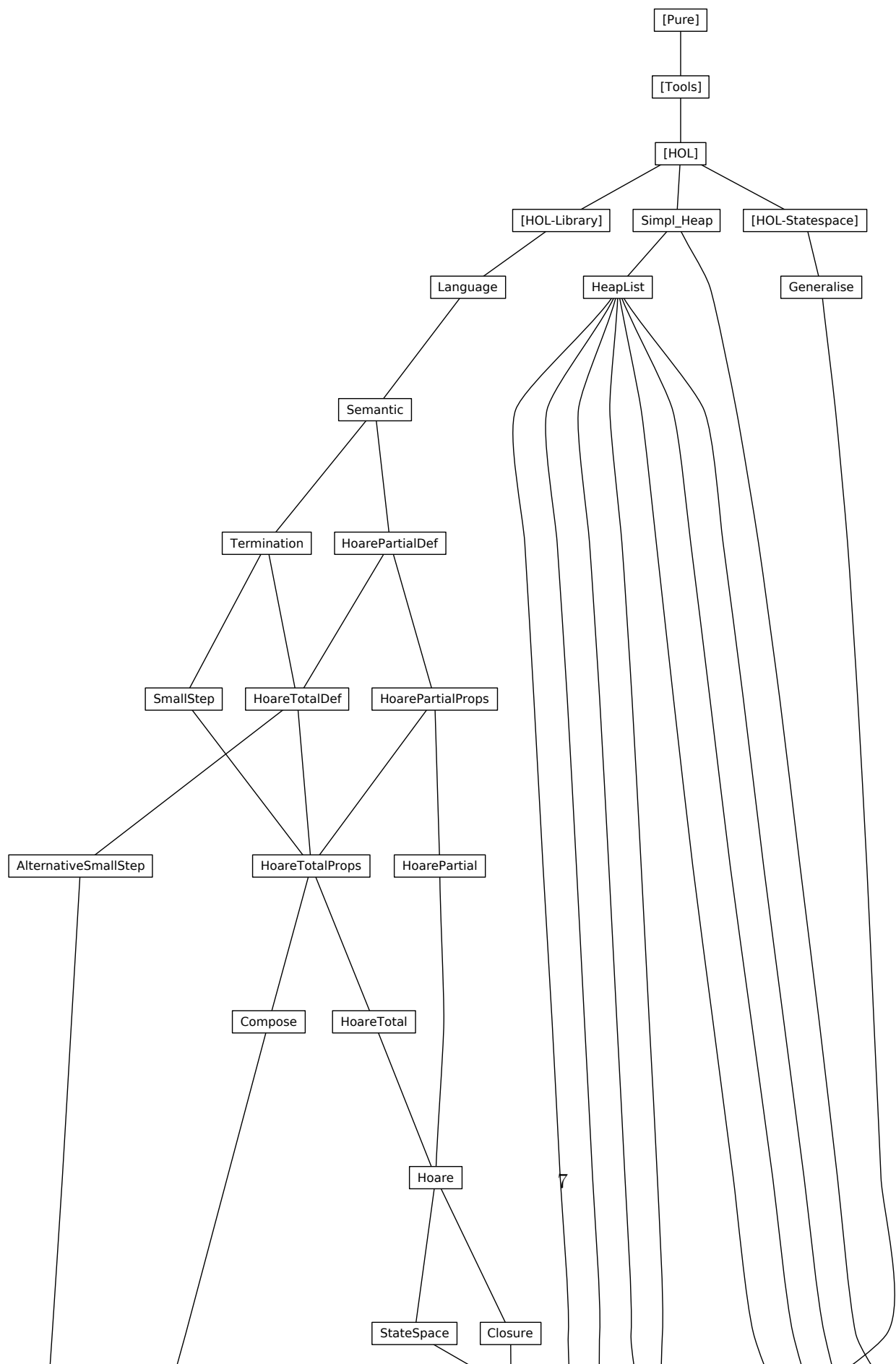
<b>1</b>	<b>Introduction</b>	<b>8</b>
<b>2</b>	<b>The Simpl Syntax</b>	<b>8</b>
2.1	The Core Language . . . . .	8
2.2	Derived Language Constructs . . . . .	8
2.3	Operations on Simpl-Syntax . . . . .	11
2.3.1	Normalisation of Sequential Composition: <i>sequence</i> , <i>flatten</i> and <i>normalize</i> . . . . .	11
2.3.2	Stripping Guards: <i>strip-guards</i> . . . . .	18
2.3.3	Marking Guards: <i>mark-guards</i> . . . . .	21
2.3.4	Merging Guards: <i>merge-guards</i> . . . . .	24
2.3.5	Intersecting Guards: $c_1 \cap_g c_2$ . . . . .	28
2.3.6	Subset on Guards: $c_1 \subseteq_g c_2$ . . . . .	32
<b>3</b>	<b>Big-Step Semantics for Simpl</b>	<b>33</b>
3.1	Big-Step Execution: $\Gamma \vdash \langle c, s \rangle \Rightarrow t$ . . . . .	34
3.2	Big-Step Execution with Recursion Limit: $\Gamma \vdash \langle c, s \rangle =_n \Rightarrow t$ . . . . .	47
3.3	Lemmas about <i>sequence</i> , <i>flatten</i> and <i>Language.normalize</i> . . . . .	68
3.4	Lemmas about $c_1 \subseteq_g c_2$ . . . . .	75
3.5	Lemmas about <i>merge-guards</i> . . . . .	84
3.6	Lemmas about <i>mark-guards</i> . . . . .	89
3.7	Lemmas about <i>strip-guards</i> . . . . .	102
3.8	Lemmas about $c_1 \cap_g c_2$ . . . . .	121
3.9	Restriction of Procedure Environment . . . . .	133
3.10	Miscellaneous . . . . .	138

<b>4</b>	<b>Hoare Logic for Partial Correctness</b>	<b>139</b>
4.1	Validity of Hoare Tuples: $\Gamma, \Theta \models_{/F} P \ c \ Q, A$ . . . . .	139
4.2	Properties of Validity . . . . .	140
4.3	The Hoare Rules: $\Gamma, \Theta \vdash_{/F} P \ c \ Q, A$ . . . . .	143
4.4	Some Derived Rules . . . . .	146
<b>5</b>	<b>Properties of Partial Correctness Hoare Logic</b>	<b>147</b>
5.1	Soundness . . . . .	147
5.2	Completeness . . . . .	155
5.3	And Now: Some Useful Rules . . . . .	169
5.3.1	Consequence . . . . .	169
5.3.2	Modify Return . . . . .	175
5.3.3	DynCall . . . . .	181
5.3.4	Conjunction of Postcondition . . . . .	188
5.3.5	Weaken Context . . . . .	190
5.3.6	Guards and Guarantees . . . . .	191
5.3.7	Restricting the Procedure Environment . . . . .	199
<b>6</b>	<b>Derived Hoare Rules for Partial Correctness</b>	<b>200</b>
6.1	Rules for Single-Step Proof . . . . .	228
<b>7</b>	<b>Terminating Programs</b>	<b>230</b>
7.1	Inductive Characterisation: $\Gamma \vdash c \downarrow s$ . . . . .	230
7.2	Lemmas about <i>sequence</i> , <i>flatten</i> and <i>Language.normalize</i> . . . . .	238
7.3	Lemmas about <i>strip-guards</i> . . . . .	245
7.4	Lemmas about $c_1 \cap_g c_2$ . . . . .	250
7.5	Lemmas about <i>mark-guards</i> . . . . .	255
7.6	Lemmas about <i>merge-guards</i> . . . . .	260
7.7	Lemmas about $c_1 \subseteq_g c_2$ . . . . .	264
7.8	Lemmas about <i>strip-guards</i> . . . . .	272
7.9	Miscellaneous . . . . .	275
<b>8</b>	<b>Small-Step Semantics and Infinite Computations</b>	<b>279</b>
8.1	Small-Step Computation: $\Gamma \vdash (c, s) \rightarrow (c', s')$ . . . . .	279
8.2	Structural Properties of Small Step Computations . . . . .	282
8.3	Equivalence between Small-Step and Big-Step Semantics . . . . .	286
8.4	Infinite Computations: $\Gamma \vdash (c, s) \rightarrow \dots(\infty)$ . . . . .	298
8.5	Equivalence between Termination and the Absence of Infinite Computations . . . . .	298
8.6	Generalised Redexes . . . . .	335
<b>9</b>	<b>Hoare Logic for Total Correctness</b>	<b>342</b>
9.1	Validity of Hoare Tuples: $\Gamma \models_{t/F} P \ c \ Q, A$ . . . . .	342
9.2	Properties of Validity . . . . .	342

9.3	The Hoare Rules: $\Gamma, \Theta \vdash_{t/F} P \ c \ Q, A$ . . . . .	343
9.4	Some Derived Rules . . . . .	346
<b>10</b>	<b>Properties of Total Correctness Hoare Logic</b>	<b>348</b>
10.1	Soundness . . . . .	348
10.2	Completeness . . . . .	360
10.3	And Now: Some Useful Rules . . . . .	393
10.3.1	Modify Return . . . . .	393
10.3.2	DynCall . . . . .	400
10.3.3	Conjunction of Postcondition . . . . .	410
10.3.4	Guards and Guarantees . . . . .	411
10.3.5	Restricting the Procedure Environment . . . . .	421
10.3.6	Miscellaneous . . . . .	422
<b>11</b>	<b>Derived Hoare Rules for Total Correctness</b>	<b>424</b>
11.0.1	Rules for Single-Step Proof . . . . .	452
<b>12</b>	<b>Auxiliary Definitions/Lemmas to Facilitate Hoare Logic</b>	<b>454</b>
<b>13</b>	<b>State Space Template</b>	<b>462</b>
<b>14</b>	<b>Alternative Small Step Semantics</b>	<b>462</b>
14.1	Small-Step Computation: $\Gamma \vdash (cs, css, s) \rightarrow (cs', css', s')$ . . .	463
14.1.1	Structural Properties of Small Step Computations . .	465
14.1.2	Equivalence between Big and Small-Step Semantics .	470
14.2	Infinite Computations: $\text{inf } \Gamma \ cs \ css \ s$ . . . . .	476
14.3	Equivalence of Termination and Absence of Infinite Computations . . . . .	476
14.4	Completeness of Total Correctness Hoare Logic . . . . .	523
14.5	References . . . . .	542
<b>15</b>	<b>Paths and Lists in the Heap</b>	<b>544</b>
15.1	Paths in The Heap . . . . .	544
15.2	Lists on The Heap . . . . .	545
15.2.1	Relational Abstraction . . . . .	545
15.3	Functional abstraction . . . . .	547
<b>16</b>	<b>Facilitating the Hoare Logic</b>	<b>552</b>
16.1	Some Fancy Syntax . . . . .	553
<b>17</b>	<b>Examples using the Verification Environment</b>	<b>565</b>
17.1	State Spaces . . . . .	565
17.2	Basic Examples . . . . .	566
17.3	Multiplication by Addition . . . . .	568
17.4	Summing Natural Numbers . . . . .	571

17.5 SWITCH . . . . .	573
17.6 (Mutually) Recursive Procedures . . . . .	573
17.6.1 Factorial . . . . .	573
17.6.2 Odd and Even . . . . .	576
17.7 Expressions With Side Effects . . . . .	578
17.8 Global Variables and Heap . . . . .	578
17.8.1 Insertion Sort . . . . .	581
17.8.2 Memory Allocation and Deallocation . . . . .	583
17.9 Fault Avoiding Semantics . . . . .	585
17.10 Circular Lists . . . . .	588
<b>18 Examples using Statespaces</b>	<b>592</b>
18.1 State Spaces . . . . .	592
18.2 Basic Examples . . . . .	593
18.3 Multiplication by Addition . . . . .	595
18.4 Summing Natural Numbers . . . . .	597
18.5 SWITCH . . . . .	599
18.6 (Mutually) Recursive Procedures . . . . .	600
18.6.1 Factorial . . . . .	600
18.6.2 Odd and Even . . . . .	603
18.7 Expressions With Side Effects . . . . .	604
18.8 Global Variables and Heap . . . . .	605
18.8.1 Insertion Sort . . . . .	607
18.8.2 Memory Allocation and Deallocation . . . . .	609
18.9 Fault Avoiding Semantics . . . . .	611
18.10 Circular Lists . . . . .	614
<b>19 Examples for Total Correctness</b>	<b>617</b>
<b>20 Example: Quicksort on Heap Lists</b>	<b>624</b>
<b>21 Examples for Parallel Assignments</b>	<b>629</b>
<b>22 Examples for Procedures as Parameters</b>	<b>631</b>
<b>23 Examples for Procedures as Parameters using Statespaces</b>	<b>635</b>
<b>24 Experiments with Closures</b>	<b>641</b>
<b>25 Experiments on State Composition</b>	<b>653</b>
25.1 Changing the State-Space . . . . .	653
25.2 Renaming Procedures . . . . .	670

<b>26 User Guide</b>	<b>683</b>
26.1 Basics	683
26.2 Procedures	685
26.2.1 Declaration	685
26.2.2 Verification	687
26.2.3 Usage	688
26.2.4 Recursion	689
26.3 Global Variables and Heap	691
26.4 Total Correctness	695
26.5 Guards	701
26.6 Miscellaneous Techniques	702
26.6.1 Modifies Clause	702
26.6.2 Annotations	703
26.6.3 Total Correctness of Nested Loops	708
26.7 Functional Correctness, Termination and Runtime Faults	710
26.8 Procedures and Locales	711
26.9 Records	712
26.9.1 Extending State Spaces	713
26.9.2 Mapping Variables to Record Fields	713



# 1 Introduction

The work presented in these theories was developed within the German Verisoft project<sup>1</sup>. A thorough description of the core parts can be found in my PhD thesis [9]. A tutorial-like user guide is in Section 26.

Applications so far include BDD-normalisation [8], a C0 compiler [4], a page fault handler [1] and extensions towards separation logic [10].

## 2 The Simpl Syntax

**theory** *Language* **imports** *HOL–Library.Old-Recdef* **begin**

### 2.1 The Core Language

We use a shallow embedding of boolean expressions as well as assertions as sets of states.

**type-synonym** *'s bexp* = *'s set*  
**type-synonym** *'s assn* = *'s set*

**datatype** (*dead 's, 'p, 'f*) *com* =  
  *Skip*  
  | *Basic 's*  $\Rightarrow$  *'s*  
  | *Spec ('s*  $\times$  *'s)* *set*  
  | *Seq ('s, 'p, 'f) com ('s, 'p, 'f) com*  
  | *Cond 's bexp ('s, 'p, 'f) com ('s, 'p, 'f) com*  
  | *While 's bexp ('s, 'p, 'f) com*  
  | *Call 'p*  
  | *DynCom 's*  $\Rightarrow$  (*'s, 'p, 'f*) *com*  
  | *Guard 'f 's bexp ('s, 'p, 'f) com*  
  | *Throw*  
  | *Catch ('s, 'p, 'f) com ('s, 'p, 'f) com*

### 2.2 Derived Language Constructs

**definition**

*raise*:: (*'s*  $\Rightarrow$  *'s*)  $\Rightarrow$  (*'s, 'p, 'f*) *com* **where**  
*raise f* = *Seq (Basic f) Throw*

**definition**

*condCatch*:: (*'s, 'p, 'f*) *com*  $\Rightarrow$  *'s bexp*  $\Rightarrow$  (*'s, 'p, 'f*) *com*  $\Rightarrow$  (*'s, 'p, 'f*) *com* **where**  
*condCatch c*<sub>1</sub> *b c*<sub>2</sub> = *Catch c*<sub>1</sub> (*Cond b c*<sub>2</sub> *Throw*)

**definition**

*bind*:: (*'s*  $\Rightarrow$  *'v*)  $\Rightarrow$  (*'v*  $\Rightarrow$  (*'s, 'p, 'f*) *com*)  $\Rightarrow$  (*'s, 'p, 'f*) *com* **where**  
*bind e c* = *DynCom* ( $\lambda s. c (e s)$ )

---

<sup>1</sup><http://www.verisoft.de>



**definition**

$bseq:: ('s, 'p, 'f) com \Rightarrow ('s, 'p, 'f) com \Rightarrow ('s, 'p, 'f) com$  **where**  
 $bseq = Seq$

**definition**

$block-ern:: ['s \Rightarrow 's, ('s, 'p, 'f) com, 's \Rightarrow 's \Rightarrow 's, 's \Rightarrow 's \Rightarrow 's, 's \Rightarrow 's \Rightarrow ('s, 'p, 'f) com] \Rightarrow ('s, 'p, 'f) com$

**where**

$block-ern$  *init* *bdy* *return* *result-ern*  $c =$   
 $DynCom (\lambda s. (Seq (Catch (Seq (Basic$  *init*) *bdy*) (Seq (Basic ( $\lambda t. result-ern$  (return  $s$   $t$ )  $t$ )) Throw)))  
 $(DynCom (\lambda t. Seq (Basic (return  $s$ )) (c  $s$   $t$ ))))$   
 $)$

**definition**

$call-ern:: ('s \Rightarrow 's) \Rightarrow 'p \Rightarrow ('s \Rightarrow 's \Rightarrow 's) \Rightarrow ('s \Rightarrow 's \Rightarrow 's) \Rightarrow ('s \Rightarrow 's \Rightarrow ('s, 'p, 'f) com) \Rightarrow ('s, 'p, 'f) com$  **where**  
 $call-ern$  *init*  $p$  *return* *result-ern*  $c = block-ern$  *init* (Call  $p$ ) *return* *result-ern*  $c$

**primrec**  $guards:: ('f \times 's set) list \Rightarrow ('s, 'p, 'f) com \Rightarrow ('s, 'p, 'f) com$

**where**

$guards [] c = c$  |  
 $guards (g\#gs) c = Guard (fst g) (snd g) (guards gs c)$

**definition**  $maybe-guard:: 'f \Rightarrow 's set \Rightarrow ('s, 'p, 'f) com \Rightarrow ('s, 'p, 'f) com$

**where**

$maybe-guard f g c = (if g = UNIV then c else Guard f g c)$

**lemma**  $maybe-guard-UNIV$  [*simp*]:  $maybe-guard f UNIV c = c$

**by** (*simp* *add: maybe-guard-def*)

**definition**

$dynCall-ern:: 'f \Rightarrow 's set \Rightarrow ('s \Rightarrow 's) \Rightarrow ('s \Rightarrow 'p) \Rightarrow ('s \Rightarrow 's \Rightarrow 's) \Rightarrow ('s \Rightarrow 's \Rightarrow 's) \Rightarrow ('s \Rightarrow 's \Rightarrow ('s, 'p, 'f) com) \Rightarrow ('s, 'p, 'f) com$  **where**

$dynCall-ern f g$  *init*  $p$  *return* *result-ern*  $c =$   
 $maybe-guard f g (DynCom (\lambda s. call-ern$  *init* ( $p$   $s$ ) *return* *result-ern*  $c$ ))

**definition**

$block:: ['s \Rightarrow 's, ('s, 'p, 'f) com, 's \Rightarrow 's \Rightarrow 's, 's \Rightarrow 's \Rightarrow ('s, 'p, 'f) com] \Rightarrow ('s, 'p, 'f) com$

**where**

$block$  *init* *bdy* *return*  $c = block-ern$  *init* *bdy* *return* ( $\lambda s t. s$ )  $c$

**definition**

$call:: ('s \Rightarrow 's) \Rightarrow 'p \Rightarrow ('s \Rightarrow 's \Rightarrow 's) \Rightarrow ('s \Rightarrow 's \Rightarrow ('s, 'p, 'f) com) \Rightarrow ('s, 'p, 'f) com$

**where**

$call\ init\ p\ return\ c = block\ init\ (Call\ p)\ return\ c$

**definition**

$dynCall:: ('s \Rightarrow 's) \Rightarrow ('s \Rightarrow 'p) \Rightarrow$   
 $('s \Rightarrow 's \Rightarrow 's) \Rightarrow ('s \Rightarrow 's \Rightarrow ('s, 'p, 'f)\ com) \Rightarrow ('s, 'p, 'f)\ com$  **where**  
 $dynCall\ init\ p\ return\ c = DynCom\ (\lambda s. call\ init\ (p\ s)\ return\ c)$

**definition**

$fcall:: ('s \Rightarrow 's) \Rightarrow 'p \Rightarrow ('s \Rightarrow 's \Rightarrow 's) \Rightarrow ('s \Rightarrow 'v) \Rightarrow ('v \Rightarrow ('s, 'p, 'f)\ com)$   
 $\Rightarrow ('s, 'p, 'f)\ com$  **where**  
 $fcall\ init\ p\ return\ result\ c = call\ init\ p\ return\ (\lambda s\ t. c\ (result\ t))$

**definition**

$lem:: 'x \Rightarrow ('s, 'p, 'f)\ com \Rightarrow ('s, 'p, 'f)\ com$  **where**  
 $lem\ x\ c = c$

**primrec**  $switch:: ('s \Rightarrow 'v) \Rightarrow ('v\ set \times ('s, 'p, 'f)\ com)\ list \Rightarrow ('s, 'p, 'f)\ com$

**where**

$switch\ v\ [] = Skip\ |$   
 $switch\ v\ (Vc\ \#\ vs) = Cond\ \{s. v\ s \in fst\ Vc\}\ (snd\ Vc)\ (switch\ v\ vs)$

**definition**  $guaranteeStrip:: 'f \Rightarrow 's\ set \Rightarrow ('s, 'p, 'f)\ com \Rightarrow ('s, 'p, 'f)\ com$

**where**  $guaranteeStrip\ f\ g\ c = Guard\ f\ g\ c$

**definition**  $guaranteeStripPair:: 'f \Rightarrow 's\ set \Rightarrow ('f \times 's\ set)$

**where**  $guaranteeStripPair\ f\ g = (f, g)$

**definition**

$while:: ('f \times 's\ set)\ list \Rightarrow 's\ bexp \Rightarrow ('s, 'p, 'f)\ com \Rightarrow ('s, 'p, 'f)\ com$

**where**

$while\ gs\ b\ c = guards\ gs\ (While\ b\ (Seq\ c\ (guards\ gs\ Skip)))$

**definition**

$whileAnno::$   
 $'s\ bexp \Rightarrow 's\ assn \Rightarrow ('s \times 's)\ assn \Rightarrow ('s, 'p, 'f)\ com \Rightarrow ('s, 'p, 'f)\ com$  **where**  
 $whileAnno\ b\ I\ V\ c = While\ b\ c$

**definition**

$whileAnnoG::$   
 $('f \times 's\ set)\ list \Rightarrow 's\ bexp \Rightarrow 's\ assn \Rightarrow ('s \times 's)\ assn \Rightarrow$   
 $('s, 'p, 'f)\ com \Rightarrow ('s, 'p, 'f)\ com$  **where**  
 $whileAnnoG\ gs\ b\ I\ V\ c = while\ gs\ b\ c$

**definition**

*specAnno*:: ('a ⇒ 's assn) ⇒ ('a ⇒ ('s, 'p, 'f) com) ⇒  
('a ⇒ 's assn) ⇒ ('a ⇒ 's assn) ⇒ ('s, 'p, 'f) com  
**where** *specAnno* P c Q A = (c undefined)

**definition**

*whileAnnoFix*::  
's bexp ⇒ ('a ⇒ 's assn) ⇒ ('a ⇒ ('s × 's) assn) ⇒ ('a ⇒ ('s, 'p, 'f) com) ⇒  
('s, 'p, 'f) com **where**  
*whileAnnoFix* b I V c = While b (c undefined)

**definition**

*whileAnnoGFix*::  
('f × 's set) list ⇒ 's bexp ⇒ ('a ⇒ 's assn) ⇒ ('a ⇒ ('s × 's) assn) ⇒  
('a ⇒ ('s, 'p, 'f) com) ⇒ ('s, 'p, 'f) com **where**  
*whileAnnoGFix* gs b I V c = while gs b (c undefined)

**definition** *if-rel*:: ('s ⇒ bool) ⇒ ('s ⇒ 's) ⇒ ('s ⇒ 's) ⇒ ('s ⇒ 's) ⇒ ('s × 's) set  
**where** *if-rel* b f g h = {(s, t). if b s then t = f s else t = g s ∨ t = h s}

**lemma** *fst-guaranteeStripPair*: fst (guaranteeStripPair f g) = f  
**by** (simp add: guaranteeStripPair-def)

**lemma** *snd-guaranteeStripPair*: snd (guaranteeStripPair f g) = g  
**by** (simp add: guaranteeStripPair-def)

**lemma** *call-call-exn*: call init p return result = call-exn init p return (λs t. s) result  
**by** (simp add: call-def call-exn-def block-def)

**lemma** *dynCall-dynCall-exn*: dynCall init p return result = dynCall-exn undefined  
UNIV init p return (λs t. s) result  
**by** (simp add: dynCall-def dynCall-exn-def call-call-exn)

## 2.3 Operations on Simpl-Syntax

### 2.3.1 Normalisation of Sequential Composition: *sequence*, *flatten* and *normalize*

**primrec** *flatten*:: ('s, 'p, 'f) com ⇒ ('s, 'p, 'f) com list  
**where**

*flatten* Skip = [Skip] |  
*flatten* (Basic f) = [Basic f] |  
*flatten* (Spec r) = [Spec r] |  
*flatten* (Seq c<sub>1</sub> c<sub>2</sub>) = flatten c<sub>1</sub> @ flatten c<sub>2</sub> |  
*flatten* (Cond b c<sub>1</sub> c<sub>2</sub>) = [Cond b c<sub>1</sub> c<sub>2</sub>] |  
*flatten* (While b c) = [While b c] |  
*flatten* (Call p) = [Call p] |  
*flatten* (DynCom c) = [DynCom c] |  
*flatten* (Guard f g c) = [Guard f g c] |  
*flatten* Throw = [Throw] |

*flatten* (*Catch*  $c_1$   $c_2$ ) = [*Catch*  $c_1$   $c_2$ ]

**primrec** *sequence*:: (('s,'p,'f) com  $\Rightarrow$  ('s,'p,'f) com  $\Rightarrow$  ('s,'p,'f) com)  $\Rightarrow$   
('s,'p,'f) com list  $\Rightarrow$  ('s,'p,'f) com

**where**

*sequence* seq [] = *Skip* |  
*sequence* seq (c#cs) = (case cs of []  $\Rightarrow$  c  
| -  $\Rightarrow$  seq c (*sequence* seq cs))

**primrec** *normalize*:: ('s,'p,'f) com  $\Rightarrow$  ('s,'p,'f) com

**where**

*normalize* *Skip* = *Skip* |  
*normalize* (*Basic* f) = *Basic* f |  
*normalize* (*Spec* r) = *Spec* r |  
*normalize* (*Seq*  $c_1$   $c_2$ ) = *sequence* *Seq*  
((*flatten* (*normalize*  $c_1$ )) @ (*flatten* (*normalize*  $c_2$ ))) |  
*normalize* (*Cond* b  $c_1$   $c_2$ ) = *Cond* b (*normalize*  $c_1$ ) (*normalize*  $c_2$ ) |  
*normalize* (*While* b c) = *While* b (*normalize* c) |  
*normalize* (*Call* p) = *Call* p |  
*normalize* (*DynCom* c) = *DynCom* ( $\lambda s. (\text{normalize } (c\ s))$ ) |  
*normalize* (*Guard* f g c) = *Guard* f g (*normalize* c) |  
*normalize* *Throw* = *Throw* |  
*normalize* (*Catch*  $c_1$   $c_2$ ) = *Catch* (*normalize*  $c_1$ ) (*normalize*  $c_2$ )

**lemma** *flatten-nonEmpty*: *flatten* c  $\neq$  []

**by** (*induct* c) *simp-all*

**lemma** *flatten-single*:  $\forall c \in \text{set } (\text{flatten } c'). \text{flatten } c = [c]$

**apply** (*induct* c')

**apply** *simp*

**apply** *simp*

**apply** *simp*

**apply** (*simp* (*no-asm-use*))

**apply** *blast*

**apply** (*simp* (*no-asm-use*))

**apply** (*simp* (*no-asm-use*))

**apply** *simp*

**apply** (*simp* (*no-asm-use*))

**apply** (*simp* (*no-asm-use*))

**apply** *simp*

**apply** (*simp* (*no-asm-use*))

**done**

**lemma** *flatten-sequence-id*:

$\llbracket cs \neq [] \rrbracket; \forall c \in \text{set } cs. \text{flatten } c = [c] \implies \text{flatten } (\text{sequence } \text{Seq } cs) = cs$   
**apply** (*induct* cs)

```

apply simp
apply (case-tac cs)
apply simp
apply auto
done

```

```

lemma flatten-app:
  flatten (sequence Seq (flatten c1 @ flatten c2)) = flatten c1 @ flatten c2
apply (rule flatten-sequence-id)
apply (simp add: flatten-nonEmpty)
apply (simp)
apply (insert flatten-single)
apply blast
done

```

```

lemma flatten-sequence-flatten: flatten (sequence Seq (flatten c)) = flatten c
apply (induct c)
apply (auto simp add: flatten-app)
done

```

```

lemma sequence-flatten-normalize: sequence Seq (flatten (normalize c)) = normalize c
apply (induct c)
apply (auto simp add: flatten-app)
done

```

```

lemma flatten-normalize:  $\bigwedge x xs. \text{flatten (normalize c)} = x \# xs$ 
   $\implies (\text{case } xs \text{ of } [] \Rightarrow \text{normalize } c = x$ 
     $| (x' \# xs') \Rightarrow \text{normalize } c = \text{Seq } x (\text{sequence Seq } xs))$ 
proof (induct c)
  case (Seq c1 c2)
  have flatten (normalize (Seq c1 c2)) = x # xs by fact
  hence flatten (sequence Seq (flatten (normalize c1) @ flatten (normalize c2)))
  =
    x # xs
  by simp
  hence x-xs: flatten (normalize c1) @ flatten (normalize c2) = x # xs
  by (simp add: flatten-app)
  show ?case
  proof (cases flatten (normalize c1))
  case Nil
  with flatten-nonEmpty show ?thesis by auto
  next
  case (Cons x1 xs1)
  note Cons-x1-xs1 = this

```

```

with  $x$ - $xs$  obtain
   $x$ - $x1$ :  $x=x1$  and  $xs$ - $rest$ :  $xs=x1@flatten$  ( $normalize$   $c2$ )
  by  $auto$ 
show  $?thesis$ 
proof ( $cases$   $xs1$ )
  case  $Nil$ 
  from  $Seq.hyps$  (1) [ $OF$   $Cons$ - $x1$ - $xs1$ ]  $Nil$ 
  have  $normalize$   $c1 = x1$ 
  by  $simp$ 
  with  $Cons$ - $x1$ - $xs1$   $Nil$   $x$ - $x1$   $xs$ - $rest$  show  $?thesis$ 
  apply ( $cases$   $flatten$  ( $normalize$   $c2$ ))
  apply ( $fastforce$   $simp$   $add$ :  $flatten$ - $nonEmpty$ )
  apply  $simp$ 
  done
next
  case  $Cons$ 
  from  $Seq.hyps$  (1) [ $OF$   $Cons$ - $x1$ - $xs1$ ]  $Cons$ 
  have  $normalize$   $c1 = Seq$   $x1$  ( $sequence$   $Seq$   $xs1$ )
  by  $simp$ 
  with  $Cons$ - $x1$ - $xs1$   $Nil$   $x$ - $x1$   $xs$ - $rest$  show  $?thesis$ 
  apply ( $cases$   $flatten$  ( $normalize$   $c2$ ))
  apply ( $fastforce$   $simp$   $add$ :  $flatten$ - $nonEmpty$ )
  apply ( $simp$   $split$ :  $list.splits$ )
  done
  qed
qed
qed ( $auto$ )

lemma  $flatten$ - $raise$  [ $simp$ ]:  $flatten$  ( $raise$   $f$ ) = [ $Basic$   $f$ ,  $Throw$ ]
  by ( $simp$   $add$ :  $raise$ - $def$ )

lemma  $flatten$ - $condCatch$  [ $simp$ ]:  $flatten$  ( $condCatch$   $c1$   $b$   $c2$ ) = [ $condCatch$   $c1$   $b$ 
 $c2$ ]
  by ( $simp$   $add$ :  $condCatch$ - $def$ )

lemma  $flatten$ - $bind$  [ $simp$ ]:  $flatten$  ( $bind$   $e$   $c$ ) = [ $bind$   $e$   $c$ ]
  by ( $simp$   $add$ :  $bind$ - $def$ )

lemma  $flatten$ - $bseq$  [ $simp$ ]:  $flatten$  ( $bseq$   $c1$   $c2$ ) =  $flatten$   $c1$  @  $flatten$   $c2$ 
  by ( $simp$   $add$ :  $bseq$ - $def$ )

lemma  $flatten$ - $block$ - $exn$  [ $simp$ ]:
   $flatten$  ( $block$ - $exn$   $init$   $bdy$   $return$   $result$ - $exn$   $result$ ) = [ $block$ - $exn$   $init$   $bdy$   $return$ 
 $result$ - $exn$   $result$ ]
  by ( $simp$   $add$ :  $block$ - $exn$ - $def$ )

lemma  $flatten$ - $block$  [ $simp$ ]:
   $flatten$  ( $block$   $init$   $bdy$   $return$   $result$ ) = [ $block$   $init$   $bdy$   $return$   $result$ ]
  by ( $simp$   $add$ :  $block$ - $def$ )

```

**lemma** *flatten-call* [*simp*]: *flatten* (*call* *init* *p* *return* *result*) = [*call* *init* *p* *return* *result*]

**by** (*simp* *add*: *call-def*)

**lemma** *flatten-dynCall* [*simp*]: *flatten* (*dynCall* *init* *p* *return* *result*) = [*dynCall* *init* *p* *return* *result*]

**by** (*simp* *add*: *dynCall-def*)

**lemma** *flatten-call-exn* [*simp*]: *flatten* (*call-exn* *init* *p* *return* *result-exn* *result*) = [*call-exn* *init* *p* *return* *result-exn* *result*]

**by** (*simp* *add*: *call-exn-def*)

**lemma** *flatten-dynCall-exn* [*simp*]: *flatten* (*dynCall-exn* *f* *g* *init* *p* *return* *result-exn* *result*) = [*dynCall-exn* *f* *g* *init* *p* *return* *result-exn* *result*]

**by** (*simp* *add*: *dynCall-exn-def* *maybe-guard-def*)

**lemma** *flatten-fcall* [*simp*]: *flatten* (*fcall* *init* *p* *return* *result* *c*) = [*fcall* *init* *p* *return* *result* *c*]

**by** (*simp* *add*: *fcall-def*)

**lemma** *flatten-switch* [*simp*]: *flatten* (*switch* *v* *Vcs*) = [*switch* *v* *Vcs*]

**by** (*cases* *Vcs*) *auto*

**lemma** *flatten-guaranteeStrip* [*simp*]:

*flatten* (*guaranteeStrip* *f* *g* *c*) = [*guaranteeStrip* *f* *g* *c*]

**by** (*simp* *add*: *guaranteeStrip-def*)

**lemma** *flatten-while* [*simp*]: *flatten* (*while* *gs* *b* *c*) = [*while* *gs* *b* *c*]

**apply** (*simp* *add*: *while-def*)

**apply** (*induct* *gs*)

**apply** *auto*

**done**

**lemma** *flatten-whileAnno* [*simp*]:

*flatten* (*whileAnno* *b* *I* *V* *c*) = [*whileAnno* *b* *I* *V* *c*]

**by** (*simp* *add*: *whileAnno-def*)

**lemma** *flatten-whileAnnoG* [*simp*]:

*flatten* (*whileAnnoG* *gs* *b* *I* *V* *c*) = [*whileAnnoG* *gs* *b* *I* *V* *c*]

**by** (*simp* *add*: *whileAnnoG-def*)

**lemma** *flatten-specAnno* [*simp*]:

*flatten* (*specAnno* *P* *c* *Q* *A*) = *flatten* (*c* *undefined*)

**by** (*simp* *add*: *specAnno-def*)

**lemmas** *flatten-simps* = *flatten.simps* *flatten-raise* *flatten-condCatch* *flatten-bind*  
*flatten-block* *flatten-call* *flatten-dynCall* *flatten-fcall* *flatten-switch*  
*flatten-guaranteeStrip*

*flatten-while flatten-whileAnno flatten-whileAnnoG flatten-specAnno*

**lemma** *normalize-raise [simp]:*  
*normalize (raise f) = raise f*  
**by** (*simp add: raise-def*)

**lemma** *normalize-condCatch [simp]:*  
*normalize (condCatch c1 b c2) = condCatch (normalize c1) b (normalize c2)*  
**by** (*simp add: condCatch-def*)

**lemma** *normalize-bind [simp]:*  
*normalize (bind e c) = bind e ( $\lambda v.$  normalize (c v))*  
**by** (*simp add: bind-def*)

**lemma** *normalize-bseq [simp]:*  
*normalize (bseq c1 c2) = sequence bseq*  
*((flatten (normalize c1)) @ (flatten (normalize c2)))*  
**by** (*simp add: bseq-def*)

**lemma** *normalize-block-exn [simp]:* *normalize (block-exn init bdy return result-exn*  
*c) =*

*block-exn init (normalize bdy) return result-exn ( $\lambda s t.$  normalize*  
*(c s t))*  
**apply** (*simp add: block-exn-def*)  
**apply** (*rule ext*)  
**apply** (*simp*)  
**apply** (*cases flatten (normalize bdy)*)  
**apply** (*simp add: flatten-nonEmpty*)  
**apply** (*rule conjI*)  
**apply** *simp*  
**apply** (*drule flatten-normalize*)  
**apply** (*case-tac list*)  
**apply** *simp*  
**apply** *simp*  
**apply** (*rule ext*)  
**apply** (*case-tac flatten (normalize (c s sa))*)  
**apply** (*simp add: flatten-nonEmpty*)  
**apply** *simp*  
**apply** (*thin-tac flatten (normalize bdy) = P for P*)  
**apply** (*drule flatten-normalize*)  
**apply** (*case-tac lista*)  
**apply** *simp*  
**apply** *simp*  
**done**

**lemma** *normalize-block [simp]:* *normalize (block init bdy return c) =*  
*block init (normalize bdy) return ( $\lambda s t.$  normalize (c s t))*  
**by** (*simp add: block-def*)



**lemma** *normalize-call* [*simp*]:

*normalize (call init p return c) = call init p return ( $\lambda i t. \text{normalize } (c i t)$ )*

**by** (*simp add: call-def*)

**lemma** *normalize-call-exn* [*simp*]:

*normalize (call-exn init p return result-exn c) = call-exn init p return result-exn ( $\lambda i t. \text{normalize } (c i t)$ )*

**by** (*simp add: call-exn-def*)

**lemma** *normalize-dynCall* [*simp*]:

*normalize (dynCall init p return c) =  
dynCall init p return ( $\lambda s t. \text{normalize } (c s t)$ )*

**by** (*simp add: dynCall-def*)

**lemma** *normalize-guards* [*simp*]:

*normalize (guards gs c) = guards gs (normalize c)*

**by** (*induct gs auto*)

**lemma** *normalize-dynCall-exn* [*simp*]:

*normalize (dynCall-exn f g init p return result-exn c) =  
dynCall-exn f g init p return result-exn ( $\lambda s t. \text{normalize } (c s t)$ )*

**by** (*simp add: dynCall-exn-def maybe-guard-def*)

**lemma** *normalize-fcall* [*simp*]:

*normalize (fcall init p return result c) =  
fcall init p return result ( $\lambda v. \text{normalize } (c v)$ )*

**by** (*simp add: fcall-def*)

**lemma** *normalize-switch* [*simp*]:

*normalize (switch v Vcs) = switch v (map ( $\lambda(V,c). (V, \text{normalize } c)$ ) Vcs)*

**apply** (*induct Vcs*)

**apply** *auto*

**done**

**lemma** *normalize-guaranteeStrip* [*simp*]:

*normalize (guaranteeStrip f g c) = guaranteeStrip f g (normalize c)*

**by** (*simp add: guaranteeStrip-def*)

Sequential composition with guards in the body is not preserved by normalize

**lemma** *normalize-while* [*simp*]:

*normalize (while gs b c) = guards gs  
(While b (sequence Seq (flatten (normalize c) @ flatten (guards gs Skip))))*

**by** (*simp add: while-def*)

**lemma** *normalize-whileAnno* [*simp*]:

*normalize (whileAnno b I V c) = whileAnno b I V (normalize c)*

**by** (*simp add: whileAnno-def*)

**lemma** *normalize-whileAnnoG* [simp]:  
*normalize (whileAnnoG gs b I V c) = guards gs*  
*(While b (sequence Seq (flatten (normalize c) @ flatten (guards gs Skip))))*  
**by** (*simp add: whileAnnoG-def*)

**lemma** *normalize-specAnno* [simp]:  
*normalize (specAnno P c Q A) = specAnno P (λs. normalize (c undefined)) Q A*  
**by** (*simp add: specAnno-def*)

**lemmas** *normalize-simps* =  
*normalize.simps normalize-raise normalize-condCatch normalize-bind*  
*normalize-block normalize-call normalize-dynCall normalize-fcall normalize-switch*  
*normalize-guaranteeStrip normalize-guards*  
*normalize-while normalize-whileAnno normalize-whileAnnoG normalize-specAnno*

### 2.3.2 Stripping Guards: *strip-guards*

**primrec** *strip-guards*:: 'f set ⇒ ('s,'p,'f) com ⇒ ('s,'p,'f) com

**where**

*strip-guards F Skip = Skip |*  
*strip-guards F (Basic f) = Basic f |*  
*strip-guards F (Spec r) = Spec r |*  
*strip-guards F (Seq c<sub>1</sub> c<sub>2</sub>) = (Seq (strip-guards F c<sub>1</sub>) (strip-guards F c<sub>2</sub>)) |*  
*strip-guards F (Cond b c<sub>1</sub> c<sub>2</sub>) = Cond b (strip-guards F c<sub>1</sub>) (strip-guards F c<sub>2</sub>) |*  
*strip-guards F (While b c) = While b (strip-guards F c) |*  
*strip-guards F (Call p) = Call p |*  
*strip-guards F (DynCom c) = DynCom (λs. (strip-guards F (c s))) |*  
*strip-guards F (Guard f g c) = (if f ∈ F then strip-guards F c*  
*else Guard f g (strip-guards F c)) |*  
*strip-guards F Throw = Throw |*  
*strip-guards F (Catch c<sub>1</sub> c<sub>2</sub>) = Catch (strip-guards F c<sub>1</sub>) (strip-guards F c<sub>2</sub>)*

**definition** *strip*:: 'f set ⇒  
('p ⇒ ('s,'p,'f) com option) ⇒ ('p ⇒ ('s,'p,'f) com option)  
**where** *strip F Γ = (λp. map-option (strip-guards F) (Γ p))*

**lemma** *strip-simp* [simp]: *(strip F Γ) p = map-option (strip-guards F) (Γ p)*  
**by** (*simp add: strip-def*)

**lemma** *dom-strip*: *dom (strip F Γ) = dom Γ*  
**by** (*auto*)

**lemma** *strip-guards-idem*: *strip-guards F (strip-guards F c) = strip-guards F c*  
**by** (*induct c auto*)

**lemma** *strip-idem*: *strip F (strip F Γ) = strip F Γ*  
**apply** (*rule ext*)  
**apply** (*case-tac Γ x*)

**apply** (*auto simp add: strip-guards-idem strip-def*)  
**done**

**lemma** *strip-guards-raise* [*simp*]:  
*strip-guards F (raise f) = raise f*  
**by** (*simp add: raise-def*)

**lemma** *strip-guards-condCatch* [*simp*]:  
*strip-guards F (condCatch c1 b c2) =*  
*condCatch (strip-guards F c1) b (strip-guards F c2)*  
**by** (*simp add: condCatch-def*)

**lemma** *strip-guards-bind* [*simp*]:  
*strip-guards F (bind e c) = bind e (λv. strip-guards F (c v))*  
**by** (*simp add: bind-def*)

**lemma** *strip-guards-bseq* [*simp*]:  
*strip-guards F (bseq c1 c2) = bseq (strip-guards F c1) (strip-guards F c2)*  
**by** (*simp add: bseq-def*)

**lemma** *strip-guards-block-exn* [*simp*]:  
*strip-guards F (block-exn init bdy return result-exn c) =*  
*block-exn init (strip-guards F bdy) return result-exn (λs t. strip-guards F (c s t))*  
**by** (*simp add: block-exn-def*)

**lemma** *strip-guards-block* [*simp*]:  
*strip-guards F (block init bdy return c) =*  
*block init (strip-guards F bdy) return (λs t. strip-guards F (c s t))*  
**by** (*simp add: block-def*)

**lemma** *strip-guards-call* [*simp*]:  
*strip-guards F (call init p return c) =*  
*call init p return (λs t. strip-guards F (c s t))*  
**by** (*simp add: call-def*)

**lemma** *strip-guards-call-exn* [*simp*]:  
*strip-guards F (call-exn init p return result-exn c) =*  
*call-exn init p return result-exn (λs t. strip-guards F (c s t))*  
**by** (*simp add: call-exn-def*)

**lemma** *strip-guards-dynCall* [*simp*]:  
*strip-guards F (dynCall init p return c) =*  
*dynCall init p return (λs t. strip-guards F (c s t))*  
**by** (*simp add: dynCall-def*)

**lemma** *strip-guards-guards* [*simp*]: *strip-guards F (guards gs c) =*  
*guards (filter (λ(f,g). f ∉ F) gs) (strip-guards F c)*  
**by** (*induct gs auto*)

**lemma** *strip-guards-dynCall-exn* [simp]:  
 $strip\text{-}guards\ F\ (dynCall\text{-}exn\ f\ g\ init\ p\ return\ result\text{-}exn\ c) =$   
 $dynCall\text{-}exn\ f\ (if\ f \in F\ then\ UNIV\ else\ g)\ init\ p\ return\ result\text{-}exn\ (\lambda s\ t.$   
 $strip\text{-}guards\ F\ (c\ s\ t))$   
**by** (*simp add: dynCall-exn-def maybe-guard-def*)

**lemma** *strip-guards-fcall* [simp]:  
 $strip\text{-}guards\ F\ (fcall\ init\ p\ return\ result\ c) =$   
 $fcall\ init\ p\ return\ result\ (\lambda v. strip\text{-}guards\ F\ (c\ v))$   
**by** (*simp add: fcall-def*)

**lemma** *strip-guards-switch* [simp]:  
 $strip\text{-}guards\ F\ (switch\ v\ Vc) =$   
 $switch\ v\ (map\ (\lambda(V,c). (V, strip\text{-}guards\ F\ c))\ Vc)$   
**by** (*induct Vc*) *auto*

**lemma** *strip-guards-guaranteeStrip* [simp]:  
 $strip\text{-}guards\ F\ (guaranteeStrip\ f\ g\ c) =$   
 $(if\ f \in F\ then\ strip\text{-}guards\ F\ c$   
 $else\ guaranteeStrip\ f\ g\ (strip\text{-}guards\ F\ c))$   
**by** (*simp add: guaranteeStrip-def*)

**lemma** *guaranteeStripPair-split-conv* [simp]: *case-prod*  $c\ (guaranteeStripPair\ f\ g)$   
 $=\ c\ f\ g$   
**by** (*simp add: guaranteeStripPair-def*)

**lemma** *strip-guards-while* [simp]:  
 $strip\text{-}guards\ F\ (while\ gs\ b\ c) =$   
 $while\ (filter\ (\lambda(f,g). f \notin F)\ gs)\ b\ (strip\text{-}guards\ F\ c)$   
**by** (*simp add: while-def*)

**lemma** *strip-guards-whileAnno* [simp]:  
 $strip\text{-}guards\ F\ (whileAnno\ b\ I\ V\ c) =\ whileAnno\ b\ I\ V\ (strip\text{-}guards\ F\ c)$   
**by** (*simp add: whileAnno-def while-def*)

**lemma** *strip-guards-whileAnnoG* [simp]:  
 $strip\text{-}guards\ F\ (whileAnnoG\ gs\ b\ I\ V\ c) =$   
 $whileAnnoG\ (filter\ (\lambda(f,g). f \notin F)\ gs)\ b\ I\ V\ (strip\text{-}guards\ F\ c)$   
**by** (*simp add: whileAnnoG-def*)

**lemma** *strip-guards-specAnno* [simp]:  
 $strip\text{-}guards\ F\ (specAnno\ P\ c\ Q\ A) =$   
 $specAnno\ P\ (\lambda s. strip\text{-}guards\ F\ (c\ undefined))\ Q\ A$   
**by** (*simp add: specAnno-def*)

**lemmas** *strip-guards-simps* = *strip-guards.simps strip-guards-raise*

*strip-guards-condCatch strip-guards-bind strip-guards-bseq strip-guards-block*  
*strip-guards-dynCall strip-guards-fcall strip-guards-switch*  
*strip-guards-guaranteeStrip guaranteeStripPair-split-conv strip-guards-guards*  
*strip-guards-while strip-guards-whileAnno strip-guards-whileAnnoG*  
*strip-guards-specAnno*

### 2.3.3 Marking Guards: *mark-guards*

**primrec** *mark-guards*:: '*f* ⇒ ('*s*, '*p*, '*g*) *com* ⇒ ('*s*, '*p*, '*f*) *com*

**where**

*mark-guards f Skip = Skip* |  
*mark-guards f (Basic g) = Basic g* |  
*mark-guards f (Spec r) = Spec r* |  
*mark-guards f (Seq c<sub>1</sub> c<sub>2</sub>) = (Seq (mark-guards f c<sub>1</sub>) (mark-guards f c<sub>2</sub>))* |  
*mark-guards f (Cond b c<sub>1</sub> c<sub>2</sub>) = Cond b (mark-guards f c<sub>1</sub>) (mark-guards f c<sub>2</sub>)* |  
*mark-guards f (While b c) = While b (mark-guards f c)* |  
*mark-guards f (Call p) = Call p* |  
*mark-guards f (DynCom c) = DynCom (λs. (mark-guards f (c s)))* |  
*mark-guards f (Guard f' g c) = Guard f g (mark-guards f c)* |  
*mark-guards f Throw = Throw* |  
*mark-guards f (Catch c<sub>1</sub> c<sub>2</sub>) = Catch (mark-guards f c<sub>1</sub>) (mark-guards f c<sub>2</sub>)*

**lemma** *mark-guards-raise*: *mark-guards f (raise g) = raise g*  
**by** (*simp add: raise-def*)

**lemma** *mark-guards-condCatch [simp]*:  
*mark-guards f (condCatch c1 b c2) =*  
*condCatch (mark-guards f c1) b (mark-guards f c2)*  
**by** (*simp add: condCatch-def*)

**lemma** *mark-guards-bind [simp]*:  
*mark-guards f (bind e c) = bind e (λv. mark-guards f (c v))*  
**by** (*simp add: bind-def*)

**lemma** *mark-guards-bseq [simp]*:  
*mark-guards f (bseq c1 c2) = bseq (mark-guards f c1) (mark-guards f c2)*  
**by** (*simp add: bseq-def*)

**lemma** *mark-guards-block-exn [simp]*:  
*mark-guards f (block-exn init bdy return result-exn c) =*  
*block-exn init (mark-guards f bdy) return result-exn (λs t. mark-guards f (c s*  
*t))*  
**by** (*simp add: block-exn-def*)

**lemma** *mark-guards-block [simp]*:  
*mark-guards f (block init bdy return c) =*  
*block init (mark-guards f bdy) return (λs t. mark-guards f (c s t))*  
**by** (*simp add: block-def*)

**lemma** *mark-guards-call* [*simp*]:  
 $mark-guards\ f\ (call\ init\ p\ return\ c) =$   
 $call\ init\ p\ return\ (\lambda s\ t.\ mark-guards\ f\ (c\ s\ t))$   
**by** (*simp add: call-def*)

**lemma** *mark-guards-call-exn* [*simp*]:  
 $mark-guards\ f\ (call-exn\ init\ p\ return\ result-exn\ c) =$   
 $call-exn\ init\ p\ return\ result-exn\ (\lambda s\ t.\ mark-guards\ f\ (c\ s\ t))$   
**by** (*simp add: call-exn-def*)

**lemma** *mark-guards-dynCall* [*simp*]:  
 $mark-guards\ f\ (dynCall\ init\ p\ return\ c) =$   
 $dynCall\ init\ p\ return\ (\lambda s\ t.\ mark-guards\ f\ (c\ s\ t))$   
**by** (*simp add: dynCall-def*)

**lemma** *mark-guards-guards* [*simp*]:  
 $mark-guards\ f\ (guards\ gs\ c) = guards\ (map\ (\lambda(f',g).\ (f,g))\ gs)\ (mark-guards\ f\ c)$   
**by** (*induct gs*) *auto*

**lemma** *mark-guards-dynCall-exn* [*simp*]:  
 $mark-guards\ f\ (dynCall-exn\ f'\ g\ init\ p\ return\ result-exn\ c) =$   
 $dynCall-exn\ f'\ g\ init\ p\ return\ result-exn\ (\lambda s\ t.\ mark-guards\ f\ (c\ s\ t))$   
**by** (*simp add: dynCall-exn-def maybe-guard-def*)

**lemma** *mark-guards-fcall* [*simp*]:  
 $mark-guards\ f\ (fcall\ init\ p\ return\ result\ c) =$   
 $fcall\ init\ p\ return\ result\ (\lambda v.\ mark-guards\ f\ (c\ v))$   
**by** (*simp add: fcall-def*)

**lemma** *mark-guards-switch* [*simp*]:  
 $mark-guards\ f\ (switch\ v\ vs) =$   
 $switch\ v\ (map\ (\lambda(V,c).\ (V,mark-guards\ f\ c))\ vs)$   
**by** (*induct vs*) *auto*

**lemma** *mark-guards-guaranteeStrip* [*simp*]:  
 $mark-guards\ f\ (guaranteeStrip\ f'\ g\ c) = guaranteeStrip\ f\ g\ (mark-guards\ f\ c)$   
**by** (*simp add: guaranteeStrip-def*)

**lemma** *mark-guards-while* [*simp*]:  
 $mark-guards\ f\ (while\ gs\ b\ c) =$   
 $while\ (map\ (\lambda(f',g).\ (f,g))\ gs)\ b\ (mark-guards\ f\ c)$   
**by** (*simp add: while-def*)

**lemma** *mark-guards-whileAnno* [*simp*]:  
 $mark-guards\ f\ (whileAnno\ b\ I\ V\ c) = whileAnno\ b\ I\ V\ (mark-guards\ f\ c)$   
**by** (*simp add: whileAnno-def while-def*)

**lemma** *mark-guards-whileAnnoG* [*simp*]:

$\text{mark-guards } f \text{ (whileAnnoG } gs \text{ b I V c) =}$   
 $\text{whileAnnoG (map } (\lambda(f',g). (f,g)) \text{ gs) b I V (mark-guards } f \text{ c)}$   
**by** (*simp add: whileAnno-def whileAnnoG-def while-def*)

**lemma** *mark-guards-specAnno* [*simp*]:  
 $\text{mark-guards } f \text{ (specAnno } P \text{ c Q A) =}$   
 $\text{specAnno } P \text{ (}\lambda s. \text{ mark-guards } f \text{ (c undefined)) Q A}$   
**by** (*simp add: specAnno-def*)

**lemmas** *mark-guards-simps = mark-guards.simps mark-guards-raise*  
*mark-guards-condCatch mark-guards-bind mark-guards-bseq mark-guards-block*  
*mark-guards-dynCall mark-guards-fcall mark-guards-switch*  
*mark-guards-guaranteeStrip guaranteeStripPair-split-conv mark-guards-guards*  
*mark-guards-while mark-guards-whileAnno mark-guards-whileAnnoG*  
*mark-guards-specAnno*

**definition** *is-Guard*:: ('s,'p,'f) com  $\Rightarrow$  bool  
**where** *is-Guard* c = (case c of Guard f g c'  $\Rightarrow$  True | -  $\Rightarrow$  False)

**lemma** *is-Guard-basic-simps* [*simp*]:  
 $\text{is-Guard (guards (pg\# pgs) c) = True}$   
 $\text{is-Guard Skip = False}$   
 $\text{is-Guard (Basic f) = False}$   
 $\text{is-Guard (Spec r) = False}$   
 $\text{is-Guard (Seq c1 c2) = False}$   
 $\text{is-Guard (Cond b c1 c2) = False}$   
 $\text{is-Guard (While b c) = False}$   
 $\text{is-Guard (Call p) = False}$   
 $\text{is-Guard (DynCom C) = False}$   
 $\text{is-Guard (Guard F g c) = True}$   
 $\text{is-Guard (Throw) = False}$   
 $\text{is-Guard (Catch c1 c2) = False}$   
 $\text{is-Guard (raise f) = False}$   
 $\text{is-Guard (condCatch c1 b c2) = False}$   
 $\text{is-Guard (bind e cv) = False}$   
 $\text{is-Guard (bseq c1 c2) = False}$   
 $\text{is-Guard (block-ern init bdy return result-ern cont) = False}$   
 $\text{is-Guard (block init bdy return cont) = False}$   
 $\text{is-Guard (call init p return cont) = False}$   
 $\text{is-Guard (dynCall init P return cont) = False}$   
 $\text{is-Guard (call-ern init p return result-ern cont) = False}$   
 $\text{is-Guard (dynCall-ern f UNIV init P return result-ern cont) = False}$   
 $\text{is-Guard (fcall init p return result cont') = False}$   
 $\text{is-Guard (whileAnno b I V c) = False}$   
 $\text{is-Guard (guaranteeStrip F g c) = True}$   
**by** (*auto simp add: is-Guard-def raise-def condCatch-def bind-def bseq-def*  
*block-def block-ern-def call-def dynCall-def call-ern-def dynCall-ern-def*  
*fcall-def whileAnno-def guaranteeStrip-def*)

**lemma** *is-Guard-switch* [simp]:  
*is-Guard* (switch  $v$   $Vc$ ) = *False*  
 by (induct  $Vc$ ) auto

**lemmas** *is-Guard-simps* = *is-Guard-basic-simps is-Guard-switch*

**primrec** *dest-Guard*:: ( $'s, 'p, 'f$ ) *com*  $\Rightarrow$  ( $'f \times 's$  *set*  $\times$  ( $'s, 'p, 'f$ ) *com*)  
 where *dest-Guard* (*Guard*  $f$   $g$   $c$ ) = ( $f, g, c$ )

**lemma** *dest-Guard-guaranteeStrip* [simp]: *dest-Guard* (*guaranteeStrip*  $f$   $g$   $c$ ) =  
 ( $f, g, c$ )  
 by (simp add: *guaranteeStrip-def*)

**lemmas** *dest-Guard-simps* = *dest-Guard.simps dest-Guard-guaranteeStrip*

### 2.3.4 Merging Guards: *merge-guards*

**primrec** *merge-guards*:: ( $'s, 'p, 'f$ ) *com*  $\Rightarrow$  ( $'s, 'p, 'f$ ) *com*  
 where

*merge-guards* *Skip* = *Skip* |  
*merge-guards* (*Basic*  $g$ ) = *Basic*  $g$  |  
*merge-guards* (*Spec*  $r$ ) = *Spec*  $r$  |  
*merge-guards* (*Seq*  $c_1$   $c_2$ ) = (*Seq* (*merge-guards*  $c_1$ ) (*merge-guards*  $c_2$ )) |  
*merge-guards* (*Cond*  $b$   $c_1$   $c_2$ ) = *Cond*  $b$  (*merge-guards*  $c_1$ ) (*merge-guards*  $c_2$ ) |  
*merge-guards* (*While*  $b$   $c$ ) = *While*  $b$  (*merge-guards*  $c$ ) |  
*merge-guards* (*Call*  $p$ ) = *Call*  $p$  |  
*merge-guards* (*DynCom*  $c$ ) = *DynCom* ( $\lambda s. (*merge-guards* (c s))$ ) |

*merge-guards* (*Guard*  $f$   $g$   $c$ ) =  
 (let  $c' = (*merge-guards* c)$   
 in if *is-Guard*  $c'$   
 then let ( $f', g', c''$ ) = *dest-Guard*  $c'$   
 in if  $f=f'$  then *Guard*  $f$  ( $g \cap g'$ )  $c''$   
 else *Guard*  $f$   $g$  (*Guard*  $f'$   $g'$   $c''$ )  
 else *Guard*  $f$   $g$   $c'$ ) |  
*merge-guards* *Throw* = *Throw* |  
*merge-guards* (*Catch*  $c_1$   $c_2$ ) = *Catch* (*merge-guards*  $c_1$ ) (*merge-guards*  $c_2$ )

**lemma** *merge-guards-res-Skip*: *merge-guards*  $c = \textit{Skip} \Longrightarrow c = \textit{Skip}$   
 by (cases  $c$ ) (auto split: *com.splits if-split-asm simp add: is-Guard-def Let-def*)

**lemma** *merge-guards-res-Basic*: *merge-guards*  $c = \textit{Basic} f \Longrightarrow c = \textit{Basic} f$   
 by (cases  $c$ ) (auto split: *com.splits if-split-asm simp add: is-Guard-def Let-def*)

**lemma** *merge-guards-res-Spec*: *merge-guards*  $c = \textit{Spec} r \Longrightarrow c = \textit{Spec} r$   
 by (cases  $c$ ) (auto split: *com.splits if-split-asm simp add: is-Guard-def Let-def*)

**lemma** *merge-guards-res-Seq*: *merge-guards*  $c = \textit{Seq} c_1 c_2 \Longrightarrow$



$\exists c1' c2'. c = \text{Seq } c1' c2' \wedge \text{merge-guards } c1' = c1 \wedge \text{merge-guards } c2' = c2$   
**by** (cases c) (auto split: com.splits if-split-asm simp add: is-Guard-def Let-def)

**lemma** merge-guards-res-Cond:  $\text{merge-guards } c = \text{Cond } b c1 c2 \implies$   
 $\exists c1' c2'. c = \text{Cond } b c1' c2' \wedge \text{merge-guards } c1' = c1 \wedge \text{merge-guards } c2' = c2$   
**by** (cases c) (auto split: com.splits if-split-asm simp add: is-Guard-def Let-def)

**lemma** merge-guards-res-While:  $\text{merge-guards } c = \text{While } b c' \implies$   
 $\exists c''. c = \text{While } b c'' \wedge \text{merge-guards } c'' = c'$   
**by** (cases c) (auto split: com.splits if-split-asm simp add: is-Guard-def Let-def)

**lemma** merge-guards-res-Call:  $\text{merge-guards } c = \text{Call } p \implies c = \text{Call } p$   
**by** (cases c) (auto split: com.splits if-split-asm simp add: is-Guard-def Let-def)

**lemma** merge-guards-res-DynCom:  $\text{merge-guards } c = \text{DynCom } c' \implies$   
 $\exists c''. c = \text{DynCom } c'' \wedge (\lambda s. (\text{merge-guards } (c'' s))) = c'$   
**by** (cases c) (auto split: com.splits if-split-asm simp add: is-Guard-def Let-def)

**lemma** merge-guards-res-Throw:  $\text{merge-guards } c = \text{Throw} \implies c = \text{Throw}$   
**by** (cases c) (auto split: com.splits if-split-asm simp add: is-Guard-def Let-def)

**lemma** merge-guards-res-Catch:  $\text{merge-guards } c = \text{Catch } c1 c2 \implies$   
 $\exists c1' c2'. c = \text{Catch } c1' c2' \wedge \text{merge-guards } c1' = c1 \wedge \text{merge-guards } c2' = c2$   
**by** (cases c) (auto split: com.splits if-split-asm simp add: is-Guard-def Let-def)

**lemma** merge-guards-res-Guard:  
 $\text{merge-guards } c = \text{Guard } f g c' \implies \exists c'' f' g'. c = \text{Guard } f' g' c''$   
**by** (cases c) (auto split: com.splits if-split-asm simp add: is-Guard-def Let-def)

**lemmas** merge-guards-res-simps = merge-guards-res-Skip merge-guards-res-Basic  
merge-guards-res-Spec merge-guards-res-Seq merge-guards-res-Cond  
merge-guards-res-While merge-guards-res-Call  
merge-guards-res-DynCom merge-guards-res-Throw merge-guards-res-Catch  
merge-guards-res-Guard

**lemma** merge-guards-guards-empty:  $\text{merge-guards } (\text{guards } [] c) = \text{merge-guards } c$   
**by** (simp)

**lemma** merge-guards-raise:  $\text{merge-guards } (\text{raise } g) = \text{raise } g$   
**by** (simp add: raise-def)

**lemma** merge-guards-condCatch [simp]:  
 $\text{merge-guards } (\text{condCatch } c1 b c2) =$   
 $\text{condCatch } (\text{merge-guards } c1) b (\text{merge-guards } c2)$   
**by** (simp add: condCatch-def)

**lemma** merge-guards-bind [simp]:  
 $\text{merge-guards } (\text{bind } e c) = \text{bind } e (\lambda v. \text{merge-guards } (c v))$

**by** (*simp add: bind-def*)

**lemma** *merge-guards-bseq* [*simp*]:  
 $merge-guards (bseq\ c1\ c2) = bseq (merge-guards\ c1) (merge-guards\ c2)$   
**by** (*simp add: bseq-def*)

**lemma** *merge-guards-block-expr* [*simp*]:  
 $merge-guards (block-expr\ init\ bdy\ return\ result-expr\ c) =$   
 $block-expr\ init (merge-guards\ bdy)\ return\ result-expr (\lambda s\ t.\ merge-guards (c\ s\ t))$   
**by** (*simp add: block-expr-def*)

**lemma** *merge-guards-block* [*simp*]:  
 $merge-guards (block\ init\ bdy\ return\ c) =$   
 $block\ init (merge-guards\ bdy)\ return (\lambda s\ t.\ merge-guards (c\ s\ t))$   
**by** (*simp add: block-def*)

**lemma** *merge-guards-call* [*simp*]:  
 $merge-guards (call\ init\ p\ return\ c) =$   
 $call\ init\ p\ return (\lambda s\ t.\ merge-guards (c\ s\ t))$   
**by** (*simp add: call-def*)

**lemma** *merge-guards-call-expr* [*simp*]:  
 $merge-guards (call-expr\ init\ p\ return\ result-expr\ c) =$   
 $call-expr\ init\ p\ return\ result-expr (\lambda s\ t.\ merge-guards (c\ s\ t))$   
**by** (*simp add: call-expr-def*)

**lemma** *merge-guards-dynCall* [*simp*]:  
 $merge-guards (dynCall\ init\ p\ return\ c) =$   
 $dynCall\ init\ p\ return (\lambda s\ t.\ merge-guards (c\ s\ t))$   
**by** (*simp add: dynCall-def*)

**lemma** *merge-guards-fcall* [*simp*]:  
 $merge-guards (fcall\ init\ p\ return\ result\ c) =$   
 $fcall\ init\ p\ return\ result (\lambda v.\ merge-guards (c\ v))$   
**by** (*simp add: fcall-def*)

**lemma** *merge-guards-switch* [*simp*]:  
 $merge-guards (switch\ v\ vs) =$   
 $switch\ v (map (\lambda (V,c). (V, merge-guards\ c))\ vs)$   
**by** (*induct vs*) *auto*

**lemma** *merge-guards-guaranteeStrip* [*simp*]:  
 $merge-guards (guaranteeStrip\ f\ g\ c) =$   
 $(let\ c' = (merge-guards\ c)$   
 $in\ if\ is-Guard\ c'$   
 $then\ let\ (f',g',c') = dest-Guard\ c'$   
 $in\ if\ f=f'\ then\ Guard\ f\ (g \cap g')\ c'$   
 $else\ Guard\ f\ g\ (Guard\ f'\ g'\ c')$   
 $else\ Guard\ f\ g\ c')$

**by** (*simp add: guaranteeStrip-def*)

**lemma** *merge-guards-whileAnno* [*simp*]:  
*merge-guards (whileAnno b I V c) = whileAnno b I V (merge-guards c)*  
**by** (*simp add: whileAnno-def while-def*)

**lemma** *merge-guards-specAnno* [*simp*]:  
*merge-guards (specAnno P c Q A) =*  
*specAnno P ( $\lambda s.$  merge-guards (c undefined)) Q A*  
**by** (*simp add: specAnno-def*)

*merge-guards* for guard-lists as in *guards*, *while* and *whileAnnoG* may have funny effects since the guard-list has to be merged with the body statement too.

**lemmas** *merge-guards-simps = merge-guards.simps merge-guards-raise*  
*merge-guards-condCatch merge-guards-bind merge-guards-bseq merge-guards-block*  
*merge-guards-dynCall merge-guards-fcall merge-guards-switch*  
*merge-guards-block-exn merge-guards-call-exn*  
*merge-guards-guaranteeStrip merge-guards-whileAnno merge-guards-specAnno*

**primrec** *noguards:: ('s,'p,'f) com  $\Rightarrow$  bool*

**where**

*noguards Skip = True |*  
*noguards (Basic f) = True |*  
*noguards (Spec r) = True |*  
*noguards (Seq c<sub>1</sub> c<sub>2</sub>) = (noguards c<sub>1</sub>  $\wedge$  noguards c<sub>2</sub>) |*  
*noguards (Cond b c<sub>1</sub> c<sub>2</sub>) = (noguards c<sub>1</sub>  $\wedge$  noguards c<sub>2</sub>) |*  
*noguards (While b c) = (noguards c) |*  
*noguards (Call p) = True |*  
*noguards (DynCom c) = ( $\forall s.$  noguards (c s)) |*  
*noguards (Guard f g c) = False |*  
*noguards Throw = True |*  
*noguards (Catch c<sub>1</sub> c<sub>2</sub>) = (noguards c<sub>1</sub>  $\wedge$  noguards c<sub>2</sub>)*

**lemma** *noguards-strip-guards: noguards (strip-guards UNIV c)*  
**by** (*induct c*) *auto*

**primrec** *nothrows:: ('s,'p,'f) com  $\Rightarrow$  bool*

**where**

*nothrows Skip = True |*  
*nothrows (Basic f) = True |*  
*nothrows (Spec r) = True |*  
*nothrows (Seq c<sub>1</sub> c<sub>2</sub>) = (nothrows c<sub>1</sub>  $\wedge$  nothrows c<sub>2</sub>) |*  
*nothrows (Cond b c<sub>1</sub> c<sub>2</sub>) = (nothrows c<sub>1</sub>  $\wedge$  nothrows c<sub>2</sub>) |*  
*nothrows (While b c) = nothrows c |*  
*nothrows (Call p) = True |*  
*nothrows (DynCom c) = ( $\forall s.$  nothrows (c s)) |*  
*nothrows (Guard f g c) = nothrows c |*  
*nothrows Throw = False |*

$nothrows (Catch\ c_1\ c_2) = (nothrows\ c_1 \wedge nothrows\ c_2)$

### 2.3.5 Intersecting Guards: $c_1 \cap_g c_2$

**inductive-set**  $com-rel :: ('s, 'p, 'f)\ com \times ('s, 'p, 'f)\ com) set$

**where**

$(c_1, Seq\ c_1\ c_2) \in com-rel$   
 $| (c_2, Seq\ c_1\ c_2) \in com-rel$   
 $| (c_1, Cond\ b\ c_1\ c_2) \in com-rel$   
 $| (c_2, Cond\ b\ c_1\ c_2) \in com-rel$   
 $| (c, While\ b\ c) \in com-rel$   
 $| (c\ x, DynCom\ c) \in com-rel$   
 $| (c, Guard\ f\ g\ c) \in com-rel$   
 $| (c_1, Catch\ c_1\ c_2) \in com-rel$   
 $| (c_2, Catch\ c_1\ c_2) \in com-rel$

**inductive-cases**  $com-rel-elim-cases:$

$(c, Skip) \in com-rel$   
 $(c, Basic\ f) \in com-rel$   
 $(c, Spec\ r) \in com-rel$   
 $(c, Seq\ c_1\ c_2) \in com-rel$   
 $(c, Cond\ b\ c_1\ c_2) \in com-rel$   
 $(c, While\ b\ c_1) \in com-rel$   
 $(c, Call\ p) \in com-rel$   
 $(c, DynCom\ c_1) \in com-rel$   
 $(c, Guard\ f\ g\ c_1) \in com-rel$   
 $(c, Throw) \in com-rel$   
 $(c, Catch\ c_1\ c_2) \in com-rel$

**lemma**  $wf-com-rel: wf\ com-rel$

**apply**  $(rule\ wfUNIVI)$

**apply**  $(induct-tac\ x)$

**apply**  $(erule\ allE, erule\ mp, (rule\ allI\ impI)+, erule\ com-rel-elim-cases)$

**apply**  $(erule\ allE, erule\ mp, (rule\ allI\ impI)+, erule\ com-rel-elim-cases)$

**apply**  $(erule\ allE, erule\ mp, (rule\ allI\ impI)+, erule\ com-rel-elim-cases,$

$simp, simp)$

**apply**  $(erule\ allE, erule\ mp, (rule\ allI\ impI)+, erule\ com-rel-elim-cases,$

$simp, simp)$

**apply**  $(erule\ allE, erule\ mp, (rule\ allI\ impI)+, erule\ com-rel-elim-cases, simp)$

**apply**  $(erule\ allE, erule\ mp, (rule\ allI\ impI)+, erule\ com-rel-elim-cases)$

**apply**  $(erule\ allE, erule\ mp, (rule\ allI\ impI)+, erule\ com-rel-elim-cases, simp)$

**apply**  $(erule\ allE, erule\ mp, (rule\ allI\ impI)+, erule\ com-rel-elim-cases, simp)$

**apply**  $(erule\ allE, erule\ mp, (rule\ allI\ impI)+, erule\ com-rel-elim-cases)$

**apply**  $(erule\ allE, erule\ mp, (rule\ allI\ impI)+, erule\ com-rel-elim-cases, simp, simp)$

**done**

**consts**  $inter-guards :: ('s, 'p, 'f)\ com \times ('s, 'p, 'f)\ com \Rightarrow ('s, 'p, 'f)\ com\ option$

### abbreviation

$inter\text{-guards}\text{-syntax} :: ('s, 'p, 'f) com \Rightarrow ('s, 'p, 'f) com \Rightarrow ('s, 'p, 'f) com option$

$(\langle \cdot \rangle \cap_g \rightarrow [20, 20] 19)$

**where**  $c \cap_g d == inter\text{-guards} (c, d)$

### recdef $inter\text{-guards} inv\text{-image} com\text{-rel} fst$

$(Skip \cap_g Skip) = Some Skip$

$(Basic f1 \cap_g Basic f2) = (if f1 = f2 then Some (Basic f1) else None)$

$(Spec r1 \cap_g Spec r2) = (if r1 = r2 then Some (Spec r1) else None)$

$(Seq a1 a2 \cap_g Seq b1 b2) =$

$(case a1 \cap_g b1 of$

$None \Rightarrow None$

$| Some c1 \Rightarrow (case a2 \cap_g b2 of$

$None \Rightarrow None$

$| Some c2 \Rightarrow Some (Seq c1 c2)))$

$(Cond cnd1 t1 e1 \cap_g Cond cnd2 t2 e2) =$

$(if cnd1 = cnd2$

$then (case t1 \cap_g t2 of$

$None \Rightarrow None$

$| Some t \Rightarrow (case e1 \cap_g e2 of$

$None \Rightarrow None$

$| Some e \Rightarrow Some (Cond cnd1 t e)))$

$else None)$

$(While cnd1 c1 \cap_g While cnd2 c2) =$

$(if cnd1 = cnd2$

$then (case c1 \cap_g c2 of$

$None \Rightarrow None$

$| Some c \Rightarrow Some (While cnd1 c))$

$else None)$

$(Call p1 \cap_g Call p2) =$

$(if p1 = p2$

$then Some (Call p1)$

$else None)$

$(DynCom P1 \cap_g DynCom P2) =$

$(if (\forall s. (P1 s \cap_g P2 s) \neq None)$

$then Some (DynCom (\lambda s. the (P1 s \cap_g P2 s)))$

$else None)$

$(Guard m1 g1 c1 \cap_g Guard m2 g2 c2) =$

$(if m1 = m2 then$

$(case c1 \cap_g c2 of$

$None \Rightarrow None$

$| Some c \Rightarrow Some (Guard m1 (g1 \cap g2) c))$

$else None)$

$(Throw \cap_g Throw) = Some Throw$

$(Catch a1 a2 \cap_g Catch b1 b2) =$

$(case a1 \cap_g b1 of$

$None \Rightarrow None$

$| Some c1 \Rightarrow (case a2 \cap_g b2 of$

$None \Rightarrow None$

|  $\text{Some } c2 \Rightarrow \text{Some } (\text{Catch } c1 \ c2))$ )  
 $(c \cap_g d) = \text{None}$   
**(hints** *cong add: option.case-cong if-cong*  
*recdef-wf: wf-com-rel simp: com-rel.intros*)

**lemma** *inter-guards-strip-eq*:  
 $\bigwedge c. (c1 \cap_g c2) = \text{Some } c \implies$   
 $(\text{strip-guards UNIV } c = \text{strip-guards UNIV } c1) \wedge$   
 $(\text{strip-guards UNIV } c = \text{strip-guards UNIV } c2)$   
**apply** (*induct c1 c2 rule: inter-guards.induct*)  
**prefer** 8  
**apply** (*simp split: if-split-asm*)  
**apply** *hypsubst*  
**apply** *simp*  
**apply** (*rule ext*)  
**apply** (*erule-tac x=s in allE, erule exE*)  
**apply** (*erule-tac x=s in allE*)  
**apply** *fastforce*  
**apply** (*fastforce split: option.splits if-split-asm*)+  
**done**

**lemma** *inter-guards-sym*:  $\bigwedge c. (c1 \cap_g c2) = \text{Some } c \implies (c2 \cap_g c1) = \text{Some } c$   
**apply** (*induct c1 c2 rule: inter-guards.induct*)  
**apply** (*simp-all*)  
**prefer** 7  
**apply** (*simp split: if-split-asm add: not-None-eq*)  
**apply** (*rule conjI*)  
**apply** (*clarsimp*)  
**apply** (*rule ext*)  
**apply** (*erule-tac x=s in allE*)+  
**apply** *fastforce*  
**apply** *fastforce*  
**apply** (*fastforce split: option.splits if-split-asm*)+  
**done**

**lemma** *inter-guards-Skip*:  $(\text{Skip} \cap_g c2) = \text{Some } c = (c2 = \text{Skip} \wedge c = \text{Skip})$   
**by** (*cases c2*) *auto*

**lemma** *inter-guards-Basic*:  
 $((\text{Basic } f) \cap_g c2) = \text{Some } c = (c2 = \text{Basic } f \wedge c = \text{Basic } f)$   
**by** (*cases c2*) *auto*

**lemma** *inter-guards-Spec*:  
 $((\text{Spec } r) \cap_g c2) = \text{Some } c = (c2 = \text{Spec } r \wedge c = \text{Spec } r)$   
**by** (*cases c2*) *auto*

**lemma** *inter-guards-Seq*:  
 $(\text{Seq } a1 \ a2 \cap_g c2) = \text{Some } c =$

$(\exists b1\ b2\ d1\ d2. c2 = \text{Seq } b1\ b2 \wedge (a1 \cap_g b1) = \text{Some } d1 \wedge$   
 $(a2 \cap_g b2) = \text{Some } d2 \wedge c = \text{Seq } d1\ d2)$   
**by** (cases c2) (auto split: option.splits)

**lemma** *inter-guards-Cond*:  
 $(\text{Cond } cnd\ t1\ e1 \cap_g c2) = \text{Some } c =$   
 $(\exists t2\ e2\ t\ e. c2 = \text{Cond } cnd\ t2\ e2 \wedge (t1 \cap_g t2) = \text{Some } t \wedge$   
 $(e1 \cap_g e2) = \text{Some } e \wedge c = \text{Cond } cnd\ t\ e)$   
**by** (cases c2) (auto split: option.splits)

**lemma** *inter-guards-While*:  
 $(\text{While } cnd\ bdy1 \cap_g c2) = \text{Some } c =$   
 $(\exists bdy2\ bdy. c2 = \text{While } cnd\ bdy2 \wedge (bdy1 \cap_g bdy2) = \text{Some } bdy \wedge$   
 $c = \text{While } cnd\ bdy)$   
**by** (cases c2) (auto split: option.splits if-split-asm)

**lemma** *inter-guards-Call*:  
 $(\text{Call } p \cap_g c2) = \text{Some } c =$   
 $(c2 = \text{Call } p \wedge c = \text{Call } p)$   
**by** (cases c2) (auto split: if-split-asm)

**lemma** *inter-guards-DynCom*:  
 $(\text{DynCom } f1 \cap_g c2) = \text{Some } c =$   
 $(\exists f2. c2 = \text{DynCom } f2 \wedge (\forall s. ((f1\ s) \cap_g (f2\ s)) \neq \text{None}) \wedge$   
 $c = \text{DynCom } (\lambda s. \text{the } ((f1\ s) \cap_g (f2\ s))))$   
**by** (cases c2) (auto split: if-split-asm)

**lemma** *inter-guards-Guard*:  
 $(\text{Guard } f\ g1\ bdy1 \cap_g c2) = \text{Some } c =$   
 $(\exists g2\ bdy2\ bdy. c2 = \text{Guard } f\ g2\ bdy2 \wedge (bdy1 \cap_g bdy2) = \text{Some } bdy \wedge$   
 $c = \text{Guard } f\ (g1 \cap_g g2)\ bdy)$   
**by** (cases c2) (auto split: option.splits)

**lemma** *inter-guards-Throw*:  
 $(\text{Throw } \cap_g c2) = \text{Some } c = (c2 = \text{Throw} \wedge c = \text{Throw})$   
**by** (cases c2) auto

**lemma** *inter-guards-Catch*:  
 $(\text{Catch } a1\ a2 \cap_g c2) = \text{Some } c =$   
 $(\exists b1\ b2\ d1\ d2. c2 = \text{Catch } b1\ b2 \wedge (a1 \cap_g b1) = \text{Some } d1 \wedge$   
 $(a2 \cap_g b2) = \text{Some } d2 \wedge c = \text{Catch } d1\ d2)$   
**by** (cases c2) (auto split: option.splits)

**lemmas** *inter-guards-simps = inter-guards-Skip inter-guards-Basic inter-guards-Spec*  
*inter-guards-Seq inter-guards-Cond inter-guards-While inter-guards-Call*  
*inter-guards-DynCom inter-guards-Guard inter-guards-Throw*  
*inter-guards-Catch*

### 2.3.6 Subset on Guards: $c_1 \subseteq_g c_2$

**inductive** *subseteq-guards* :: ('s,'p,'f) com  $\Rightarrow$  ('s,'p,'f) com  $\Rightarrow$  bool

( $\langle - \subseteq_g \rightarrow [20,20] \ 19$ ) **where**  
*Skip*  $\subseteq_g$  *Skip*  
 $| f1 = f2 \Longrightarrow \text{Basic } f1 \subseteq_g \text{Basic } f2$   
 $| r1 = r2 \Longrightarrow \text{Spec } r1 \subseteq_g \text{Spec } r2$   
 $| a1 \subseteq_g b1 \Longrightarrow a2 \subseteq_g b2 \Longrightarrow \text{Seq } a1 \ a2 \subseteq_g \text{Seq } b1 \ b2$   
 $| \text{cnd1} = \text{cnd2} \Longrightarrow t1 \subseteq_g t2 \Longrightarrow e1 \subseteq_g e2 \Longrightarrow \text{Cond } \text{cnd1 } t1 \ e1 \subseteq_g \text{Cond } \text{cnd2 } t2 \ e2$   
 $| \text{cnd1} = \text{cnd2} \Longrightarrow c1 \subseteq_g c2 \Longrightarrow \text{While } \text{cnd1 } c1 \subseteq_g \text{While } \text{cnd2 } c2$   
 $| p1 = p2 \Longrightarrow \text{Call } p1 \subseteq_g \text{Call } p2$   
 $| (\bigwedge s. P1 \ s \subseteq_g P2 \ s) \Longrightarrow \text{DynCom } P1 \subseteq_g \text{DynCom } P2$   
 $| m1 = m2 \Longrightarrow g1 = g2 \Longrightarrow c1 \subseteq_g c2 \Longrightarrow \text{Guard } m1 \ g1 \ c1 \subseteq_g \text{Guard } m2 \ g2 \ c2$   
 $| c1 \subseteq_g c2 \Longrightarrow c1 \subseteq_g \text{Guard } m2 \ g2 \ c2$   
 $| \text{Throw} \subseteq_g \text{Throw}$   
 $| a1 \subseteq_g b1 \Longrightarrow a2 \subseteq_g b2 \Longrightarrow \text{Catch } a1 \ a2 \subseteq_g \text{Catch } b1 \ b2$

**lemma** *subseteq-guards-Skip*:

$c = \text{Skip}$  **if**  $c \subseteq_g \text{Skip}$   
**using** *that by cases*

**lemma** *subseteq-guards-Basic*:

$c = \text{Basic } f$  **if**  $c \subseteq_g \text{Basic } f$   
**using** *that by cases simp*

**lemma** *subseteq-guards-Spec*:

$c = \text{Spec } r$  **if**  $c \subseteq_g \text{Spec } r$   
**using** *that by cases simp*

**lemma** *subseteq-guards-Seq*:

$\exists c1' \ c2'. c = \text{Seq } c1' \ c2' \wedge (c1' \subseteq_g c1) \wedge (c2' \subseteq_g c2)$  **if**  $c \subseteq_g \text{Seq } c1 \ c2$   
**using** *that by cases simp*

**lemma** *subseteq-guards-Cond*:

$\exists c1' \ c2'. c = \text{Cond } b \ c1' \ c2' \wedge (c1' \subseteq_g c1) \wedge (c2' \subseteq_g c2)$  **if**  $c \subseteq_g \text{Cond } b \ c1 \ c2$   
**using** *that by cases simp*

**lemma** *subseteq-guards-While*:

$\exists c''. c = \text{While } b \ c'' \wedge (c'' \subseteq_g c')$  **if**  $c \subseteq_g \text{While } b \ c'$   
**using** *that by cases simp*

**lemma** *subseteq-guards-Call*:

$c = \text{Call } p$  **if**  $c \subseteq_g \text{Call } p$   
**using** *that by cases simp*

**lemma** *subseteq-guards-DynCom*:

$\exists C'. c = \text{DynCom } C' \wedge (\forall s. C' \ s \subseteq_g C \ s)$  **if**  $c \subseteq_g \text{DynCom } C$   
**using** *that by cases simp*



**lemma** *subsetq-guards-Guard*:  
 $(c \subseteq_g c') \vee (\exists c''. c = \text{Guard } f \ g \ c'' \wedge (c'' \subseteq_g c'))$  **if**  $c \subseteq_g \text{Guard } f \ g \ c'$   
**using** *that by cases simp-all*

**lemma** *subsetq-guards-Throw*:  
 $c = \text{Throw}$  **if**  $c \subseteq_g \text{Throw}$   
**using** *that by cases*

**lemma** *subsetq-guards-Catch*:  
 $\exists c1' \ c2'. c = \text{Catch } c1' \ c2' \wedge (c1' \subseteq_g c1) \wedge (c2' \subseteq_g c2)$  **if**  $c \subseteq_g \text{Catch } c1 \ c2$   
**using** *that by cases simp*

**lemmas** *subsetq-guardsD = subsetq-guards-Skip subsetq-guards-Basic  
subsetq-guards-Spec subsetq-guards-Seq subsetq-guards-Cond subsetq-guards-While  
subsetq-guards-Call subsetq-guards-DynCom subsetq-guards-Guard  
subsetq-guards-Throw subsetq-guards-Catch*

**lemma** *subsetq-guards-Guard'*:  
 $\exists f' \ b' \ c'. d = \text{Guard } f' \ b' \ c'$  **if**  $\text{Guard } f \ b \ c \subseteq_g d$   
**using** *that by cases auto*

**lemma** *subsetq-guards-refl*:  $c \subseteq_g c$   
**by** (*induct c*) (*auto intro: subsetq-guards.intros*)

**end**

### 3 Big-Step Semantics for Simpl

**theory** *Semantic* **imports** *Language* **begin**

**notation**  
*restrict-map*  $\langle \cdot | \cdot \rangle [90, 91] \ 90$

**datatype**  $(s, f) \ xstate = \text{Normal } 's \ | \ \text{Abrupt } 's \ | \ \text{Fault } 'f \ | \ \text{Stuck}$

**definition** *isAbr*:: $(s, f) \ xstate \Rightarrow \text{bool}$   
**where**  $\text{isAbr } S = (\exists s. S = \text{Abrupt } s)$

**lemma** *isAbr-simps* [*simp*]:  
 $\text{isAbr } (\text{Normal } s) = \text{False}$   
 $\text{isAbr } (\text{Abrupt } s) = \text{True}$   
 $\text{isAbr } (\text{Fault } f) = \text{False}$   
 $\text{isAbr } \text{Stuck} = \text{False}$   
**by** (*auto simp add: isAbr-def*)

**lemma** *isAbrE* [*consumes 1, elim?*]:  $\llbracket \text{isAbr } S; \bigwedge s. S = \text{Abrupt } s \implies P \rrbracket \implies P$

**by** (*auto simp add: isAbr-def*)

**lemma** *not-isAbrD*:

$\neg \text{isAbr } s \implies (\exists s'. s = \text{Normal } s') \vee s = \text{Stuck} \vee (\exists f. s = \text{Fault } f)$   
**by** (*cases s*) *auto*

**definition** *isFault*::  $(s, f) \text{ xstate} \Rightarrow \text{bool}$

**where**  $\text{isFault } S = (\exists f. S = \text{Fault } f)$

**lemma** *isFault-simps* [*simp*]:

$\text{isFault } (\text{Normal } s) = \text{False}$

$\text{isFault } (\text{Abrupt } s) = \text{False}$

$\text{isFault } (\text{Fault } f) = \text{True}$

$\text{isFault } \text{Stuck} = \text{False}$

**by** (*auto simp add: isFault-def*)

**lemma** *isFaultE* [*consumes 1, elim?*]:  $\llbracket \text{isFault } s; \bigwedge f. s = \text{Fault } f \implies P \rrbracket \implies P$

**by** (*auto simp add: isFault-def*)

**lemma** *not-isFault-iff*:  $(\neg \text{isFault } t) = (\forall f. t \neq \text{Fault } f)$

**by** (*auto elim: isFaultE*)

### 3.1 Big-Step Execution: $\Gamma \vdash \langle c, s \rangle \Rightarrow t$

The procedure environment

**type-synonym**  $(s, p, f) \text{ body} = p \Rightarrow (s, p, f) \text{ com option}$

**inductive**

$\text{exec}::[(s, p, f) \text{ body}, (s, p, f) \text{ com}, (s, f) \text{ xstate}, (s, f) \text{ xstate}]$   
 $\implies \text{bool } (\langle \vdash \langle -, - \rangle \Rightarrow \cdot \rangle [60, 20, 98, 98] 89)$

**for**  $\Gamma::(s, p, f) \text{ body}$

**where**

*Skip*:  $\Gamma \vdash \langle \text{Skip}, \text{Normal } s \rangle \Rightarrow \text{Normal } s$

| *Guard*:  $\llbracket s \in g; \Gamma \vdash \langle c, \text{Normal } s \rangle \Rightarrow t \rrbracket$

$\implies$

$\Gamma \vdash \langle \text{Guard } f \ g \ c, \text{Normal } s \rangle \Rightarrow t$

| *GuardFault*:  $s \notin g \implies \Gamma \vdash \langle \text{Guard } f \ g \ c, \text{Normal } s \rangle \Rightarrow \text{Fault } f$

| *FaultProp* [*intro, simp*]:  $\Gamma \vdash \langle c, \text{Fault } f \rangle \Rightarrow \text{Fault } f$

| *Basic*:  $\Gamma \vdash \langle \text{Basic } f, \text{Normal } s \rangle \Rightarrow \text{Normal } (f \ s)$

| *Spec*:  $(s, t) \in r$

$\implies$

$\Gamma \vdash \langle \text{Spec } r, \text{Normal } s \rangle \Rightarrow \text{Normal } t$

| *SpecStuck*:  $\forall t. (s, t) \notin r$

$$\begin{aligned}
& \Longrightarrow \\
& \Gamma \vdash \langle \text{Spec } r, \text{Normal } s \rangle \Rightarrow \text{Stuck} \\
| \text{Seq}: & \llbracket \Gamma \vdash \langle c_1, \text{Normal } s \rangle \Rightarrow s'; \Gamma \vdash \langle c_2, s' \rangle \Rightarrow t \rrbracket \\
& \Longrightarrow \\
& \Gamma \vdash \langle \text{Seq } c_1 \ c_2, \text{Normal } s \rangle \Rightarrow t \\
| \text{CondTrue}: & \llbracket s \in b; \Gamma \vdash \langle c_1, \text{Normal } s \rangle \Rightarrow t \rrbracket \\
& \Longrightarrow \\
& \Gamma \vdash \langle \text{Cond } b \ c_1 \ c_2, \text{Normal } s \rangle \Rightarrow t \\
| \text{CondFalse}: & \llbracket s \notin b; \Gamma \vdash \langle c_2, \text{Normal } s \rangle \Rightarrow t \rrbracket \\
& \Longrightarrow \\
& \Gamma \vdash \langle \text{Cond } b \ c_1 \ c_2, \text{Normal } s \rangle \Rightarrow t \\
| \text{WhileTrue}: & \llbracket s \in b; \Gamma \vdash \langle c, \text{Normal } s \rangle \Rightarrow s'; \Gamma \vdash \langle \text{While } b \ c, s' \rangle \Rightarrow t \rrbracket \\
& \Longrightarrow \\
& \Gamma \vdash \langle \text{While } b \ c, \text{Normal } s \rangle \Rightarrow t \\
| \text{WhileFalse}: & \llbracket s \notin b \rrbracket \\
& \Longrightarrow \\
& \Gamma \vdash \langle \text{While } b \ c, \text{Normal } s \rangle \Rightarrow \text{Normal } s \\
| \text{Call}: & \llbracket \Gamma \ p = \text{Some } \text{bdy}; \Gamma \vdash \langle \text{bdy}, \text{Normal } s \rangle \Rightarrow t \rrbracket \\
& \Longrightarrow \\
& \Gamma \vdash \langle \text{Call } p, \text{Normal } s \rangle \Rightarrow t \\
| \text{CallUndefined}: & \llbracket \Gamma \ p = \text{None} \rrbracket \\
& \Longrightarrow \\
& \Gamma \vdash \langle \text{Call } p, \text{Normal } s \rangle \Rightarrow \text{Stuck} \\
| \text{StuckProp } [\text{intro}, \text{simp}]: & \Gamma \vdash \langle c, \text{Stuck} \rangle \Rightarrow \text{Stuck} \\
| \text{DynCom}: & \llbracket \Gamma \vdash \langle (c \ s), \text{Normal } s \rangle \Rightarrow t \rrbracket \\
& \Longrightarrow \\
& \Gamma \vdash \langle \text{DynCom } c, \text{Normal } s \rangle \Rightarrow t \\
| \text{Throw}: & \Gamma \vdash \langle \text{Throw}, \text{Normal } s \rangle \Rightarrow \text{Abrupt } s \\
| \text{AbruptProp } [\text{intro}, \text{simp}]: & \Gamma \vdash \langle c, \text{Abrupt } s \rangle \Rightarrow \text{Abrupt } s \\
| \text{CatchMatch}: & \llbracket \Gamma \vdash \langle c_1, \text{Normal } s \rangle \Rightarrow \text{Abrupt } s'; \Gamma \vdash \langle c_2, \text{Normal } s' \rangle \Rightarrow t \rrbracket \\
& \Longrightarrow \\
& \Gamma \vdash \langle \text{Catch } c_1 \ c_2, \text{Normal } s \rangle \Rightarrow t \\
| \text{CatchMiss}: & \llbracket \Gamma \vdash \langle c_1, \text{Normal } s \rangle \Rightarrow t; \neg \text{isAbr } t \rrbracket \\
& \Longrightarrow \\
& \Gamma \vdash \langle \text{Catch } c_1 \ c_2, \text{Normal } s \rangle \Rightarrow t
\end{aligned}$$

**inductive-cases** *exec-elim-cases* [cases set]:

$$\begin{aligned}
&\Gamma \langle c, \text{Fault } f \rangle \Rightarrow t \\
&\Gamma \langle c, \text{Stuck} \rangle \Rightarrow t \\
&\Gamma \langle c, \text{Abrupt } s \rangle \Rightarrow t \\
&\Gamma \langle \text{Skip}, s \rangle \Rightarrow t \\
&\Gamma \langle \text{Seq } c1 \ c2, s \rangle \Rightarrow t \\
&\Gamma \langle \text{Guard } f \ g \ c, s \rangle \Rightarrow t \\
&\Gamma \langle \text{Basic } f, s \rangle \Rightarrow t \\
&\Gamma \langle \text{Spec } r, s \rangle \Rightarrow t \\
&\Gamma \langle \text{Cond } b \ c1 \ c2, s \rangle \Rightarrow t \\
&\Gamma \langle \text{While } b \ c, s \rangle \Rightarrow t \\
&\Gamma \langle \text{Call } p, s \rangle \Rightarrow t \\
&\Gamma \langle \text{DynCom } c, s \rangle \Rightarrow t \\
&\Gamma \langle \text{Throw}, s \rangle \Rightarrow t \\
&\Gamma \langle \text{Catch } c1 \ c2, s \rangle \Rightarrow t
\end{aligned}$$

**inductive-cases** *exec-Normal-elim-cases* [cases set]:

$$\begin{aligned}
&\Gamma \langle c, \text{Fault } f \rangle \Rightarrow t \\
&\Gamma \langle c, \text{Stuck} \rangle \Rightarrow t \\
&\Gamma \langle c, \text{Abrupt } s \rangle \Rightarrow t \\
&\Gamma \langle \text{Skip}, \text{Normal } s \rangle \Rightarrow t \\
&\Gamma \langle \text{Guard } f \ g \ c, \text{Normal } s \rangle \Rightarrow t \\
&\Gamma \langle \text{Basic } f, \text{Normal } s \rangle \Rightarrow t \\
&\Gamma \langle \text{Spec } r, \text{Normal } s \rangle \Rightarrow t \\
&\Gamma \langle \text{Seq } c1 \ c2, \text{Normal } s \rangle \Rightarrow t \\
&\Gamma \langle \text{Cond } b \ c1 \ c2, \text{Normal } s \rangle \Rightarrow t \\
&\Gamma \langle \text{While } b \ c, \text{Normal } s \rangle \Rightarrow t \\
&\Gamma \langle \text{Call } p, \text{Normal } s \rangle \Rightarrow t \\
&\Gamma \langle \text{DynCom } c, \text{Normal } s \rangle \Rightarrow t \\
&\Gamma \langle \text{Throw}, \text{Normal } s \rangle \Rightarrow t \\
&\Gamma \langle \text{Catch } c1 \ c2, \text{Normal } s \rangle \Rightarrow t
\end{aligned}$$

**lemma** *exec-block-exn*:

$$\begin{aligned}
&\llbracket \Gamma \langle \text{bdy}, \text{Normal } (\text{init } s) \rangle \Rightarrow \text{Normal } t; \Gamma \langle c \ s \ t, \text{Normal } (\text{return } s \ t) \rangle \Rightarrow u \rrbracket \\
&\implies
\end{aligned}$$

$$\Gamma \langle \text{block-exn } \text{init } \text{bdy } \text{return } \text{result-exn } c, \text{Normal } s \rangle \Rightarrow u$$

**apply** (*unfold block-exn-def*)

**by** (*fastforce intro: exec.intros*)

**lemma** *exec-block*:

$$\begin{aligned}
&\llbracket \Gamma \langle \text{bdy}, \text{Normal } (\text{init } s) \rangle \Rightarrow \text{Normal } t; \Gamma \langle c \ s \ t, \text{Normal } (\text{return } s \ t) \rangle \Rightarrow u \rrbracket \\
&\implies
\end{aligned}$$

$$\Gamma \langle \text{block } \text{init } \text{bdy } \text{return } c, \text{Normal } s \rangle \Rightarrow u$$

**unfolding** *block-def*

**by** (*rule exec-block-exn*)

**lemma** *exec-block-exnAbrupt*:

$$\llbracket \Gamma \langle \text{bdy}, \text{Normal } (\text{init } s) \rangle \Rightarrow \text{Abrupt } t \rrbracket$$

$\Rightarrow$   
 $\Gamma \vdash \langle \text{block-exn init bdy return result-exn } c, \text{Normal } s \rangle \Rightarrow \text{Abrupt } (\text{result-exn } (\text{return } s \ t) \ t)$   
**apply** (*unfold block-exn-def*)  
**by** (*fastforce intro: exec.intros*)

**lemma** *exec-blockAbrupt*:  
 $\llbracket \Gamma \vdash \langle \text{bdy}, \text{Normal } (\text{init } s) \rangle \Rightarrow \text{Abrupt } t \rrbracket$   
 $\Rightarrow$   
 $\Gamma \vdash \langle \text{block init bdy return } c, \text{Normal } s \rangle \Rightarrow \text{Abrupt } (\text{return } s \ t)$   
**unfolding** *block-def*  
**by** (*rule exec-block-exnAbrupt*)

**lemma** *exec-block-exnFault*:  
 $\llbracket \Gamma \vdash \langle \text{bdy}, \text{Normal } (\text{init } s) \rangle \Rightarrow \text{Fault } f \rrbracket$   
 $\Rightarrow$   
 $\Gamma \vdash \langle \text{block-exn init bdy return result-exn } c, \text{Normal } s \rangle \Rightarrow \text{Fault } f$   
**apply** (*unfold block-exn-def*)  
**by** (*fastforce intro: exec.intros*)

**lemma** *exec-blockFault*:  
 $\llbracket \Gamma \vdash \langle \text{bdy}, \text{Normal } (\text{init } s) \rangle \Rightarrow \text{Fault } f \rrbracket$   
 $\Rightarrow$   
 $\Gamma \vdash \langle \text{block init bdy return } c, \text{Normal } s \rangle \Rightarrow \text{Fault } f$   
**unfolding** *block-def*  
**by** (*rule exec-block-exnFault*)

**lemma** *exec-block-exnStuck*:  
 $\llbracket \Gamma \vdash \langle \text{bdy}, \text{Normal } (\text{init } s) \rangle \Rightarrow \text{Stuck} \rrbracket$   
 $\Rightarrow$   
 $\Gamma \vdash \langle \text{block-exn init bdy return result-exn } c, \text{Normal } s \rangle \Rightarrow \text{Stuck}$   
**apply** (*unfold block-exn-def*)  
**by** (*fastforce intro: exec.intros*)

**lemma** *exec-blockStuck*:  
 $\llbracket \Gamma \vdash \langle \text{bdy}, \text{Normal } (\text{init } s) \rangle \Rightarrow \text{Stuck} \rrbracket$   
 $\Rightarrow$   
 $\Gamma \vdash \langle \text{block init bdy return } c, \text{Normal } s \rangle \Rightarrow \text{Stuck}$   
**unfolding** *block-def*  
**by** (*rule exec-block-exnStuck*)

**lemma** *exec-call*:  
 $\llbracket \Gamma \ p = \text{Some } \text{bdy}; \Gamma \vdash \langle \text{bdy}, \text{Normal } (\text{init } s) \rangle \Rightarrow \text{Normal } t; \Gamma \vdash \langle c \ s \ t, \text{Normal } (\text{return } s \ t) \rangle \Rightarrow u \rrbracket$   
 $\Rightarrow$   
 $\Gamma \vdash \langle \text{call init } p \text{ return } c, \text{Normal } s \rangle \Rightarrow u$   
**apply** (*simp add: call-def*)  
**apply** (*rule exec-block*)  
**apply** (*erule (1) Call*)

**apply** *assumption*  
**done**

**lemma** *exec-callAbrupt*:

$\llbracket \Gamma \ p=Some \ bdy; \Gamma \vdash \langle bdy, Normal \ (init \ s) \rangle \Rightarrow \ Abrupt \ t \rrbracket$   
 $\implies$   
 $\Gamma \vdash \langle call \ init \ p \ return \ c, Normal \ s \rangle \Rightarrow \ Abrupt \ (return \ s \ t)$   
**apply** (*simp add: call-def*)  
**apply** (*rule exec-blockAbrupt*)  
**apply** (*erule (1) Call*)  
**done**

**lemma** *exec-callFault*:

$\llbracket \Gamma \ p=Some \ bdy; \Gamma \vdash \langle bdy, Normal \ (init \ s) \rangle \Rightarrow \ Fault \ f \rrbracket$   
 $\implies$   
 $\Gamma \vdash \langle call \ init \ p \ return \ c, Normal \ s \rangle \Rightarrow \ Fault \ f$   
**apply** (*simp add: call-def*)  
**apply** (*rule exec-blockFault*)  
**apply** (*erule (1) Call*)  
**done**

**lemma** *exec-callStuck*:

$\llbracket \Gamma \ p=Some \ bdy; \Gamma \vdash \langle bdy, Normal \ (init \ s) \rangle \Rightarrow \ Stuck \rrbracket$   
 $\implies$   
 $\Gamma \vdash \langle call \ init \ p \ return \ c, Normal \ s \rangle \Rightarrow \ Stuck$   
**apply** (*simp add: call-def*)  
**apply** (*rule exec-blockStuck*)  
**apply** (*erule (1) Call*)  
**done**

**lemma** *exec-callUndefined*:

$\llbracket \Gamma \ p=None \rrbracket$   
 $\implies$   
 $\Gamma \vdash \langle call \ init \ p \ return \ c, Normal \ s \rangle \Rightarrow \ Stuck$   
**apply** (*simp add: call-def*)  
**apply** (*rule exec-blockStuck*)  
**apply** (*erule CallUndefined*)  
**done**

**lemma** *exec-call-exn*:

$\llbracket \Gamma \ p=Some \ bdy; \Gamma \vdash \langle bdy, Normal \ (init \ s) \rangle \Rightarrow \ Normal \ t; \Gamma \vdash \langle c \ s \ t, Normal \ (return \ s \ t) \rangle \Rightarrow \ u \rrbracket$   
 $\implies$   
 $\Gamma \vdash \langle call \ exn \ init \ p \ return \ result \ exn \ c, Normal \ s \rangle \Rightarrow \ u$   
**apply** (*simp add: call-exn-def*)  
**apply** (*rule exec-block-exn*)  
**apply** (*erule (1) Call*)  
**apply** *assumption*  
**done**

**lemma** *exec-call-exnAbrupt*:

$\llbracket \Gamma \ p=Some \ bdy; \Gamma \vdash \langle bdy, Normal \ (init \ s) \rangle \Rightarrow \ Abrupt \ t \rrbracket$   
 $\implies$   
 $\Gamma \vdash \langle call-exn \ init \ p \ return \ result-exn \ c, Normal \ s \rangle \Rightarrow \ Abrupt \ (result-exn \ (return \ s \ t) \ t)$   
**apply** (*simp add: call-exn-def*)  
**apply** (*rule exec-block-exnAbrupt*)  
**apply** (*erule (1) Call*)  
**done**

**lemma** *exec-call-exnFault*:

$\llbracket \Gamma \ p=Some \ bdy; \Gamma \vdash \langle bdy, Normal \ (init \ s) \rangle \Rightarrow \ Fault \ f \rrbracket$   
 $\implies$   
 $\Gamma \vdash \langle call-exn \ init \ p \ return \ result-exn \ c, Normal \ s \rangle \Rightarrow \ Fault \ f$   
**apply** (*simp add: call-exn-def*)  
**apply** (*rule exec-block-exnFault*)  
**apply** (*erule (1) Call*)  
**done**

**lemma** *exec-call-exnStuck*:

$\llbracket \Gamma \ p=Some \ bdy; \Gamma \vdash \langle bdy, Normal \ (init \ s) \rangle \Rightarrow \ Stuck \rrbracket$   
 $\implies$   
 $\Gamma \vdash \langle call-exn \ init \ p \ return \ result-exn \ c, Normal \ s \rangle \Rightarrow \ Stuck$   
**apply** (*simp add: call-exn-def*)  
**apply** (*rule exec-block-exnStuck*)  
**apply** (*erule (1) Call*)  
**done**

**lemma** *exec-call-exnUndefined*:

$\llbracket \Gamma \ p=None \rrbracket$   
 $\implies$   
 $\Gamma \vdash \langle call-exn \ init \ p \ return \ result-exn \ c, Normal \ s \rangle \Rightarrow \ Stuck$   
**apply** (*simp add: call-exn-def*)  
**apply** (*rule exec-block-exnStuck*)  
**apply** (*erule CallUndefined*)  
**done**

**lemma** *Fault-end*: **assumes** *exec*:  $\Gamma \vdash \langle c, s \rangle \Rightarrow \ t$  **and** *s*:  $s = Fault \ f$   
**shows**  $t = Fault \ f$   
**using** *exec s* **by** (*induct*) *auto*

**lemma** *Stuck-end*: **assumes** *exec*:  $\Gamma \vdash \langle c, s \rangle \Rightarrow \ t$  **and** *s*:  $s = Stuck$   
**shows**  $t = Stuck$   
**using** *exec s* **by** (*induct*) *auto*

**lemma** *Abrupt-end*: **assumes** *exec*:  $\Gamma \vdash \langle c, s \rangle \Rightarrow \ t$  **and** *s*:  $s = Abrupt \ s'$   
**shows**  $t = Abrupt \ s'$

**using** *exec s by (induct) auto*

**lemma** *exec-Call-body-aux*:

$\Gamma \text{ } p = \text{Some } bdy \implies$   
 $\Gamma \vdash \langle \text{Call } p, s \rangle \Rightarrow t = \Gamma \vdash \langle bdy, s \rangle \Rightarrow t$   
**apply** (*rule*)  
**apply** (*fastforce elim: exec-elim-cases*)  
**apply** (*cases s*)  
**apply** (*cases t*)  
**apply** (*auto intro: exec.intros dest: Fault-end Stuck-end Abrupt-end*)  
**done**

**lemma** *exec-Call-body'*:

$p \in \text{dom } \Gamma \implies$   
 $\Gamma \vdash \langle \text{Call } p, s \rangle \Rightarrow t = \Gamma \vdash \langle \text{the } (\Gamma \text{ } p), s \rangle \Rightarrow t$   
**apply** *clarsimp*  
**by** (*rule exec-Call-body-aux*)

**lemma** *exec-block-exn-Normal-elim* [*consumes 1*]:

**assumes** *exec-block*:  $\Gamma \vdash \langle \text{block-exn init bdy return result-exn } c, \text{Normal } s \rangle \Rightarrow t$

**assumes** *Normal*:

$\wedge t'$ .  
 $\llbracket \Gamma \vdash \langle bdy, \text{Normal } (\text{init } s) \rangle \Rightarrow \text{Normal } t';$   
 $\Gamma \vdash \langle c \text{ } s \text{ } t', \text{Normal } (\text{return } s \text{ } t') \rangle \Rightarrow t \rrbracket$   
 $\implies P$

**assumes** *Abrupt*:

$\wedge t'$ .  
 $\llbracket \Gamma \vdash \langle bdy, \text{Normal } (\text{init } s) \rangle \Rightarrow \text{Abrupt } t';$   
 $t = \text{Abrupt } (\text{result-exn } (\text{return } s \text{ } t') \text{ } t') \rrbracket$   
 $\implies P$

**assumes** *Fault*:

$\wedge f$ .  
 $\llbracket \Gamma \vdash \langle bdy, \text{Normal } (\text{init } s) \rangle \Rightarrow \text{Fault } f;$   
 $t = \text{Fault } f \rrbracket$   
 $\implies P$

**assumes** *Stuck*:

$\llbracket \Gamma \vdash \langle bdy, \text{Normal } (\text{init } s) \rangle \Rightarrow \text{Stuck};$   
 $t = \text{Stuck} \rrbracket$   
 $\implies P$

**assumes**

$\llbracket \Gamma \text{ } p = \text{None}; t = \text{Stuck} \rrbracket \implies P$

**shows** *P*

**using** *exec-block*

**apply** (*unfold block-exn-def*)

**apply** (*elim exec-Normal-elim-cases*)

**apply** *simp-all*

**apply** (*case-tac s'*)

**apply** *simp-all*



```

apply (elim exec-Normal-elim-cases)
apply simp
apply (drule Abrupt-end) apply simp
apply (erule exec-Normal-elim-cases)
apply simp
apply (rule Abrupt,assumption+)
apply (drule Fault-end) apply simp
apply (erule exec-Normal-elim-cases)
apply simp
apply (drule Stuck-end) apply simp
apply (erule exec-Normal-elim-cases)
apply simp
apply (case-tac s')
apply simp-all
apply (elim exec-Normal-elim-cases)
apply simp
apply (rule Normal, assumption+)
apply (drule Fault-end) apply simp
apply (rule Fault,assumption+)
apply (drule Stuck-end) apply simp
apply (rule Stuck,assumption+)
done

```

**lemma** *exec-block-Normal-elim* [consumes 1]:

**assumes** *exec-block*:  $\Gamma \vdash \langle \text{block init bdy return } c, \text{Normal } s \rangle \Rightarrow t$

**assumes** *Normal*:

$\wedge t'$ .  
 $\llbracket \Gamma \vdash \langle \text{bdy}, \text{Normal } (\text{init } s) \rangle \Rightarrow \text{Normal } t';$   
 $\Gamma \vdash \langle c \ s \ t', \text{Normal } (\text{return } s \ t') \rangle \Rightarrow t \rrbracket$   
 $\Rightarrow P$

**assumes** *Abrupt*:

$\wedge t'$ .  
 $\llbracket \Gamma \vdash \langle \text{bdy}, \text{Normal } (\text{init } s) \rangle \Rightarrow \text{Abrupt } t';$   
 $t = \text{Abrupt } (\text{return } s \ t') \rrbracket$   
 $\Rightarrow P$

**assumes** *Fault*:

$\wedge f$ .  
 $\llbracket \Gamma \vdash \langle \text{bdy}, \text{Normal } (\text{init } s) \rangle \Rightarrow \text{Fault } f;$   
 $t = \text{Fault } f \rrbracket$   
 $\Rightarrow P$

**assumes** *Stuck*:

$\llbracket \Gamma \vdash \langle \text{bdy}, \text{Normal } (\text{init } s) \rangle \Rightarrow \text{Stuck};$   
 $t = \text{Stuck} \rrbracket$   
 $\Rightarrow P$

**assumes**

*Undef*:  $\llbracket \Gamma \ p = \text{None}; t = \text{Stuck} \rrbracket \Rightarrow P$

**shows** *P*

**by** (rule *exec-block-exn-Normal-elim* [OF *exec-block* [simplified block-def]

*Normal Abrupt Fault Stuck Undef* ])

**lemma** *exec-call-exn-Normal-elim* [*consumes 1*]:

**assumes** *exec-call*:  $\Gamma \vdash \langle \text{call-exn } \text{init } p \text{ return result-exn } c, \text{Normal } s \rangle \Rightarrow t$

**assumes** *Normal*:

$\wedge \text{bdy } t'$ .

$\llbracket \Gamma p = \text{Some bdy}; \Gamma \vdash \langle \text{bdy}, \text{Normal } (\text{init } s) \rangle \Rightarrow \text{Normal } t' \rrbracket$

$\llbracket \Gamma \vdash \langle c \ s \ t', \text{Normal } (\text{return } s \ t') \rangle \Rightarrow t \rrbracket$

$\Longrightarrow P$

**assumes** *Abrupt*:

$\wedge \text{bdy } t'$ .

$\llbracket \Gamma p = \text{Some bdy}; \Gamma \vdash \langle \text{bdy}, \text{Normal } (\text{init } s) \rangle \Rightarrow \text{Abrupt } t' \rrbracket$

$\llbracket t = \text{Abrupt } (\text{result-exn } (\text{return } s \ t') \ t') \rrbracket$

$\Longrightarrow P$

**assumes** *Fault*:

$\wedge \text{bdy } f$ .

$\llbracket \Gamma p = \text{Some bdy}; \Gamma \vdash \langle \text{bdy}, \text{Normal } (\text{init } s) \rangle \Rightarrow \text{Fault } f \rrbracket$

$\llbracket t = \text{Fault } f \rrbracket$

$\Longrightarrow P$

**assumes** *Stuck*:

$\wedge \text{bdy}$ .

$\llbracket \Gamma p = \text{Some bdy}; \Gamma \vdash \langle \text{bdy}, \text{Normal } (\text{init } s) \rangle \Rightarrow \text{Stuck} \rrbracket$

$\llbracket t = \text{Stuck} \rrbracket$

$\Longrightarrow P$

**assumes** *Undef*:

$\llbracket \Gamma p = \text{None}; t = \text{Stuck} \rrbracket \Longrightarrow P$

**shows** *P*

**using** *exec-call*

**apply** (*unfold call-exn-def*)

**apply** (*cases*  $\Gamma p$ )

**apply** (*erule exec-block-exn-Normal-elim*)

**apply** (*elim exec-Normal-elim-cases*)

**apply** *simp*

**apply** *simp*

**apply** (*elim exec-Normal-elim-cases*)

**apply** *simp*

**apply** (*elim exec-Normal-elim-cases*)

**apply** *simp*

**apply** *simp*

**apply** (*elim exec-Normal-elim-cases*)

**apply** *simp*

**apply** (*rule Undef,assumption,assumption*)

**apply** (*rule Undef,assumption+*)

**apply** (*erule exec-block-exn-Normal-elim*)

**apply** (*elim exec-Normal-elim-cases*)

**apply** *simp*

**apply** (*rule Normal,assumption+*)

**apply** *simp*

**apply** (*elim exec-Normal-elim-cases*)  
**apply** *simp*  
**apply** (*rule Abrupt,assumption+*)  
**apply** *simp*  
**apply** (*elim exec-Normal-elim-cases*)  
**apply** *simp*  
**apply** (*rule Fault, assumption+*)  
**apply** *simp*  
**apply** (*elim exec-Normal-elim-cases*)  
**apply** *simp*  
**apply** (*rule Stuck,assumption,assumption,assumption*)  
**apply** *simp*  
**apply** (*rule Undef,assumption+*)  
**done**

**lemma** *exec-call-Normal-elim* [*consumes 1*]:  
**assumes** *exec-call*:  $\Gamma \vdash \langle \text{call init } p \text{ return } c, \text{Normal } s \rangle \Rightarrow t$   
**assumes** *Normal*:  
 $\wedge \text{bdy } t'.$   
 $\llbracket \Gamma \text{ } p = \text{Some bdy}; \Gamma \vdash \langle \text{bdy}, \text{Normal } (\text{init } s) \rangle \Rightarrow \text{Normal } t';$   
 $\Gamma \vdash \langle c \text{ } s \text{ } t', \text{Normal } (\text{return } s \text{ } t') \rangle \Rightarrow t \rrbracket$   
 $\Rightarrow P$   
**assumes** *Abrupt*:  
 $\wedge \text{bdy } t'.$   
 $\llbracket \Gamma \text{ } p = \text{Some bdy}; \Gamma \vdash \langle \text{bdy}, \text{Normal } (\text{init } s) \rangle \Rightarrow \text{Abrupt } t';$   
 $t = \text{Abrupt } (\text{return } s \text{ } t') \rrbracket$   
 $\Rightarrow P$   
**assumes** *Fault*:  
 $\wedge \text{bdy } f.$   
 $\llbracket \Gamma \text{ } p = \text{Some bdy}; \Gamma \vdash \langle \text{bdy}, \text{Normal } (\text{init } s) \rangle \Rightarrow \text{Fault } f;$   
 $t = \text{Fault } f \rrbracket$   
 $\Rightarrow P$   
**assumes** *Stuck*:  
 $\wedge \text{bdy}.$   
 $\llbracket \Gamma \text{ } p = \text{Some bdy}; \Gamma \vdash \langle \text{bdy}, \text{Normal } (\text{init } s) \rangle \Rightarrow \text{Stuck};$   
 $t = \text{Stuck} \rrbracket$   
 $\Rightarrow P$   
**assumes** *Undef*:  
 $\llbracket \Gamma \text{ } p = \text{None}; t = \text{Stuck} \rrbracket \Longrightarrow P$   
**shows** *P*  
**using** *exec-call* [*simplified call-call-exn*] *Normal Abrupt Fault Stuck Undef*  
**by** (*rule exec-call-exn-Normal-elim*)

**lemma** *exec-dynCall*:  
 $\llbracket \Gamma \vdash \langle \text{call init } (p \text{ } s) \text{ return } c, \text{Normal } s \rangle \Rightarrow t \rrbracket$   
 $\Longrightarrow$   
 $\Gamma \vdash \langle \text{dynCall init } p \text{ return } c, \text{Normal } s \rangle \Rightarrow t$   
**apply** (*simp add: dynCall-def*)

by (rule DynCom)

**lemma** *exec-dynCall-exn*:

$\llbracket \Gamma \vdash \langle \text{call-exn init } (p \ s) \ \text{return result-exn } c, \text{Normal } s \rangle \Rightarrow t \rrbracket$   
 $\Longrightarrow$

$\Gamma \vdash \langle \text{dynCall-exn } f \ \text{UNIV init } p \ \text{return result-exn } c, \text{Normal } s \rangle \Rightarrow t$

apply (simp add: dynCall-exn-def)

by (rule DynCom)

**lemma** *exec-dynCall-Normal-elim*:

assumes *exec*:  $\Gamma \vdash \langle \text{dynCall init } p \ \text{return } c, \text{Normal } s \rangle \Rightarrow t$

assumes *call*:  $\Gamma \vdash \langle \text{call init } (p \ s) \ \text{return } c, \text{Normal } s \rangle \Rightarrow t \Longrightarrow P$

shows *P*

using *exec*

apply (simp add: dynCall-def)

apply (erule *exec-Normal-elim-cases*)

apply (rule *call,assumption*)

done

**lemma** *exec-guards-Normal-elim-cases* [*consumes 1, case-names noFault someFault*]:

assumes *exec-guards*:  $\Gamma \vdash \langle \text{guards } gs \ c, \text{Normal } s \rangle \Rightarrow t$

assumes *noFault*:  $\forall f \ g. (f, g) \in \text{set } gs \longrightarrow s \in g \Longrightarrow \Gamma \vdash \langle c, \text{Normal } s \rangle \Rightarrow t \Longrightarrow P$

assumes *someFault*:  $\bigwedge f \ g. \text{find } (\lambda(f, g). s \notin g) \ gs = \text{Some } (f, g) \Longrightarrow t = \text{Fault } f \Longrightarrow P$

shows *P*

using *exec-guards noFault someFault*

**proof** (induct *gs*)

case *Nil*

then show ?*case* by simp

**next**

case (*Cons pg gs*)

obtain *f g* where *pg*: *pg* = (*f, g*)

by (cases *pg*) auto

show ?*thesis*

**proof** (cases *s*  $\in$  *g*)

case *True*

from *Cons.prem1* have *exec-gs*:  $\Gamma \vdash \langle \text{guards } gs \ c, \text{Normal } s \rangle \Rightarrow t$

by (simp add: *pg*) (meson *True pg exec-Normal-elim-cases*)

from *Cons.hyps* [*OF exec-gs*] *Cons.prem2,3*)

show ?*thesis*

by (simp add: *pg True*)

**next**

case *False*

from *Cons.prem1* have *t*: *t* = *Fault f*

by (simp add: *pg*) (meson *False pg exec-Normal-elim-cases*)

```

    from  $t$  Cons.prems(3)
    show ?thesis
    by (simp add: pg False)
  qed
qed

```

```

lemma exec-guards-noFault:
  assumes exec:  $\Gamma \vdash \langle c, \text{Normal } s \rangle \Rightarrow t$ 
  assumes noFault:  $\forall f g. (f, g) \in \text{set } gs \longrightarrow s \in g$ 
  shows  $\Gamma \vdash \langle \text{guards } gs \ c, \text{Normal } s \rangle \Rightarrow t$ 
  using exec noFault by (induct gs) (auto intro: exec.intros)

```

```

lemma exec-guards-Fault:
  assumes Fault:  $\text{find } (\lambda(f,g). s \notin g) \ gs = \text{Some } (f, g)$ 
  shows  $\Gamma \vdash \langle \text{guards } gs \ c, \text{Normal } s \rangle \Rightarrow \text{Fault } f$ 
  using Fault by (induct gs) (auto intro: exec.intros split: prod.splits if-split-asm)

```

```

lemma exec-guards-DynCom:
  assumes exec-c:  $\Gamma \vdash \langle \text{guards } gs \ (c \ s), \text{Normal } s \rangle \Rightarrow t$ 
  shows  $\Gamma \vdash \langle \text{guards } gs \ (\text{DynCom } c), \text{Normal } s \rangle \Rightarrow t$ 
  using exec-c apply (induct gs)
  apply (fastforce intro: exec.intros)
  apply simp
  by (metis exec.Guard exec.GuardFault exec-Normal-elim-cases)

```

```

lemma exec-guards-DynCom-Normal-elim:
  assumes exec:  $\Gamma \vdash \langle \text{guards } gs \ (\text{DynCom } c), \text{Normal } s \rangle \Rightarrow t$ 
  assumes call:  $\Gamma \vdash \langle \text{guards } gs \ (c \ s), \text{Normal } s \rangle \Rightarrow t \Longrightarrow P$ 
  shows  $P$ 
  using exec call proof (induct gs)
  case Nil
  then show ?case
    apply simp
    apply (erule exec-Normal-elim-cases)
    apply simp
  done
next
  case (Cons g gs)
  then show ?case
    apply (cases g)
    apply simp
    apply (erule exec-Normal-elim-cases)
    apply simp
    apply (meson Guard)
    apply simp
    apply (meson GuardFault)
  done
qed

```

**lemma** *exec-maybe-guard-DynCom*:  
**assumes** *exec-c*:  $\Gamma \vdash \langle \text{maybe-guard } f \ g \ (c \ s), \text{Normal } s \rangle \Rightarrow t$   
**shows**  $\Gamma \vdash \langle \text{maybe-guard } f \ g \ (\text{DynCom } c), \text{Normal } s \rangle \Rightarrow t$   
**using** *exec-c*  
**by** (*metis DynCom Guard GuardFault exec-Normal-elim-cases(5) maybe-guard-def*)

**lemma** *exec-maybe-guard-Normal-elim-cases* [*consumes 1, case-names noFault someFault*]:

**assumes** *exec-guards*:  $\Gamma \vdash \langle \text{maybe-guard } f \ g \ c, \text{Normal } s \rangle \Rightarrow t$   
**assumes** *noFault*:  $s \in g \Longrightarrow \Gamma \vdash \langle c, \text{Normal } s \rangle \Rightarrow t \Longrightarrow P$   
**assumes** *someFault*:  $s \notin g \Longrightarrow t = \text{Fault } f \Longrightarrow P$   
**shows** *P*  
**using** *exec-guards noFault someFault*  
**by** (*metis UNIV-I exec-Normal-elim-cases(5) maybe-guard-def*)

**lemma** *exec-maybe-guard-noFault*:  
**assumes** *exec*:  $\Gamma \vdash \langle c, \text{Normal } s \rangle \Rightarrow t$   
**assumes** *noFault*:  $s \in g$   
**shows**  $\Gamma \vdash \langle \text{maybe-guard } f \ g \ c, \text{Normal } s \rangle \Rightarrow t$   
**using** *exec noFault*  
**by** (*simp add: Guard maybe-guard-def*)

**lemma** *exec-maybe-guard-Fault*:  
**assumes** *Fault*:  $s \notin g$   
**shows**  $\Gamma \vdash \langle \text{maybe-guard } f \ g \ c, \text{Normal } s \rangle \Rightarrow \text{Fault } f$   
**using** *Fault*  
**by** (*metis GuardFault iso-tuple-UNIV-I maybe-guard-def*)

**lemma** *exec-maybe-guard-DynCom-Normal-elim*:  
**assumes** *exec*:  $\Gamma \vdash \langle \text{maybe-guard } f \ g \ (\text{DynCom } c), \text{Normal } s \rangle \Rightarrow t$   
**assumes** *call*:  $\Gamma \vdash \langle \text{maybe-guard } f \ g \ (c \ s), \text{Normal } s \rangle \Rightarrow t \Longrightarrow P$   
**shows** *P*  
**using** *exec call*  
**apply** (*cases g=UNIV*)  
**subgoal**  
**apply** *simp*  
**apply** (*erule exec-Normal-elim-cases*)  
**apply** *simp*  
**done**  
**subgoal**  
**apply** (*simp add: maybe-guard-def*)  
**apply** (*erule exec-Normal-elim-cases*)  
**apply** (*meson Guard exec-Normal-elim-cases(12)*)  
**by** (*meson GuardFault*)  
**done**

**lemma** *exec-dynCall-exn-Normal-elim*:  
**assumes** *exec*:  $\Gamma \vdash \langle \text{dynCall-exn } f \ g \ \text{init } p \ \text{return result-exn } c, \text{Normal } s \rangle \Rightarrow t$

**assumes** *call*:  $\Gamma \vdash \langle \text{maybe-guard } f \ g \ (\text{call-exn init } (p \ s) \ \text{return result-exn } c), \text{Normal } s \rangle \Rightarrow t \Longrightarrow P$   
**shows**  $P$   
**using** *exec*  
**apply** (*simp add: dynCall-exn-def*)  
**apply** (*erule exec-maybe-guard-DynCom-Normal-elim*)  
**by** (*rule call*)

**lemma** *exec-Call-body*:

$\Gamma \ p = \text{Some } \text{bdy} \Longrightarrow$   
 $\Gamma \vdash \langle \text{Call } p, s \rangle \Rightarrow t = \Gamma \vdash \langle \text{the } (\Gamma \ p), s \rangle \Rightarrow t$   
**apply** (*rule*)  
**apply** (*fastforce elim: exec-elim-cases*)  
**apply** (*cases s*)  
**apply** (*cases t*)  
**apply** (*fastforce intro: exec.intros dest: Fault-end Abrupt-end Stuck-end*)  
**done**

**lemma** *exec-Seq'*:  $\llbracket \Gamma \vdash \langle c1, s \rangle \Rightarrow s'; \Gamma \vdash \langle c2, s' \rangle \Rightarrow s'' \rrbracket$

$\Longrightarrow$   
 $\Gamma \vdash \langle \text{Seq } c1 \ c2, s \rangle \Rightarrow s''$   
**apply** (*cases s*)  
**apply** (*fastforce intro: exec.intros*)  
**apply** (*fastforce dest: Abrupt-end*)  
**apply** (*fastforce dest: Fault-end*)  
**apply** (*fastforce dest: Stuck-end*)  
**done**

**lemma** *exec-assoc*:  $\Gamma \vdash \langle \text{Seq } c1 \ (\text{Seq } c2 \ c3), s \rangle \Rightarrow t = \Gamma \vdash \langle \text{Seq } (\text{Seq } c1 \ c2) \ c3, s \rangle \Rightarrow t$

**by** (*blast elim!: exec-elim-cases intro: exec-Seq'*)

**3.2 Big-Step Execution with Recursion Limit:**  $\Gamma \vdash \langle c, s \rangle = n \Rightarrow t$

**inductive** *execn*:: $[(\ 's, 'p, 'f) \ \text{body}, (\ 's, 'p, 'f) \ \text{com}, (\ 's, 'f) \ \text{xstate}, \text{nat}, (\ 's, 'f) \ \text{xstate}]$   
 $\Rightarrow \text{bool } (\ \vdash \ \langle -, - \rangle = n \Rightarrow \rightarrow \ [60, 20, 98, 65, 98] \ 89)$

**for**  $\Gamma :: (\ 's, 'p, 'f) \ \text{body}$

**where**

$\text{Skip}: \Gamma \vdash \langle \text{Skip}, \text{Normal } s \rangle = n \Rightarrow \text{Normal } s$   
 $\text{Guard}: \llbracket s \in g; \Gamma \vdash \langle c, \text{Normal } s \rangle = n \Rightarrow t \rrbracket$   
 $\Longrightarrow$   
 $\Gamma \vdash \langle \text{Guard } f \ g \ c, \text{Normal } s \rangle = n \Rightarrow t$

$\text{GuardFault}: s \notin g \Longrightarrow \Gamma \vdash \langle \text{Guard } f \ g \ c, \text{Normal } s \rangle = n \Rightarrow \text{Fault } f$

$\text{FaultProp } [\text{intro}, \text{simp}]: \Gamma \vdash \langle c, \text{Fault } f \rangle = n \Rightarrow \text{Fault } f$

- | *Basic*:  $\Gamma \vdash \langle \text{Basic } f, \text{Normal } s \rangle = n \Rightarrow \text{Normal } (f s)$
- | *Spec*:  $(s, t) \in r$   
 $\implies$   
 $\Gamma \vdash \langle \text{Spec } r, \text{Normal } s \rangle = n \Rightarrow \text{Normal } t$
- | *SpecStuck*:  $\forall t. (s, t) \notin r$   
 $\implies$   
 $\Gamma \vdash \langle \text{Spec } r, \text{Normal } s \rangle = n \Rightarrow \text{Stuck}$
- | *Seq*:  $\llbracket \Gamma \vdash \langle c_1, \text{Normal } s \rangle = n \Rightarrow s'; \Gamma \vdash \langle c_2, s' \rangle = n \Rightarrow t \rrbracket$   
 $\implies$   
 $\Gamma \vdash \langle \text{Seq } c_1 c_2, \text{Normal } s \rangle = n \Rightarrow t$
- | *CondTrue*:  $\llbracket s \in b; \Gamma \vdash \langle c_1, \text{Normal } s \rangle = n \Rightarrow t \rrbracket$   
 $\implies$   
 $\Gamma \vdash \langle \text{Cond } b c_1 c_2, \text{Normal } s \rangle = n \Rightarrow t$
- | *CondFalse*:  $\llbracket s \notin b; \Gamma \vdash \langle c_2, \text{Normal } s \rangle = n \Rightarrow t \rrbracket$   
 $\implies$   
 $\Gamma \vdash \langle \text{Cond } b c_1 c_2, \text{Normal } s \rangle = n \Rightarrow t$
- | *WhileTrue*:  $\llbracket s \in b; \Gamma \vdash \langle c, \text{Normal } s \rangle = n \Rightarrow s'; \Gamma \vdash \langle \text{While } b c, s' \rangle = n \Rightarrow t \rrbracket$   
 $\implies$   
 $\Gamma \vdash \langle \text{While } b c, \text{Normal } s \rangle = n \Rightarrow t$
- | *WhileFalse*:  $\llbracket s \notin b \rrbracket$   
 $\implies$   
 $\Gamma \vdash \langle \text{While } b c, \text{Normal } s \rangle = n \Rightarrow \text{Normal } s$
- | *Call*:  $\llbracket \Gamma \vdash p = \text{Some } bdy; \Gamma \vdash \langle bdy, \text{Normal } s \rangle = n \Rightarrow t \rrbracket$   
 $\implies$   
 $\Gamma \vdash \langle \text{Call } p, \text{Normal } s \rangle = \text{Suc } n \Rightarrow t$
- | *CallUndefined*:  $\llbracket \Gamma \vdash p = \text{None} \rrbracket$   
 $\implies$   
 $\Gamma \vdash \langle \text{Call } p, \text{Normal } s \rangle = \text{Suc } n \Rightarrow \text{Stuck}$
- | *StuckProp* [*intro, simp*]:  $\Gamma \vdash \langle c, \text{Stuck} \rangle = n \Rightarrow \text{Stuck}$
- | *DynCom*:  $\llbracket \Gamma \vdash \langle (c s), \text{Normal } s \rangle = n \Rightarrow t \rrbracket$   
 $\implies$   
 $\Gamma \vdash \langle \text{DynCom } c, \text{Normal } s \rangle = n \Rightarrow t$
- | *Throw*:  $\Gamma \vdash \langle \text{Throw}, \text{Normal } s \rangle = n \Rightarrow \text{Abrupt } s$
- | *AbruptProp* [*intro, simp*]:  $\Gamma \vdash \langle c, \text{Abrupt } s \rangle = n \Rightarrow \text{Abrupt } s$



| *CatchMatch*:  $\llbracket \Gamma \vdash \langle c_1, \text{Normal } s \rangle = n \Rightarrow \text{Abrupt } s'; \Gamma \vdash \langle c_2, \text{Normal } s' \rangle = n \Rightarrow t \rrbracket$   
 $\implies$   
 $\Gamma \vdash \langle \text{Catch } c_1 \ c_2, \text{Normal } s \rangle = n \Rightarrow t$   
| *CatchMiss*:  $\llbracket \Gamma \vdash \langle c_1, \text{Normal } s \rangle = n \Rightarrow t; \neg \text{isAbr } t \rrbracket$   
 $\implies$   
 $\Gamma \vdash \langle \text{Catch } c_1 \ c_2, \text{Normal } s \rangle = n \Rightarrow t$

**inductive-cases** *execn-elim-cases* [*cases set*]:

$\Gamma \vdash \langle c, \text{Fault } f \rangle = n \Rightarrow t$   
 $\Gamma \vdash \langle c, \text{Stuck} \rangle = n \Rightarrow t$   
 $\Gamma \vdash \langle c, \text{Abrupt } s \rangle = n \Rightarrow t$   
 $\Gamma \vdash \langle \text{Skip}, s \rangle = n \Rightarrow t$   
 $\Gamma \vdash \langle \text{Seq } c_1 \ c_2, s \rangle = n \Rightarrow t$   
 $\Gamma \vdash \langle \text{Guard } f \ g \ c, s \rangle = n \Rightarrow t$   
 $\Gamma \vdash \langle \text{Basic } f, s \rangle = n \Rightarrow t$   
 $\Gamma \vdash \langle \text{Spec } r, s \rangle = n \Rightarrow t$   
 $\Gamma \vdash \langle \text{Cond } b \ c_1 \ c_2, s \rangle = n \Rightarrow t$   
 $\Gamma \vdash \langle \text{While } b \ c, s \rangle = n \Rightarrow t$   
 $\Gamma \vdash \langle \text{Call } p, s \rangle = n \Rightarrow t$   
 $\Gamma \vdash \langle \text{DynCom } c, s \rangle = n \Rightarrow t$   
 $\Gamma \vdash \langle \text{Throw}, s \rangle = n \Rightarrow t$   
 $\Gamma \vdash \langle \text{Catch } c_1 \ c_2, s \rangle = n \Rightarrow t$

**inductive-cases** *execn-Normal-elim-cases* [*cases set*]:

$\Gamma \vdash \langle c, \text{Fault } f \rangle = n \Rightarrow t$   
 $\Gamma \vdash \langle c, \text{Stuck} \rangle = n \Rightarrow t$   
 $\Gamma \vdash \langle c, \text{Abrupt } s \rangle = n \Rightarrow t$   
 $\Gamma \vdash \langle \text{Skip}, \text{Normal } s \rangle = n \Rightarrow t$   
 $\Gamma \vdash \langle \text{Guard } f \ g \ c, \text{Normal } s \rangle = n \Rightarrow t$   
 $\Gamma \vdash \langle \text{Basic } f, \text{Normal } s \rangle = n \Rightarrow t$   
 $\Gamma \vdash \langle \text{Spec } r, \text{Normal } s \rangle = n \Rightarrow t$   
 $\Gamma \vdash \langle \text{Seq } c_1 \ c_2, \text{Normal } s \rangle = n \Rightarrow t$   
 $\Gamma \vdash \langle \text{Cond } b \ c_1 \ c_2, \text{Normal } s \rangle = n \Rightarrow t$   
 $\Gamma \vdash \langle \text{While } b \ c, \text{Normal } s \rangle = n \Rightarrow t$   
 $\Gamma \vdash \langle \text{Call } p, \text{Normal } s \rangle = n \Rightarrow t$   
 $\Gamma \vdash \langle \text{DynCom } c, \text{Normal } s \rangle = n \Rightarrow t$   
 $\Gamma \vdash \langle \text{Throw}, \text{Normal } s \rangle = n \Rightarrow t$   
 $\Gamma \vdash \langle \text{Catch } c_1 \ c_2, \text{Normal } s \rangle = n \Rightarrow t$

**lemma** *execn-Skip'*:  $\Gamma \vdash \langle \text{Skip}, t \rangle = n \Rightarrow t$   
**by** (*cases t*) (*auto intro: execn.intros*)

**lemma** *execn-Fault-end*: **assumes** *exec*:  $\Gamma \vdash \langle c, s \rangle = n \Rightarrow t$  **and** *s*: *s = Fault f*  
**shows** *t = Fault f*  
**using** *exec s* **by** (*induct*) *auto*

**lemma** *execn-Stuck-end*: **assumes** *exec*:  $\Gamma \vdash \langle c, s \rangle = n \Rightarrow t$  **and** *s*: *s = Stuck*

**shows**  $t=Stuck$   
**using**  $exec\ s$  **by**  $(induct)\ auto$

**lemma**  $execn\text{-}Abrupt\text{-}end$ : **assumes**  $exec: \Gamma \vdash \langle c, s \rangle =n \Rightarrow t$  **and**  $s: s=Abrupt\ s'$   
**shows**  $t=Abrupt\ s'$   
**using**  $exec\ s$  **by**  $(induct)\ auto$

**lemma**  $execn\text{-}block\text{-}exn$ :  
 $\llbracket \Gamma \vdash \langle bdy, Normal\ (init\ s) \rangle =n \Rightarrow Normal\ t; \Gamma \vdash \langle c\ s\ t, Normal\ (return\ s\ t) \rangle =n \Rightarrow u \rrbracket$   
 $\Rightarrow$   
 $\Gamma \vdash \langle block\text{-}exn\ init\ bdy\ return\ result\text{-}exn\ c, Normal\ s \rangle =n \Rightarrow u$   
**apply**  $(unfold\ block\text{-}exn\text{-}def)$   
**by**  $(fastforce\ intro: execn.intros)$

**lemma**  $execn\text{-}block$ :  
 $\llbracket \Gamma \vdash \langle bdy, Normal\ (init\ s) \rangle =n \Rightarrow Normal\ t; \Gamma \vdash \langle c\ s\ t, Normal\ (return\ s\ t) \rangle =n \Rightarrow u \rrbracket$   
 $\Rightarrow$   
 $\Gamma \vdash \langle block\ init\ bdy\ return\ c, Normal\ s \rangle =n \Rightarrow u$   
**unfolding**  $block\text{-}def$   
**by**  $(rule\ execn\text{-}block\text{-}exn)$

**lemma**  $execn\text{-}block\text{-}exnAbrupt$ :  
 $\llbracket \Gamma \vdash \langle bdy, Normal\ (init\ s) \rangle =n \Rightarrow Abrupt\ t \rrbracket$   
 $\Rightarrow$   
 $\Gamma \vdash \langle block\text{-}exn\ init\ bdy\ return\ result\text{-}exn\ c, Normal\ s \rangle =n \Rightarrow Abrupt\ (result\text{-}exn\ (return\ s\ t)\ t)$   
**apply**  $(unfold\ block\text{-}exn\text{-}def)$   
**by**  $(fastforce\ intro: execn.intros)$

**lemma**  $execn\text{-}blockAbrupt$ :  
 $\llbracket \Gamma \vdash \langle bdy, Normal\ (init\ s) \rangle =n \Rightarrow Abrupt\ t \rrbracket$   
 $\Rightarrow$   
 $\Gamma \vdash \langle block\ init\ bdy\ return\ c, Normal\ s \rangle =n \Rightarrow Abrupt\ (return\ s\ t)$   
**unfolding**  $block\text{-}def$   
**by**  $(rule\ execn\text{-}block\text{-}exnAbrupt)$

**lemma**  $execn\text{-}block\text{-}exnFault$ :  
 $\llbracket \Gamma \vdash \langle bdy, Normal\ (init\ s) \rangle =n \Rightarrow Fault\ f \rrbracket$   
 $\Rightarrow$   
 $\Gamma \vdash \langle block\text{-}exn\ init\ bdy\ return\ result\text{-}exn\ c, Normal\ s \rangle =n \Rightarrow Fault\ f$   
**apply**  $(unfold\ block\text{-}exn\text{-}def)$   
**by**  $(fastforce\ intro: execn.intros)$

**lemma**  $execn\text{-}blockFault$ :  
 $\llbracket \Gamma \vdash \langle bdy, Normal\ (init\ s) \rangle =n \Rightarrow Fault\ f \rrbracket$   
 $\Rightarrow$   
 $\Gamma \vdash \langle block\ init\ bdy\ return\ c, Normal\ s \rangle =n \Rightarrow Fault\ f$

**unfolding** *block-def*  
**by** (*rule execn-block-exnFault*)

**lemma** *execn-block-exnStuck*:  
 $\llbracket \Gamma \vdash \langle bdy, Normal (init\ s) \rangle = n \Rightarrow Stuck \rrbracket$   
 $\Rightarrow$   
 $\Gamma \vdash \langle block\text{-}exn\ init\ bdy\ return\ result\text{-}exn\ c, Normal\ s \rangle = n \Rightarrow Stuck$   
**apply** (*unfold block-exn-def*)  
**by** (*fastforce intro: execn.intros*)

**lemma** *execn-blockStuck*:  
 $\llbracket \Gamma \vdash \langle bdy, Normal (init\ s) \rangle = n \Rightarrow Stuck \rrbracket$   
 $\Rightarrow$   
 $\Gamma \vdash \langle block\ init\ bdy\ return\ c, Normal\ s \rangle = n \Rightarrow Stuck$   
**unfolding** *block-def*  
**by** (*rule execn-block-exnStuck*)

**lemma** *execn-call*:  
 $\llbracket \Gamma\ p = Some\ bdy; \Gamma \vdash \langle bdy, Normal (init\ s) \rangle = n \Rightarrow Normal\ t; \Gamma \vdash \langle c\ s\ t, Normal (return\ s\ t) \rangle = Suc\ n \Rightarrow u \rrbracket$   
 $\Rightarrow$   
 $\Gamma \vdash \langle call\ init\ p\ return\ c, Normal\ s \rangle = Suc\ n \Rightarrow u$   
**apply** (*simp add: call-def*)  
**apply** (*rule execn-block*)  
**apply** (*erule (1) Call*)  
**apply** *assumption*  
**done**

**lemma** *execn-call-exn*:  
 $\llbracket \Gamma\ p = Some\ bdy; \Gamma \vdash \langle bdy, Normal (init\ s) \rangle = n \Rightarrow Normal\ t; \Gamma \vdash \langle c\ s\ t, Normal (return\ s\ t) \rangle = Suc\ n \Rightarrow u \rrbracket$   
 $\Rightarrow$   
 $\Gamma \vdash \langle call\text{-}exn\ init\ p\ return\ result\text{-}exn\ c, Normal\ s \rangle = Suc\ n \Rightarrow u$   
**apply** (*simp add: call-exn-def*)  
**apply** (*rule execn-block-exn*)  
**apply** (*erule (1) Call*)  
**apply** *assumption*  
**done**

**lemma** *execn-callAbrupt*:  
 $\llbracket \Gamma\ p = Some\ bdy; \Gamma \vdash \langle bdy, Normal (init\ s) \rangle = n \Rightarrow Abrupt\ t \rrbracket$   
 $\Rightarrow$   
 $\Gamma \vdash \langle call\ init\ p\ return\ c, Normal\ s \rangle = Suc\ n \Rightarrow Abrupt (return\ s\ t)$   
**apply** (*simp add: call-def*)  
**apply** (*rule execn-blockAbrupt*)  
**apply** (*erule (1) Call*)  
**done**

**lemma** *execn-call-exnAbrupt*:

$\llbracket \Gamma \ p=Some \ bdy; \Gamma \vdash \langle bdy, Normal \ (init \ s) \rangle =n \Rightarrow \ Abrupt \ t \rrbracket$   
 $\implies$   
 $\Gamma \vdash \langle call\text{-}exn \ init \ p \ return \ result\text{-}exn \ c, Normal \ s \rangle =Suc \ n \Rightarrow \ Abrupt \ (result\text{-}exn \ (return \ s \ t) \ t)$   
**apply** (*simp add: call-exn-def*)  
**apply** (*rule execn-block-exnAbrupt*)  
**apply** (*erule (1) Call*)  
**done**

**lemma** *execn-callFault*:

$\llbracket \Gamma \ p=Some \ bdy; \Gamma \vdash \langle bdy, Normal \ (init \ s) \rangle =n \Rightarrow \ Fault \ f \rrbracket$   
 $\implies$   
 $\Gamma \vdash \langle call \ init \ p \ return \ c, Normal \ s \rangle =Suc \ n \Rightarrow \ Fault \ f$   
**apply** (*simp add: call-def*)  
**apply** (*rule execn-blockFault*)  
**apply** (*erule (1) Call*)  
**done**

**lemma** *execn-call-exnFault*:

$\llbracket \Gamma \ p=Some \ bdy; \Gamma \vdash \langle bdy, Normal \ (init \ s) \rangle =n \Rightarrow \ Fault \ f \rrbracket$   
 $\implies$   
 $\Gamma \vdash \langle call\text{-}exn \ init \ p \ return \ result\text{-}exn \ c, Normal \ s \rangle =Suc \ n \Rightarrow \ Fault \ f$   
**apply** (*simp add: call-exn-def*)  
**apply** (*rule execn-block-exnFault*)  
**apply** (*erule (1) Call*)  
**done**

**lemma** *execn-callStuck*:

$\llbracket \Gamma \ p=Some \ bdy; \Gamma \vdash \langle bdy, Normal \ (init \ s) \rangle =n \Rightarrow \ Stuck \rrbracket$   
 $\implies$   
 $\Gamma \vdash \langle call \ init \ p \ return \ c, Normal \ s \rangle =Suc \ n \Rightarrow \ Stuck$   
**apply** (*simp add: call-def*)  
**apply** (*rule execn-blockStuck*)  
**apply** (*erule (1) Call*)  
**done**

**lemma** *execn-call-exnStuck*:

$\llbracket \Gamma \ p=Some \ bdy; \Gamma \vdash \langle bdy, Normal \ (init \ s) \rangle =n \Rightarrow \ Stuck \rrbracket$   
 $\implies$   
 $\Gamma \vdash \langle call\text{-}exn \ init \ p \ return \ result\text{-}exn \ c, Normal \ s \rangle =Suc \ n \Rightarrow \ Stuck$   
**apply** (*simp add: call-exn-def*)  
**apply** (*rule execn-block-exnStuck*)  
**apply** (*erule (1) Call*)  
**done**

**lemma** *execn-callUndefined*:

$\llbracket \Gamma \ p=None \rrbracket$

```

    =>
    Γ⊢⟨call init p return c,Normal s⟩ =Suc n⇒ Stuck
  apply (simp add: call-def)
  apply (rule execn-blockStuck)
  apply (erule CallUndefined)
  done

```

```

lemma execn-call-exnUndefined:
  [[Γ p=None]]
  =>
  Γ⊢⟨call-exn init p return result-exn c,Normal s⟩ =Suc n⇒ Stuck
  apply (simp add: call-exn-def)
  apply (rule execn-block-exnStuck)
  apply (erule CallUndefined)
  done

```

```

lemma execn-block-exn-Normal-elim [consumes 1]:
assumes execn-block: Γ⊢⟨block-exn init bdy return result-exn c,Normal s⟩ =n⇒ t
assumes Normal:
  ∧t'.
  [[Γ⊢⟨bdy,Normal (init s)⟩ =n⇒ Normal t';
   Γ⊢⟨c s t',Normal (return s t')⟩ =n⇒ t]]
  => P
assumes Abrupt:
  ∧t'.
  [[Γ⊢⟨bdy,Normal (init s)⟩ =n⇒ Abrupt t';
   t = Abrupt (result-exn (return s t') t)]]
  => P
assumes Fault:
  ∧f.
  [[Γ⊢⟨bdy,Normal (init s)⟩ =n⇒ Fault f;
   t = Fault f]]
  => P
assumes Stuck:
  [[Γ⊢⟨bdy,Normal (init s)⟩ =n⇒ Stuck;
   t = Stuck]]
  => P
assumes Undef:
  [[Γ p = None; t = Stuck]] ==> P
shows P
  using execn-block
  apply (unfold block-exn-def)
  apply (elim execn-Normal-elim-cases)
  apply simp-all
  apply (case-tac s')
  apply simp-all
  apply (elim execn-Normal-elim-cases)
  apply simp
  apply (drule execn-Abrupt-end) apply simp

```

```

apply (erule execn-Normal-elim-cases)
apply simp
apply (rule Abrupt,assumption+)
apply (drule execn-Fault-end) apply simp
apply (erule execn-Normal-elim-cases)
apply simp
apply (drule execn-Stuck-end) apply simp
apply (erule execn-Normal-elim-cases)
apply simp
apply (case-tac s')
apply simp-all
apply (elim execn-Normal-elim-cases)
apply simp
apply (rule Normal,assumption+)
apply (drule execn-Fault-end) apply simp
apply (rule Fault,assumption+)
apply (drule execn-Stuck-end) apply simp
apply (rule Stuck,assumption+)
done

```

**lemma** *execn-block-Normal-elim* [consumes 1]:

**assumes** *execn-block*:  $\Gamma \vdash \langle \text{block init bdy return } c, \text{Normal } s \rangle =n\Rightarrow t$

**assumes** *Normal*:

$\wedge t'$ .  
 $\llbracket \Gamma \vdash \langle \text{bdy}, \text{Normal } (\text{init } s) \rangle =n\Rightarrow \text{Normal } t' ;$   
 $\Gamma \vdash \langle c \ s \ t', \text{Normal } (\text{return } s \ t') \rangle =n\Rightarrow t \rrbracket$   
 $\implies P$

**assumes** *Abrupt*:

$\wedge t'$ .  
 $\llbracket \Gamma \vdash \langle \text{bdy}, \text{Normal } (\text{init } s) \rangle =n\Rightarrow \text{Abrupt } t' ;$   
 $t = \text{Abrupt } (\text{return } s \ t') \rrbracket$   
 $\implies P$

**assumes** *Fault*:

$\wedge f$ .  
 $\llbracket \Gamma \vdash \langle \text{bdy}, \text{Normal } (\text{init } s) \rangle =n\Rightarrow \text{Fault } f ;$   
 $t = \text{Fault } f \rrbracket$   
 $\implies P$

**assumes** *Stuck*:

$\llbracket \Gamma \vdash \langle \text{bdy}, \text{Normal } (\text{init } s) \rangle =n\Rightarrow \text{Stuck} ;$   
 $t = \text{Stuck} \rrbracket$   
 $\implies P$

**assumes** *Undef*:

$\llbracket \Gamma \ p = \text{None} ; t = \text{Stuck} \rrbracket \implies P$

**shows** *P*

**using** *execn-block* [unfolded block-def] *Normal Abrupt Fault Stuck Undef*  
**by** (rule *execn-block-exn-Normal-elim*)

**lemma** *execn-call-exn-Normal-elim* [consumes 1]:

**assumes** *exec-call*:  $\Gamma \vdash \langle \text{call-exn init } p \text{ return result-exn } c, \text{Normal } s \rangle =n\Rightarrow t$

**assumes** *Normal*:

$\wedge bdy\ i\ t'$ .

$\llbracket \Gamma\ p = \text{Some } bdy; \Gamma \vdash \langle bdy, \text{Normal } (init\ s) \rangle = i \Rightarrow \text{Normal } t';$   
 $\Gamma \vdash \langle c\ s\ t', \text{Normal } (return\ s\ t') \rangle = \text{Suc } i \Rightarrow t; n = \text{Suc } i \rrbracket$   
 $\Rightarrow P$

**assumes** *Abrupt*:

$\wedge bdy\ i\ t'$ .

$\llbracket \Gamma\ p = \text{Some } bdy; \Gamma \vdash \langle bdy, \text{Normal } (init\ s) \rangle = i \Rightarrow \text{Abrupt } t'; n = \text{Suc } i;$   
 $t = \text{Abrupt } (result\ exn\ (return\ s\ t')\ t') \rrbracket$   
 $\Rightarrow P$

**assumes** *Fault*:

$\wedge bdy\ i\ f$ .

$\llbracket \Gamma\ p = \text{Some } bdy; \Gamma \vdash \langle bdy, \text{Normal } (init\ s) \rangle = i \Rightarrow \text{Fault } f; n = \text{Suc } i;$   
 $t = \text{Fault } f \rrbracket$   
 $\Rightarrow P$

**assumes** *Stuck*:

$\wedge bdy\ i$ .

$\llbracket \Gamma\ p = \text{Some } bdy; \Gamma \vdash \langle bdy, \text{Normal } (init\ s) \rangle = i \Rightarrow \text{Stuck}; n = \text{Suc } i;$   
 $t = \text{Stuck} \rrbracket$   
 $\Rightarrow P$

**assumes** *Undef*:

$\wedge i. \llbracket \Gamma\ p = \text{None}; n = \text{Suc } i; t = \text{Stuck} \rrbracket \Rightarrow P$

**shows** *P*

**using** *exec-call*

**apply** (*unfold call-exn-def*)

**apply** (*cases n*)

**apply** (*simp only: block-exn-def*)

**apply** (*fastforce elim: execn-Normal-elim-cases*)

**apply** (*cases  $\Gamma\ p$* )

**apply** (*erule execn-block-exn-Normal-elim*)

**apply** (*elim execn-Normal-elim-cases*)

**apply** *simp*

**apply** *simp*

**apply** (*elim execn-Normal-elim-cases*)

**apply** *simp*

**apply** *simp*

**apply** (*elim execn-Normal-elim-cases*)

**apply** *simp*

**apply** *simp*

**apply** (*elim execn-Normal-elim-cases*)

**apply** *simp*

**apply** (*rule Undef,assumption,assumption,assumption*)

**apply** (*rule Undef,assumption+*)

**apply** (*erule execn-block-exn-Normal-elim*)

**apply** (*elim execn-Normal-elim-cases*)

**apply** *simp*

**apply** (*rule Normal,assumption+*)

**apply** *simp*

**apply** (*elim execn-Normal-elim-cases*)

**apply** *simp*  
**apply** (rule *Abrupt,assumption+*)  
**apply** *simp*  
**apply** (elim *execn-Normal-elim-cases*)  
**apply** *simp*  
**apply** (rule *Fault,assumption+*)  
**apply** *simp*  
**apply** (elim *execn-Normal-elim-cases*)  
**apply** *simp*  
**apply** (rule *Stuck,assumption,assumption,assumption,assumption*)  
**apply** (rule *Undef,assumption,assumption,assumption*)  
**apply** (rule *Undef,assumption+*)  
**done**

**lemma** *execn-call-Normal-elim* [*consumes 1*]:

**assumes** *exec-call*:  $\Gamma \vdash \langle \text{call } \text{init } p \text{ return } c, \text{Normal } s \rangle = n \Rightarrow t$

**assumes** *Normal*:

$\wedge \text{bdy } i \text{ } t'$ .

$\llbracket \Gamma \text{ } p = \text{Some } \text{bdy}; \Gamma \vdash \langle \text{bdy}, \text{Normal } (\text{init } s) \rangle = i \Rightarrow \text{Normal } t';$

$\Gamma \vdash \langle c \text{ } s \text{ } t', \text{Normal } (\text{return } s \text{ } t') \rangle = \text{Suc } i \Rightarrow t; n = \text{Suc } i \rrbracket$

$\Rightarrow P$

**assumes** *Abrupt*:

$\wedge \text{bdy } i \text{ } t'$ .

$\llbracket \Gamma \text{ } p = \text{Some } \text{bdy}; \Gamma \vdash \langle \text{bdy}, \text{Normal } (\text{init } s) \rangle = i \Rightarrow \text{Abrupt } t'; n = \text{Suc } i;$

$t = \text{Abrupt } (\text{return } s \text{ } t') \rrbracket$

$\Rightarrow P$

**assumes** *Fault*:

$\wedge \text{bdy } i \text{ } f$ .

$\llbracket \Gamma \text{ } p = \text{Some } \text{bdy}; \Gamma \vdash \langle \text{bdy}, \text{Normal } (\text{init } s) \rangle = i \Rightarrow \text{Fault } f; n = \text{Suc } i;$

$t = \text{Fault } f \rrbracket$

$\Rightarrow P$

**assumes** *Stuck*:

$\wedge \text{bdy } i$ .

$\llbracket \Gamma \text{ } p = \text{Some } \text{bdy}; \Gamma \vdash \langle \text{bdy}, \text{Normal } (\text{init } s) \rangle = i \Rightarrow \text{Stuck}; n = \text{Suc } i;$

$t = \text{Stuck} \rrbracket$

$\Rightarrow P$

**assumes** *Undef*:

$\wedge i. \llbracket \Gamma \text{ } p = \text{None}; n = \text{Suc } i; t = \text{Stuck} \rrbracket \Rightarrow P$

**shows** *P*

**using** *exec-call* [*simplified call-call-exn*] *Normal Abrupt Fault Stuck Undef*

**by** (rule *execn-call-exn-Normal-elim*)

**lemma** *execn-dynCall*:

$\llbracket \Gamma \vdash \langle \text{call } \text{init } (p \text{ } s) \text{ return } c, \text{Normal } s \rangle = n \Rightarrow t \rrbracket$

$\Rightarrow$

$\Gamma \vdash \langle \text{dynCall } \text{init } p \text{ return } c, \text{Normal } s \rangle = n \Rightarrow t$

**apply** (*simp add: dynCall-def*)



**by** (rule *DynCom*)

**lemma** *execn-dynCall-exn*:

$\llbracket \Gamma \vdash \langle \text{call-exn init } (p \ s) \ \text{return result-exn } c, \text{Normal } s \rangle = n \Rightarrow t \rrbracket$   
 $\Longrightarrow$   
 $\Gamma \vdash \langle \text{dynCall-exn } f \ \text{UNIV init } p \ \text{return result-exn } c, \text{Normal } s \rangle = n \Rightarrow t$   
**apply** (*simp add: dynCall-exn-def*)  
**by** (rule *DynCom*)

**lemma** *execn-dynCall-Normal-elim*:

**assumes** *exec*:  $\Gamma \vdash \langle \text{dynCall init } p \ \text{return } c, \text{Normal } s \rangle = n \Rightarrow t$   
**assumes**  $\Gamma \vdash \langle \text{call init } (p \ s) \ \text{return } c, \text{Normal } s \rangle = n \Rightarrow t \Longrightarrow P$   
**shows** *P*  
**using** *exec*  
**apply** (*simp add: dynCall-def*)  
**apply** (*erule execn-Normal-elim-cases*)  
**apply** *fact*  
**done**

**lemma** *execn-guards-Normal-elim-cases* [*consumes 1, case-names noFault someFault*]:

**assumes** *exec-guards*:  $\Gamma \vdash \langle \text{guards } gs \ c, \text{Normal } s \rangle = n \Rightarrow t$   
**assumes** *noFault*:  $\forall f \ g. (f, g) \in \text{set } gs \longrightarrow s \in g \Longrightarrow \Gamma \vdash \langle c, \text{Normal } s \rangle = n \Rightarrow t$   
 $\Longrightarrow P$   
**assumes** *someFault*:  $\bigwedge f \ g. \text{find } (\lambda(f,g). s \notin g) \ gs = \text{Some } (f, g) \Longrightarrow t = \text{Fault } f$   
 $\Longrightarrow P$   
**shows** *P*  
**using** *exec-guards noFault someFault*  
**proof** (*induct gs*)  
**case** *Nil*  
**then show** *?case* **by** *simp*  
**next**  
**case** (*Cons pg gs*)  
**obtain** *f g* **where** *pg*: *pg* = (*f, g*)  
**by** (*cases pg*) *auto*  
**show** *?thesis*  
**proof** (*cases s ∈ g*)  
**case** *True*  
**from** *Cons.prem1* **have** *exec-gs*:  $\Gamma \vdash \langle \text{guards } gs \ c, \text{Normal } s \rangle = n \Rightarrow t$   
**by** (*simp add: pg*) (*meson True pg execn-Normal-elim-cases*)  
  
**from** *Cons.hyps* [*OF exec-gs*] *Cons.prem2,3*  
**show** *?thesis*  
**by** (*simp add: pg True*)  
**next**  
**case** *False*  
**from** *Cons.prem1* **have** *t*: *t* = *Fault f*  
**by** (*simp add: pg*) (*meson False pg execn-Normal-elim-cases*)

**from**  $t$  *Cons.prem*(3)  
**show** *?thesis*  
**by** (*simp add: pg False*)  
**qed**  
**qed**

**lemma** *execn-maybe-guard-Normal-elim-cases* [*consumes 1, case-names noFault someFault*]:

**assumes** *exec-guards*:  $\Gamma \vdash \langle \text{maybe-guard } f \ g \ c, \text{Normal } s \rangle = n \Rightarrow t$   
**assumes** *noFault*:  $s \in g \Longrightarrow \Gamma \vdash \langle c, \text{Normal } s \rangle = n \Rightarrow t \Longrightarrow P$   
**assumes** *someFault*:  $s \notin g \Longrightarrow t = \text{Fault } f \Longrightarrow P$   
**shows**  $P$   
**using** *exec-guards noFault someFault*  
**by** (*metis UNIV-I execn-Normal-elim-cases(5) maybe-guard-def*)

**lemma** *execn-guards-noFault*:

**assumes** *exec*:  $\Gamma \vdash \langle c, \text{Normal } s \rangle = n \Rightarrow t$   
**assumes** *noFault*:  $\forall f \ g. (f, g) \in \text{set } gs \longrightarrow s \in g$   
**shows**  $\Gamma \vdash \langle \text{guards } gs \ c, \text{Normal } s \rangle = n \Rightarrow t$   
**using** *exec noFault* **by** (*induct gs*) (*auto intro: execn.intros*)

**lemma** *execn-guards-Fault*:

**assumes** *Fault*:  $\text{find } (\lambda(f,g). s \notin g) \ gs = \text{Some } (f, g)$   
**shows**  $\Gamma \vdash \langle \text{guards } gs \ c, \text{Normal } s \rangle = n \Rightarrow \text{Fault } f$   
**using** *Fault* **by** (*induct gs*) (*auto intro: execn.intros split: prod.splits if-split-asm*)

**lemma** *execn-maybe-guard-noFault*:

**assumes** *exec*:  $\Gamma \vdash \langle c, \text{Normal } s \rangle = n \Rightarrow t$   
**assumes** *noFault*:  $s \in g$   
**shows**  $\Gamma \vdash \langle \text{maybe-guard } f \ g \ c, \text{Normal } s \rangle = n \Rightarrow t$   
**using** *exec noFault*  
**by** (*auto intro: execn.intros simp add: maybe-guard-def*)

**lemma** *execn-maybe-guard-Fault*:

**assumes** *Fault*:  $s \notin g$   
**shows**  $\Gamma \vdash \langle \text{maybe-guard } f \ g \ c, \text{Normal } s \rangle = n \Rightarrow \text{Fault } f$   
**using** *Fault* **by** (*auto simp add: maybe-guard-def intro: execn.intros split: prod.splits if-split-asm*)

**lemma** *execn-guards-DynCom-Normal-elim*:

**assumes** *exec*:  $\Gamma \vdash \langle \text{guards } gs \ (\text{DynCom } c), \text{Normal } s \rangle = n \Rightarrow t$   
**assumes** *call*:  $\Gamma \vdash \langle \text{guards } gs \ (c \ s), \text{Normal } s \rangle = n \Rightarrow t \Longrightarrow P$   
**shows**  $P$   
**using** *exec call* **proof** (*induct gs*)  
**case** *Nil*  
**then show** *?case*  
**apply** *simp*  
**apply** (*erule execn-Normal-elim-cases*)  
**apply** *simp*

```

done
next
case (Cons g gs)
then show ?case
  apply (cases g)
  apply simp
  apply (erule execn-Normal-elim-cases)
  apply simp
  apply (meson execn.Guard)
  apply simp
  apply (meson execn.GuardFault)
done
qed

```

**lemma** *execn-maybe-guard-DynCom-Normal-elim*:  
**assumes** *exec*:  $\Gamma \vdash \langle \text{maybe-guard } f \ g \ (DynCom \ c), \ Normal \ s \rangle =n \Rightarrow t$   
**assumes** *call*:  $\Gamma \vdash \langle \text{maybe-guard } f \ g \ (c \ s), \ Normal \ s \rangle =n \Rightarrow t \Longrightarrow P$   
**shows** *P*  
**using** *exec call*  
**by** (*metis execn.Guard execn.GuardFault execn-Normal-elim-cases(12) execn-Normal-elim-cases(5) maybe-guard-def*)

**lemma** *execn-guards-DynCom*:  
**assumes** *exec-c*:  $\Gamma \vdash \langle \text{guards } gs \ (c \ s), \ Normal \ s \rangle =n \Rightarrow t$   
**shows**  $\Gamma \vdash \langle \text{guards } gs \ (DynCom \ c), \ Normal \ s \rangle =n \Rightarrow t$   
**using** *exec-c* **apply** (*induct gs*)  
**apply** (*fastforce intro: execn.intros*)  
**apply** *simp*  
**by** (*metis execn.Guard execn.GuardFault execn-Normal-elim-cases*)

**lemma** *execn-maybe-guard-DynCom*:  
**assumes** *exec-c*:  $\Gamma \vdash \langle \text{maybe-guard } f \ g \ (c \ s), \ Normal \ s \rangle =n \Rightarrow t$   
**shows**  $\Gamma \vdash \langle \text{maybe-guard } f \ g \ (DynCom \ c), \ Normal \ s \rangle =n \Rightarrow t$   
**using** *exec-c*  
**apply** (*cases g = UNIV*)  
**subgoal**  
**apply** *simp*  
**apply** (*rule execn.intros*)  
**apply** *simp*  
**done**  
**subgoal**  
**apply** (*simp add: maybe-guard-def*)  
**by** (*metis execn.DynCom execn.Guard execn.GuardFault execn-Normal-elim-cases(5)*)  
**done**

**lemma** *execn-dynCall-exn-Normal-elim*:  
**assumes** *exec*:  $\Gamma \vdash \langle \text{dynCall-exn } f \ g \ \text{init } p \ \text{return result-exn } c, \ Normal \ s \rangle =n \Rightarrow t$   
**assumes**  $\Gamma \vdash \langle \text{maybe-guard } f \ g \ (\text{call-exn } \text{init } (p \ s) \ \text{return result-exn } c), \ Normal \ s \rangle$

$=n \Rightarrow t \Longrightarrow P$   
**shows**  $P$   
**using**  $exec$   
**apply** ( $simp$   $add: dynCall-exn-def$ )  
**apply** ( $erule$   $execn-maybe-guard-DynCom-Normal-elim$ )  
**apply**  $fact$   
**done**

**lemma**  $execn-Seq'$ :  
 $\llbracket \Gamma \vdash \langle c1, s \rangle =n \Rightarrow s'; \Gamma \vdash \langle c2, s' \rangle =n \Rightarrow s'' \rrbracket$   
 $\Longrightarrow$   
 $\Gamma \vdash \langle Seq\ c1\ c2, s \rangle =n \Rightarrow s''$   
**apply** ( $cases\ s$ )  
**apply** ( $fastforce$   $intro: execn.intros$ )  
**apply** ( $fastforce$   $dest: execn-Abrupt-end$ )  
**apply** ( $fastforce$   $dest: execn-Fault-end$ )  
**apply** ( $fastforce$   $dest: execn-Stuck-end$ )  
**done**

**lemma**  $execn-mono$ :  
**assumes**  $exec: \Gamma \vdash \langle c, s \rangle =n \Rightarrow t$   
**shows**  $\bigwedge m. n \leq m \Longrightarrow \Gamma \vdash \langle c, s \rangle =m \Rightarrow t$   
**using**  $exec$   
**by** ( $induct$ ) ( $auto$   $intro: execn.intros$   $dest: Suc-le-D$ )

**lemma**  $execn-Suc$ :  
 $\Gamma \vdash \langle c, s \rangle =n \Rightarrow t \Longrightarrow \Gamma \vdash \langle c, s \rangle =Suc\ n \Rightarrow t$   
**by** ( $rule$   $execn-mono$  [ $OF$  -  $le-refl$  [ $THEN$   $le-SucI$ ]])

**lemma**  $execn-assoc$ :  
 $\Gamma \vdash \langle Seq\ c1\ (Seq\ c2\ c3), s \rangle =n \Rightarrow t = \Gamma \vdash \langle Seq\ (Seq\ c1\ c2)\ c3, s \rangle =n \Rightarrow t$   
**by** ( $auto$   $elim!$ :  $execn-elim-cases$   $intro: execn-Seq'$ )

**lemma**  $execn-to-exec$ :  
**assumes**  $execn: \Gamma \vdash \langle c, s \rangle =n \Rightarrow t$   
**shows**  $\Gamma \vdash \langle c, s \rangle \Rightarrow t$   
**using**  $execn$   
**by**  $induct$  ( $auto$   $intro: exec.intros$ )

**lemma**  $exec-to-execn$ :  
**assumes**  $execn: \Gamma \vdash \langle c, s \rangle \Rightarrow t$   
**shows**  $\exists n. \Gamma \vdash \langle c, s \rangle =n \Rightarrow t$   
**using**  $execn$   
**proof** ( $induct$ )  
**case**  $Skip$  **thus**  $?case$  **by** ( $iprover$   $intro: execn.intros$ )  
**next**  
**case**  $Guard$  **thus**  $?case$  **by** ( $iprover$   $intro: execn.intros$ )

```

next
  case GuardFault thus ?case by (iprover intro: execn.intros)
next
  case FaultProp thus ?case by (iprover intro: execn.intros)
next
  case Basic thus ?case by (iprover intro: execn.intros)
next
  case Spec thus ?case by (iprover intro: execn.intros)
next
  case SpecStuck thus ?case by (iprover intro: execn.intros)
next
  case (Seq c1 s s' c2 s'')
  then obtain n m where
     $\Gamma \vdash \langle c1, Normal\ s \rangle = n \Rightarrow s' \Gamma \vdash \langle c2, s^\wedge \rangle = m \Rightarrow s''$ 
    by blast
  then have
     $\Gamma \vdash \langle c1, Normal\ s \rangle = max\ n\ m \Rightarrow s'$ 
     $\Gamma \vdash \langle c2, s^\wedge \rangle = max\ n\ m \Rightarrow s''$ 
    by (auto elim!: execn-mono intro: max.cobounded1 max.cobounded2)
  thus ?case
    by (iprover intro: execn.intros)
next
  case CondTrue thus ?case by (iprover intro: execn.intros)
next
  case CondFalse thus ?case by (iprover intro: execn.intros)
next
  case (WhileTrue s b c s' s'')
  then obtain n m where
     $\Gamma \vdash \langle c, Normal\ s \rangle = n \Rightarrow s' \Gamma \vdash \langle While\ b\ c, s^\wedge \rangle = m \Rightarrow s''$ 
    by blast
  then have
     $\Gamma \vdash \langle c, Normal\ s \rangle = max\ n\ m \Rightarrow s' \Gamma \vdash \langle While\ b\ c, s^\wedge \rangle = max\ n\ m \Rightarrow s''$ 
    by (auto elim!: execn-mono intro: max.cobounded1 max.cobounded2)
  with WhileTrue
  show ?case
    by (iprover intro: execn.intros)
next
  case WhileFalse thus ?case by (iprover intro: execn.intros)
next
  case Call thus ?case by (iprover intro: execn.intros)
next
  case CallUndefined thus ?case by (iprover intro: execn.intros)
next
  case StuckProp thus ?case by (iprover intro: execn.intros)
next
  case DynCom thus ?case by (iprover intro: execn.intros)
next
  case Throw thus ?case by (iprover intro: execn.intros)
next

```

**case** *AbruptProp* **thus** *?case* **by** (*iprover intro: execn.intros*)  
**next**  
**case** (*CatchMatch c1 s s' c2 s''*)  
**then obtain** *n m* **where**  
 $\Gamma \vdash \langle c1, Normal\ s \rangle = n \Rightarrow Abrupt\ s' \Gamma \vdash \langle c2, Normal\ s' \rangle = m \Rightarrow s''$   
**by** *blast*  
**then have**  
 $\Gamma \vdash \langle c1, Normal\ s \rangle = max\ n\ m \Rightarrow Abrupt\ s'$   
 $\Gamma \vdash \langle c2, Normal\ s' \rangle = max\ n\ m \Rightarrow s''$   
**by** (*auto elim!: execn-mono intro: max.cobounded1 max.cobounded2*)  
**with** *CatchMatch.hyps* **show** *?case*  
**by** (*iprover intro: execn.intros*)  
**next**  
**case** *CatchMiss* **thus** *?case* **by** (*iprover intro: execn.intros*)  
**qed**

**theorem** *exec-iff-execn*:  $(\Gamma \vdash \langle c, s \rangle \Rightarrow t) = (\exists n. \Gamma \vdash \langle c, s \rangle = n \Rightarrow t)$   
**by** (*iprover intro: exec-to-execn execn-to-exec*)

**definition** *nfinal-notin*::  $(s, p, f)\ body \Rightarrow (s, p, f)\ com \Rightarrow (s, f)\ xstate \Rightarrow nat$   
 $\Rightarrow (s, f)\ xstate\ set \Rightarrow bool$   
 $(\langle + \langle -, - \rangle \Rightarrow \notin \rightarrow [60, 20, 98, 65, 60] 89)$  **where**  
 $\Gamma \vdash \langle c, s \rangle = n \Rightarrow \notin T = (\forall t. \Gamma \vdash \langle c, s \rangle = n \Rightarrow t \rightarrow t \notin T)$

**definition** *final-notin*::  $(s, p, f)\ body \Rightarrow (s, p, f)\ com \Rightarrow (s, f)\ xstate$   
 $\Rightarrow (s, f)\ xstate\ set \Rightarrow bool$   
 $(\langle + \langle -, - \rangle \Rightarrow \notin \rightarrow [60, 20, 98, 60] 89)$  **where**  
 $\Gamma \vdash \langle c, s \rangle \Rightarrow \notin T = (\forall t. \Gamma \vdash \langle c, s \rangle \Rightarrow t \rightarrow t \notin T)$

**lemma** *final-notinI*:  $\llbracket \bigwedge t. \Gamma \vdash \langle c, s \rangle \Rightarrow t \implies t \notin T \rrbracket \implies \Gamma \vdash \langle c, s \rangle \Rightarrow \notin T$   
**by** (*simp add: final-notin-def*)

**lemma** *noFaultStuck-Call-body'*:  $p \in dom\ \Gamma \implies$   
 $\Gamma \vdash \langle Call\ p, Normal\ s \rangle \Rightarrow \notin (\{Stuck\} \cup Fault\ '(-F)) =$   
 $\Gamma \vdash \langle the\ (\Gamma\ p), Normal\ s \rangle \Rightarrow \notin (\{Stuck\} \cup Fault\ '(-F))$   
**by** (*clarsimp simp add: final-notin-def exec-Call-body*)

**lemma** *noFault-startn*:  
**assumes** *execn*:  $\Gamma \vdash \langle c, s \rangle = n \Rightarrow t$  **and** *t*:  $t \neq Fault\ f$   
**shows**  $s \neq Fault\ f$   
**using** *execn t* **by** (*induct*) *auto*

**lemma** *noFault-start*:  
**assumes** *exec*:  $\Gamma \vdash \langle c, s \rangle \Rightarrow t$  **and** *t*:  $t \neq Fault\ f$   
**shows**  $s \neq Fault\ f$   
**using** *exec t* **by** (*induct*) *auto*

**lemma** *noStuck-startn*:

**assumes**  $execn: \Gamma \vdash \langle c, s \rangle = n \Rightarrow t$  **and**  $t: t \neq Stuck$   
**shows**  $s \neq Stuck$   
**using**  $execn$   $t$  **by**  $(induct)$   $auto$

**lemma**  $noStuck-start$ :  
**assumes**  $exec: \Gamma \vdash \langle c, s \rangle \Rightarrow t$  **and**  $t: t \neq Stuck$   
**shows**  $s \neq Stuck$   
**using**  $exec$   $t$  **by**  $(induct)$   $auto$

**lemma**  $noAbrupt-startn$ :  
**assumes**  $execn: \Gamma \vdash \langle c, s \rangle = n \Rightarrow t$  **and**  $t: \forall t'. t \neq Abrupt t'$   
**shows**  $s \neq Abrupt s'$   
**using**  $execn$   $t$  **by**  $(induct)$   $auto$

**lemma**  $noAbrupt-start$ :  
**assumes**  $exec: \Gamma \vdash \langle c, s \rangle \Rightarrow t$  **and**  $t: \forall t'. t \neq Abrupt t'$   
**shows**  $s \neq Abrupt s'$   
**using**  $exec$   $t$  **by**  $(induct)$   $auto$

**lemma**  $noFaultn-startD$ :  $\Gamma \vdash \langle c, s \rangle = n \Rightarrow Normal\ t \Longrightarrow s \neq Fault\ f$   
**by**  $(auto\ dest: noFault-startn)$

**lemma**  $noFaultn-startD'$ :  $t \neq Fault\ f \Longrightarrow \Gamma \vdash \langle c, s \rangle = n \Rightarrow t \Longrightarrow s \neq Fault\ f$   
**by**  $(auto\ dest: noFault-startn)$

**lemma**  $noFault-startD$ :  $\Gamma \vdash \langle c, s \rangle \Rightarrow Normal\ t \Longrightarrow s \neq Fault\ f$   
**by**  $(auto\ dest: noFault-start)$

**lemma**  $noFault-startD'$ :  $t \neq Fault\ f \Longrightarrow \Gamma \vdash \langle c, s \rangle \Rightarrow t \Longrightarrow s \neq Fault\ f$   
**by**  $(auto\ dest: noFault-start)$

**lemma**  $noStuckn-startD$ :  $\Gamma \vdash \langle c, s \rangle = n \Rightarrow Normal\ t \Longrightarrow s \neq Stuck$   
**by**  $(auto\ dest: noStuck-startn)$

**lemma**  $noStuckn-startD'$ :  $t \neq Stuck \Longrightarrow \Gamma \vdash \langle c, s \rangle = n \Rightarrow t \Longrightarrow s \neq Stuck$   
**by**  $(auto\ dest: noStuck-startn)$

**lemma**  $noStuck-startD$ :  $\Gamma \vdash \langle c, s \rangle \Rightarrow Normal\ t \Longrightarrow s \neq Stuck$   
**by**  $(auto\ dest: noStuck-start)$

**lemma**  $noStuck-startD'$ :  $t \neq Stuck \Longrightarrow \Gamma \vdash \langle c, s \rangle \Rightarrow t \Longrightarrow s \neq Stuck$   
**by**  $(auto\ dest: noStuck-start)$

**lemma**  $noAbruptn-startD$ :  $\Gamma \vdash \langle c, s \rangle = n \Rightarrow Normal\ t \Longrightarrow s \neq Abrupt\ s'$   
**by**  $(auto\ dest: noAbrupt-startn)$

**lemma**  $noAbrupt-startD$ :  $\Gamma \vdash \langle c, s \rangle \Rightarrow Normal\ t \Longrightarrow s \neq Abrupt\ s'$   
**by**  $(auto\ dest: noAbrupt-start)$

**lemma** *noFaultnI*:  $\llbracket \bigwedge t. \Gamma \vdash \langle c, s \rangle = n \Rightarrow t \implies t \neq \text{Fault } f \rrbracket \implies \Gamma \vdash \langle c, s \rangle = n \Rightarrow \notin \{ \text{Fault } f \}$

**by** (*simp add: nfinal-notin-def*)

**lemma** *noFaultnI'*:

**assumes** *contr*:  $\Gamma \vdash \langle c, s \rangle = n \Rightarrow \text{Fault } f \implies \text{False}$

**shows**  $\Gamma \vdash \langle c, s \rangle = n \Rightarrow \notin \{ \text{Fault } f \}$

**proof** (*rule noFaultnI*)

**fix** *t* **assume**  $\Gamma \vdash \langle c, s \rangle = n \Rightarrow t$

**with** *contr* **show**  $t \neq \text{Fault } f$

**by** (*cases t=Fault f*) *auto*

**qed**

**lemma** *noFaultn-def'*:  $\Gamma \vdash \langle c, s \rangle = n \Rightarrow \notin \{ \text{Fault } f \} = (\neg \Gamma \vdash \langle c, s \rangle = n \Rightarrow \text{Fault } f)$

**apply** *rule*

**apply** (*fastforce simp add: nfinal-notin-def*)

**apply** (*fastforce intro: noFaultnI'*)

**done**

**lemma** *noStucknI*:  $\llbracket \bigwedge t. \Gamma \vdash \langle c, s \rangle = n \Rightarrow t \implies t \neq \text{Stuck} \rrbracket \implies \Gamma \vdash \langle c, s \rangle = n \Rightarrow \notin \{ \text{Stuck} \}$

**by** (*simp add: nfinal-notin-def*)

**lemma** *noStucknI'*:

**assumes** *contr*:  $\Gamma \vdash \langle c, s \rangle = n \Rightarrow \text{Stuck} \implies \text{False}$

**shows**  $\Gamma \vdash \langle c, s \rangle = n \Rightarrow \notin \{ \text{Stuck} \}$

**proof** (*rule noStucknI*)

**fix** *t* **assume**  $\Gamma \vdash \langle c, s \rangle = n \Rightarrow t$

**with** *contr* **show**  $t \neq \text{Stuck}$

**by** (*cases t*) *auto*

**qed**

**lemma** *noStuckn-def'*:  $\Gamma \vdash \langle c, s \rangle = n \Rightarrow \notin \{ \text{Stuck} \} = (\neg \Gamma \vdash \langle c, s \rangle = n \Rightarrow \text{Stuck})$

**apply** *rule*

**apply** (*fastforce simp add: nfinal-notin-def*)

**apply** (*fastforce intro: noStucknI'*)

**done**

**lemma** *noFaultI*:  $\llbracket \bigwedge t. \Gamma \vdash \langle c, s \rangle \Rightarrow t \implies t \neq \text{Fault } f \rrbracket \implies \Gamma \vdash \langle c, s \rangle \Rightarrow \notin \{ \text{Fault } f \}$

**by** (*simp add: final-notin-def*)

**lemma** *noFaultI'*:

**assumes** *contr*:  $\Gamma \vdash \langle c, s \rangle \Rightarrow \text{Fault } f \implies \text{False}$

**shows**  $\Gamma \vdash \langle c, s \rangle \Rightarrow \notin \{ \text{Fault } f \}$

**proof** (*rule noFaultI*)

**fix** *t* **assume**  $\Gamma \vdash \langle c, s \rangle \Rightarrow t$

**with** *contr* **show**  $t \neq \text{Fault } f$

**by** (*cases t=Fault f*) *auto*

**qed**



**lemma** *noFaultE*:

$\llbracket \Gamma \vdash \langle c, s \rangle \Rightarrow \notin \{ \text{Fault } f \}; \Gamma \vdash \langle c, s \rangle \Rightarrow \text{Fault } f \rrbracket \Longrightarrow P$   
**by** (*auto simp add: final-notin-def*)

**lemma** *noFault-def'*:  $\Gamma \vdash \langle c, s \rangle \Rightarrow \notin \{ \text{Fault } f \} = (\neg \Gamma \vdash \langle c, s \rangle \Rightarrow \text{Fault } f)$

**apply** *rule*

**apply** (*fastforce simp add: final-notin-def*)

**apply** (*fastforce intro: noFaultI'*)

**done**

**lemma** *noStuckI*:  $\llbracket \bigwedge t. \Gamma \vdash \langle c, s \rangle \Rightarrow t \Longrightarrow t \neq \text{Stuck} \rrbracket \Longrightarrow \Gamma \vdash \langle c, s \rangle \Rightarrow \notin \{ \text{Stuck} \}$

**by** (*simp add: final-notin-def*)

**lemma** *noStuckI'*:

**assumes** *contr*:  $\Gamma \vdash \langle c, s \rangle \Rightarrow \text{Stuck} \Longrightarrow \text{False}$

**shows**  $\Gamma \vdash \langle c, s \rangle \Rightarrow \notin \{ \text{Stuck} \}$

**proof** (*rule noStuckI*)

**fix** *t* **assume**  $\Gamma \vdash \langle c, s \rangle \Rightarrow t$

**with** *contr* **show**  $t \neq \text{Stuck}$

**by** (*cases t*) *auto*

**qed**

**lemma** *noStuckE*:

$\llbracket \Gamma \vdash \langle c, s \rangle \Rightarrow \notin \{ \text{Stuck} \}; \Gamma \vdash \langle c, s \rangle \Rightarrow \text{Stuck} \rrbracket \Longrightarrow P$

**by** (*auto simp add: final-notin-def*)

**lemma** *noStuck-def'*:  $\Gamma \vdash \langle c, s \rangle \Rightarrow \notin \{ \text{Stuck} \} = (\neg \Gamma \vdash \langle c, s \rangle \Rightarrow \text{Stuck})$

**apply** *rule*

**apply** (*fastforce simp add: final-notin-def*)

**apply** (*fastforce intro: noStuckI'*)

**done**

**lemma** *noFaultn-execD*:  $\llbracket \Gamma \vdash \langle c, s \rangle = n \Rightarrow \notin \{ \text{Fault } f \}; \Gamma \vdash \langle c, s \rangle = n \Rightarrow t \rrbracket \Longrightarrow t \neq \text{Fault } f$

**by** (*simp add: nfinal-notin-def*)

**lemma** *noFault-execD*:  $\llbracket \Gamma \vdash \langle c, s \rangle \Rightarrow \notin \{ \text{Fault } f \}; \Gamma \vdash \langle c, s \rangle \Rightarrow t \rrbracket \Longrightarrow t \neq \text{Fault } f$

**by** (*simp add: final-notin-def*)

**lemma** *noFaultn-exec-startD*:  $\llbracket \Gamma \vdash \langle c, s \rangle = n \Rightarrow \notin \{ \text{Fault } f \}; \Gamma \vdash \langle c, s \rangle = n \Rightarrow t \rrbracket \Longrightarrow s \neq \text{Fault } f$

**by** (*auto simp add: nfinal-notin-def dest: noFaultn-startD*)

**lemma** *noFault-exec-startD*:  $\llbracket \Gamma \vdash \langle c, s \rangle \Rightarrow \notin \{ \text{Fault } f \}; \Gamma \vdash \langle c, s \rangle \Rightarrow t \rrbracket \Longrightarrow s \neq \text{Fault } f$

**by** (*auto simp add: final-notin-def dest: noFault-startD*)

**lemma** *noStuckn-execD*:  $\llbracket \Gamma \vdash \langle c, s \rangle = n \Rightarrow \notin \{ \text{Stuck} \}; \Gamma \vdash \langle c, s \rangle = n \Rightarrow t \rrbracket \Longrightarrow t \neq \text{Stuck}$

by (simp add: nfinal-notin-def)

**lemma** noStuck-execD:  $\llbracket \Gamma \vdash \langle c, s \rangle \Rightarrow \notin \{Stuck\}; \Gamma \vdash \langle c, s \rangle \Rightarrow t \rrbracket \Longrightarrow t \neq Stuck$   
by (simp add: final-notin-def)

**lemma** noStuckn-exec-startD:  $\llbracket \Gamma \vdash \langle c, s \rangle = n \Rightarrow \notin \{Stuck\}; \Gamma \vdash \langle c, s \rangle = n \Rightarrow t \rrbracket \Longrightarrow s \neq Stuck$   
by (auto simp add: nfinal-notin-def dest: noStuckn-startD)

**lemma** noStuck-exec-startD:  $\llbracket \Gamma \vdash \langle c, s \rangle \Rightarrow \notin \{Stuck\}; \Gamma \vdash \langle c, s \rangle \Rightarrow t \rrbracket \Longrightarrow s \neq Stuck$   
by (auto simp add: final-notin-def dest: noStuck-startD)

**lemma** noFaultStuckn-execD:  
 $\llbracket \Gamma \vdash \langle c, s \rangle = n \Rightarrow \notin \{Fault\ True, Fault\ False, Stuck\}; \Gamma \vdash \langle c, s \rangle = n \Rightarrow t \rrbracket \Longrightarrow$   
 $t \notin \{Fault\ True, Fault\ False, Stuck\}$   
by (simp add: nfinal-notin-def)

**lemma** noFaultStuck-execD:  $\llbracket \Gamma \vdash \langle c, s \rangle \Rightarrow \notin \{Fault\ True, Fault\ False, Stuck\}; \Gamma \vdash \langle c, s \rangle \Rightarrow t \rrbracket$   
 $\Longrightarrow t \notin \{Fault\ True, Fault\ False, Stuck\}$   
by (simp add: final-notin-def)

**lemma** noFaultStuckn-exec-startD:  
 $\llbracket \Gamma \vdash \langle c, s \rangle = n \Rightarrow \notin \{Fault\ True, Fault\ False, Stuck\}; \Gamma \vdash \langle c, s \rangle = n \Rightarrow t \rrbracket$   
 $\Longrightarrow s \notin \{Fault\ True, Fault\ False, Stuck\}$   
by (auto simp add: nfinal-notin-def )

**lemma** noFaultStuck-exec-startD:  
 $\llbracket \Gamma \vdash \langle c, s \rangle \Rightarrow \notin \{Fault\ True, Fault\ False, Stuck\}; \Gamma \vdash \langle c, s \rangle \Rightarrow t \rrbracket$   
 $\Longrightarrow s \notin \{Fault\ True, Fault\ False, Stuck\}$   
by (auto simp add: final-notin-def )

**lemma** noStuck-Call:  
**assumes** noStuck:  $\Gamma \vdash \langle Call\ p, Normal\ s \rangle \Rightarrow \notin \{Stuck\}$   
**shows**  $p \in dom\ \Gamma$   
**proof** (cases  $p \in dom\ \Gamma$ )  
  **case** True **thus** ?thesis **by** simp  
**next**  
  **case** False  
  **hence**  $\Gamma\ p = None$  **by** auto  
  **hence**  $\Gamma \vdash \langle Call\ p, Normal\ s \rangle \Rightarrow Stuck$   
  **by** (rule exec.CallUndefined)  
  **with** noStuck **show** ?thesis  
  **by** (auto simp add: final-notin-def)  
**qed**

**lemma** Guard-noFaultStuckD:  
**assumes**  $\Gamma \vdash \langle Guard\ f\ g\ c, Normal\ s \rangle \Rightarrow \notin (\{Stuck\} \cup Fault\ '(-F))$   
**assumes**  $f \notin F$

**shows**  $s \in g$   
**using** *assms*  
**by** (*auto simp add: final-notin-def intro: exec.intros*)

**lemma** *final-notin-to-finaln*:  
**assumes** *notin*:  $\Gamma \vdash \langle c, s \rangle \Rightarrow \notin T$   
**shows**  $\Gamma \vdash \langle c, s \rangle = n \Rightarrow \notin T$   
**proof** (*clarsimp simp add: nfinal-notin-def*)  
**fix** *t* **assume**  $\Gamma \vdash \langle c, s \rangle = n \Rightarrow t$  **and**  $t \in T$   
**with** *notin* **show** *False*  
**by** (*auto intro: execn-to-exec simp add: final-notin-def*)  
**qed**

**lemma** *noFault-Call-body*:  
 $\Gamma \ p = \text{Some } \text{bdy} \Rightarrow$   
 $\Gamma \vdash \langle \text{Call } p, \text{Normal } s \rangle \Rightarrow \notin \{\text{Fault } f\} =$   
 $\Gamma \vdash \langle \text{the } (\Gamma \ p), \text{Normal } s \rangle \Rightarrow \notin \{\text{Fault } f\}$   
**by** (*simp add: noFault-def' exec-Call-body*)

**lemma** *noStuck-Call-body*:  
 $\Gamma \ p = \text{Some } \text{bdy} \Rightarrow$   
 $\Gamma \vdash \langle \text{Call } p, \text{Normal } s \rangle \Rightarrow \notin \{\text{Stuck}\} =$   
 $\Gamma \vdash \langle \text{the } (\Gamma \ p), \text{Normal } s \rangle \Rightarrow \notin \{\text{Stuck}\}$   
**by** (*simp add: noStuck-def' exec-Call-body*)

**lemma** *exec-final-notin-to-execn*:  $\Gamma \vdash \langle c, s \rangle \Rightarrow \notin T \Longrightarrow \Gamma \vdash \langle c, s \rangle = n \Rightarrow \notin T$   
**by** (*auto simp add: final-notin-def nfinal-notin-def dest: execn-to-exec*)

**lemma** *execn-final-notin-to-exec*:  $\forall n. \Gamma \vdash \langle c, s \rangle = n \Rightarrow \notin T \Longrightarrow \Gamma \vdash \langle c, s \rangle \Rightarrow \notin T$   
**by** (*auto simp add: final-notin-def nfinal-notin-def dest: exec-to-execn*)

**lemma** *exec-final-notin-iff-execn*:  $\Gamma \vdash \langle c, s \rangle \Rightarrow \notin T = (\forall n. \Gamma \vdash \langle c, s \rangle = n \Rightarrow \notin T)$   
**by** (*auto intro: exec-final-notin-to-execn execn-final-notin-to-exec*)

**lemma** *Seq-NoFaultStuckD2*:  
**assumes** *noabort*:  $\Gamma \vdash \langle \text{Seq } c1 \ c2, s \rangle \Rightarrow \notin (\{\text{Stuck}\} \cup \text{Fault } ' F)$   
**shows**  $\forall t. \Gamma \vdash \langle c1, s \rangle \Rightarrow t \longrightarrow t \notin (\{\text{Stuck}\} \cup \text{Fault } ' F) \longrightarrow$   
 $\Gamma \vdash \langle c2, t \rangle \Rightarrow \notin (\{\text{Stuck}\} \cup \text{Fault } ' F)$

**using** *noabort*

**by** (*auto simp add: final-notin-def intro: exec-Seq'*) **lemma** *Seq-NoFaultStuckD1*:

**assumes** *noabort*:  $\Gamma \vdash \langle \text{Seq } c1 \ c2, s \rangle \Rightarrow \notin (\{\text{Stuck}\} \cup \text{Fault } ' F)$   
**shows**  $\Gamma \vdash \langle c1, s \rangle \Rightarrow \notin (\{\text{Stuck}\} \cup \text{Fault } ' F)$

**proof** (*rule final-notinI*)

**fix** *t*

**assume** *exec-c1*:  $\Gamma \vdash \langle c1, s \rangle \Rightarrow t$

**show**  $t \notin \{\text{Stuck}\} \cup \text{Fault } ' F$

**proof**

**assume**  $t \in \{\text{Stuck}\} \cup \text{Fault } ' F$

```

moreover
{
  assume  $t = Stuck$ 
  with  $exec-c1$ 
  have  $\Gamma \vdash \langle Seq\ c1\ c2, s \rangle \Rightarrow Stuck$ 
    by ( $auto\ intro: exec-Seq'$ )
  with  $noabort$  have  $False$ 
    by ( $auto\ simp\ add: final-notin-def$ )
  hence  $False ..$ 
}
moreover
{
  assume  $t \in Fault\ 'F$ 
  then obtain  $f$  where
   $t = Fault\ f$  and  $f: f \in F$ 
    by  $auto$ 
  from  $t\ exec-c1$ 
  have  $\Gamma \vdash \langle Seq\ c1\ c2, s \rangle \Rightarrow Fault\ f$ 
    by ( $auto\ intro: exec-Seq'$ )
  with  $noabort\ f$  have  $False$ 
    by ( $auto\ simp\ add: final-notin-def$ )
  hence  $False ..$ 
}
ultimately show  $False$  by  $auto$ 
qed
qed

```

```

lemma  $Seq-NoFaultStuckD2'$ :
  assumes  $noabort: \Gamma \vdash \langle Seq\ c1\ c2, s \rangle \Rightarrow \notin(\{Stuck\} \cup Fault\ 'F)$ 
  shows  $\forall t. \Gamma \vdash \langle c1, s \rangle \Rightarrow t \longrightarrow t \notin (\{Stuck\} \cup Fault\ 'F) \longrightarrow$ 
     $\Gamma \vdash \langle c2, t \rangle \Rightarrow \notin(\{Stuck\} \cup Fault\ 'F)$ 
using  $noabort$ 
by ( $auto\ simp\ add: final-notin-def\ intro: exec-Seq'$ )

```

### 3.3 Lemmas about *sequence*, *flatten* and *Language.normalize*

```

lemma  $execn-sequence-app: \bigwedge s\ s'\ t.$ 
 $\llbracket \Gamma \vdash \langle sequence\ Seq\ xs, Normal\ s \rangle =n \Rightarrow s'; \Gamma \vdash \langle sequence\ Seq\ ys, s' \rangle =n \Rightarrow t \rrbracket$ 
 $\Longrightarrow \Gamma \vdash \langle sequence\ Seq\ (xs@ys), Normal\ s \rangle =n \Rightarrow t$ 
proof ( $induct\ xs$ )
  case  $Nil$ 
  thus  $?case$  by ( $auto\ elim: execn-Normal-elim-cases$ )
next
  case ( $Cons\ x\ xs$ )
  have  $exec-x-xs: \Gamma \vdash \langle sequence\ Seq\ (x \# xs), Normal\ s \rangle =n \Rightarrow s'$  by  $fact$ 
  have  $exec-ys: \Gamma \vdash \langle sequence\ Seq\ ys, s' \rangle =n \Rightarrow t$  by  $fact$ 
  show  $?case$ 
  proof ( $cases\ xs$ )
    case  $Nil$ 

```

```

with exec-x-xs have  $\Gamma \vdash \langle x, \text{Normal } s \rangle = n \Rightarrow s'$ 
  by (auto elim: execn-Normal-elim-cases)
with Nil exec-ys show ?thesis
  by (cases ys) (auto intro: execn.intros elim: execn-elim-cases)
next
case Cons
with exec-x-xs
obtain  $s''$  where
  exec-x:  $\Gamma \vdash \langle x, \text{Normal } s \rangle = n \Rightarrow s''$  and
  exec-xs:  $\Gamma \vdash \langle \text{sequence Seq } xs, s'' \rangle = n \Rightarrow s'$ 
  by (auto elim: execn-Normal-elim-cases)
show ?thesis
proof (cases s'')
  case (Normal s''')
  from Cons.hyps [OF exec-xs [simplified Normal] exec-ys]
  have  $\Gamma \vdash \langle \text{sequence Seq } (xs @ ys), \text{Normal } s''' \rangle = n \Rightarrow t$  .
  with Cons exec-x Normal
  show ?thesis
  by (auto intro: execn.intros)
next
  case (Abrupt s''')
  with exec-xs have  $s' = \text{Abrupt } s'''$ 
    by (auto dest: execn-Abrupt-end)
  with exec-ys have  $t = \text{Abrupt } s'''$ 
    by (auto dest: execn-Abrupt-end)
  with exec-x Abrupt Cons show ?thesis
  by (auto intro: execn.intros)
next
  case (Fault f)
  with exec-xs have  $s' = \text{Fault } f$ 
    by (auto dest: execn-Fault-end)
  with exec-ys have  $t = \text{Fault } f$ 
    by (auto dest: execn-Fault-end)
  with exec-x Fault Cons show ?thesis
  by (auto intro: execn.intros)
next
  case Stuck
  with exec-xs have  $s' = \text{Stuck}$ 
    by (auto dest: execn-Stuck-end)
  with exec-ys have  $t = \text{Stuck}$ 
    by (auto dest: execn-Stuck-end)
  with exec-x Stuck Cons show ?thesis
  by (auto intro: execn.intros)
qed
qed
qed

lemma execn-sequence-appD:  $\bigwedge s t. \Gamma \vdash \langle \text{sequence Seq } (xs @ ys), \text{Normal } s \rangle = n \Rightarrow t$ 
 $\Rightarrow$ 

```

$\exists s'. \Gamma \vdash \langle \text{sequence Seq } xs, \text{Normal } s \rangle = n \Rightarrow s' \wedge \Gamma \vdash \langle \text{sequence Seq } ys, s' \rangle = n \Rightarrow$

$t$

**proof** (*induct xs*)

**case** *Nil*

**thus** *?case*

**by** (*auto intro: execn.intros*)

**next**

**case** (*Cons x xs*)

**have** *exec-app*:  $\Gamma \vdash \langle \text{sequence Seq } ((x \# xs) @ ys), \text{Normal } s \rangle = n \Rightarrow t$  **by fact**

**show** *?case*

**proof** (*cases xs*)

**case** *Nil*

**with** *exec-app* **show** *?thesis*

**by** (*cases ys*) (*auto elim: execn-Normal-elim-cases intro: execn-Skip'*)

**next**

**case** *Cons*

**with** *exec-app* **obtain**  $s'$  **where**

*exec-x*:  $\Gamma \vdash \langle x, \text{Normal } s \rangle = n \Rightarrow s'$  **and**

*exec-xs-ys*:  $\Gamma \vdash \langle \text{sequence Seq } (xs @ ys), s' \rangle = n \Rightarrow t$

**by** (*auto elim: execn-Normal-elim-cases*)

**show** *?thesis*

**proof** (*cases s'*)

**case** (*Normal s''*)

**from** *Cons.hyps* [*OF exec-xs-ys [simplified Normal]*] *Normal exec-x Cons*

**show** *?thesis*

**by** (*auto intro: execn.intros*)

**next**

**case** (*Abrupt s''*)

**with** *exec-xs-ys* **have**  $t = \text{Abrupt } s''$

**by** (*auto dest: execn-Abrupt-end*)

**with** *Abrupt exec-x Cons*

**show** *?thesis*

**by** (*auto intro: execn.intros*)

**next**

**case** (*Fault f*)

**with** *exec-xs-ys* **have**  $t = \text{Fault } f$

**by** (*auto dest: execn-Fault-end*)

**with** *Fault exec-x Cons*

**show** *?thesis*

**by** (*auto intro: execn.intros*)

**next**

**case** *Stuck*

**with** *exec-xs-ys* **have**  $t = \text{Stuck}$

**by** (*auto dest: execn-Stuck-end*)

**with** *Stuck exec-x Cons*

**show** *?thesis*

**by** (*auto intro: execn.intros*)

**qed**

**qed**

**qed**

**lemma** *execn-sequence-appE* [*consumes 1*]:

$\llbracket \Gamma \vdash \langle \text{sequence Seq } (xs \text{ @ } ys), \text{Normal } s \rangle = n \Rightarrow t; \wedge s'. \llbracket \Gamma \vdash \langle \text{sequence Seq } xs, \text{Normal } s \rangle = n \Rightarrow s'; \Gamma \vdash \langle \text{sequence Seq } ys, s' \rangle = n \Rightarrow t \rrbracket \Longrightarrow$   
 $P$   
 $\rrbracket \Longrightarrow P$   
**by** (*auto dest: execn-sequence-appD*)

**lemma** *execn-to-execn-sequence-flatten*:

**assumes** *exec*:  $\Gamma \vdash \langle c, s \rangle = n \Rightarrow t$   
**shows**  $\Gamma \vdash \langle \text{sequence Seq } (\text{flatten } c), s \rangle = n \Rightarrow t$   
**using** *exec*  
**proof** *induct*  
**case** (*Seq c1 c2 n s s' s''*) **thus** *?case*  
**by** (*auto intro: execn.intros execn-sequence-app*)  
**qed** (*auto intro: execn.intros*)

**lemma** *execn-to-execn-normalize*:

**assumes** *exec*:  $\Gamma \vdash \langle c, s \rangle = n \Rightarrow t$   
**shows**  $\Gamma \vdash \langle \text{normalize } c, s \rangle = n \Rightarrow t$   
**using** *exec*  
**proof** *induct*  
**case** (*Seq c1 c2 n s s' s''*) **thus** *?case*  
**by** (*auto intro: execn-to-execn-sequence-flatten execn-sequence-app*)  
**qed** (*auto intro: execn.intros*)

**lemma** *execn-sequence-flatten-to-execn*:

**shows**  $\wedge s t. \Gamma \vdash \langle \text{sequence Seq } (\text{flatten } c), s \rangle = n \Rightarrow t \Longrightarrow \Gamma \vdash \langle c, s \rangle = n \Rightarrow t$   
**proof** (*induct c*)  
**case** (*Seq c1 c2*)  
**have** *exec-seq*:  $\Gamma \vdash \langle \text{sequence Seq } (\text{flatten } (\text{Seq } c1 \text{ } c2)), s \rangle = n \Rightarrow t$  **by fact**  
**show** *?case*  
**proof** (*cases s*)  
**case** (*Normal s'*)  
**with** *exec-seq* **obtain** *s''* **where**  
 $\Gamma \vdash \langle \text{sequence Seq } (\text{flatten } c1), \text{Normal } s' \rangle = n \Rightarrow s''$  **and**  
 $\Gamma \vdash \langle \text{sequence Seq } (\text{flatten } c2), s' \rangle = n \Rightarrow t$   
**by** (*auto elim: execn-sequence-appE*)  
**with** *Seq.hyps Normal*  
**show** *?thesis*  
**by** (*fastforce intro: execn.intros*)  
**next**  
**case** *Abrupt*  
**with** *exec-seq*  
**show** *?thesis* **by** (*auto intro: execn.intros dest: execn-Abrupt-end*)  
**next**

```

    case Fault
    with exec-seq
    show ?thesis by (auto intro: execn.intros dest: execn-Fault-end)
  next
    case Stuck
    with exec-seq
    show ?thesis by (auto intro: execn.intros dest: execn-Stuck-end)
  qed
qed auto

lemma execn-normalize-to-execn:
  shows  $\bigwedge s t n. \Gamma \vdash \langle \text{normalize } c, s \rangle = n \Rightarrow t \implies \Gamma \vdash \langle c, s \rangle = n \Rightarrow t$ 
proof (induct c)
  case Skip thus ?case by simp
next
  case Basic thus ?case by simp
next
  case Spec thus ?case by simp
next
  case (Seq c1 c2)
  have  $\Gamma \vdash \langle \text{normalize } (\text{Seq } c1 \ c2), s \rangle = n \Rightarrow t$  by fact
  hence exec-norm-seq:
     $\Gamma \vdash \langle \text{sequence Seq } (\text{flatten } (\text{normalize } c1)) \ @ \ \text{flatten } (\text{normalize } c2), s \rangle = n \Rightarrow t$ 
  by simp
  show ?case
  proof (cases s)
    case (Normal s')
    with exec-norm-seq obtain s'' where
      exec-norm-c1:  $\Gamma \vdash \langle \text{sequence Seq } (\text{flatten } (\text{normalize } c1)), \text{Normal } s' \rangle = n \Rightarrow s''$ 
    and
      exec-norm-c2:  $\Gamma \vdash \langle \text{sequence Seq } (\text{flatten } (\text{normalize } c2)), s' \rangle = n \Rightarrow t$ 
    by (auto elim: execn-sequence-appE)
    from execn-sequence-flatten-to-execn [OF exec-norm-c1]
      execn-sequence-flatten-to-execn [OF exec-norm-c2] Seq.hyps Normal
    show ?thesis
    by (fastforce intro: execn.intros)
  next
    case (Abrupt s')
    with exec-norm-seq have  $t = \text{Abrupt } s'$ 
    by (auto dest: execn-Abrupt-end)
    with Abrupt show ?thesis
    by (auto intro: execn.intros)
  next
    case (Fault f)
    with exec-norm-seq have  $t = \text{Fault } f$ 
    by (auto dest: execn-Fault-end)
    with Fault show ?thesis
    by (auto intro: execn.intros)
  next

```



```

    case Stuck
    with exec-norm-seq have  $t = \text{Stuck}$ 
      by (auto dest: execn-Stuck-end)
    with Stuck show ?thesis
      by (auto intro: execn.intros)
  qed
next
  case Cond thus ?case
    by (auto intro: execn.intros elim!: execn-elim-cases)
next
  case (While  $b\ c$ )
  have  $\Gamma \vdash \langle \text{normalize } (While\ b\ c), s \rangle = n \Rightarrow t$  by fact
  hence exec-norm-w:  $\Gamma \vdash \langle While\ b\ (\text{normalize } c), s \rangle = n \Rightarrow t$ 
    by simp
  {
    fix  $s\ t\ w$ 
    assume exec-w:  $\Gamma \vdash \langle w, s \rangle = n \Rightarrow t$ 
    have  $w = While\ b\ (\text{normalize } c) \Longrightarrow \Gamma \vdash \langle While\ b\ c, s \rangle = n \Rightarrow t$ 
      using exec-w
    proof (induct)
      case (WhileTrue  $s\ b'\ c'\ n\ w\ t$ )
      from WhileTrue obtain
        s-in-b:  $s \in b$  and
        exec-c:  $\Gamma \vdash \langle \text{normalize } c, Normal\ s \rangle = n \Rightarrow w$  and
        hyp-w:  $\Gamma \vdash \langle While\ b\ c, w \rangle = n \Rightarrow t$ 
      by simp
      from While.hyps [OF exec-c]
      have  $\Gamma \vdash \langle c, Normal\ s \rangle = n \Rightarrow w$ 
        by simp
      with hyp-w s-in-b
      have  $\Gamma \vdash \langle While\ b\ c, Normal\ s \rangle = n \Rightarrow t$ 
        by (auto intro: execn.intros)
      with WhileTrue show ?case by simp
    qed (auto intro: execn.intros)
  }
  from this [OF exec-norm-w]
  show ?case
    by simp
next
  case Call thus ?case by simp
next
  case DynCom thus ?case by (auto intro: execn.intros elim!: execn-elim-cases)
next
  case Guard thus ?case by (auto intro: execn.intros elim!: execn-elim-cases)
next
  case Throw thus ?case by simp
next
  case Catch thus ?case by (fastforce intro: execn.intros elim!: execn-elim-cases)
qed

```

**lemma** *execn-normalize-iff-execn*:

$\Gamma \vdash \langle \text{normalize } c, s \rangle = n \Rightarrow t = \Gamma \vdash \langle c, s \rangle = n \Rightarrow t$

by (*auto intro: execn-to-execn-normalize execn-normalize-to-execn*)

**lemma** *exec-sequence-app*:

**assumes** *exec-xs*:  $\Gamma \vdash \langle \text{sequence Seq } xs, \text{Normal } s \rangle \Rightarrow s'$

**assumes** *exec-ys*:  $\Gamma \vdash \langle \text{sequence Seq } ys, s' \rangle \Rightarrow t$

**shows**  $\Gamma \vdash \langle \text{sequence Seq } (xs @ ys), \text{Normal } s \rangle \Rightarrow t$

**proof** –

**from** *exec-to-execn* [*OF exec-xs*]

**obtain** *n* **where**

*execn-xs*:  $\Gamma \vdash \langle \text{sequence Seq } xs, \text{Normal } s \rangle = n \Rightarrow s'..$

**from** *exec-to-execn* [*OF exec-ys*]

**obtain** *m* **where**

*execn-ys*:  $\Gamma \vdash \langle \text{sequence Seq } ys, s' \rangle = m \Rightarrow t..$

**with** *execn-xs* **obtain**

$\Gamma \vdash \langle \text{sequence Seq } xs, \text{Normal } s \rangle = \max n m \Rightarrow s'$

$\Gamma \vdash \langle \text{sequence Seq } ys, s' \rangle = \max n m \Rightarrow t$

by (*auto intro: execn-mono max.cobounded1 max.cobounded2*)

**from** *execn-sequence-app* [*OF this*]

**have**  $\Gamma \vdash \langle \text{sequence Seq } (xs @ ys), \text{Normal } s \rangle = \max n m \Rightarrow t .$

**thus** *?thesis*

by (*rule execn-to-exec*)

qed

**lemma** *exec-sequence-appD*:

**assumes** *exec-xs-ys*:  $\Gamma \vdash \langle \text{sequence Seq } (xs @ ys), \text{Normal } s \rangle \Rightarrow t$

**shows**  $\exists s'. \Gamma \vdash \langle \text{sequence Seq } xs, \text{Normal } s \rangle \Rightarrow s' \wedge \Gamma \vdash \langle \text{sequence Seq } ys, s' \rangle \Rightarrow t$

**proof** –

**from** *exec-to-execn* [*OF exec-xs-ys*]

**obtain** *n* **where**  $\Gamma \vdash \langle \text{sequence Seq } (xs @ ys), \text{Normal } s \rangle = n \Rightarrow t..$

**thus** *?thesis*

by (*cases rule: execn-sequence-appE*) (*auto intro: execn-to-exec*)

qed

**lemma** *exec-sequence-appE* [*consumes 1*]:

$\llbracket \Gamma \vdash \langle \text{sequence Seq } (xs @ ys), \text{Normal } s \rangle \Rightarrow t;$

$\bigwedge s'. \llbracket \Gamma \vdash \langle \text{sequence Seq } xs, \text{Normal } s \rangle \Rightarrow s'; \Gamma \vdash \langle \text{sequence Seq } ys, s' \rangle \Rightarrow t \rrbracket \implies P$

$\rrbracket \implies P$

by (*auto dest: exec-sequence-appD*)

**lemma** *exec-to-exec-sequence-flatten*:

**assumes** *exec*:  $\Gamma \vdash \langle c, s \rangle \Rightarrow t$

**shows**  $\Gamma \vdash \langle \text{sequence Seq } (\text{flatten } c), s \rangle \Rightarrow t$

**proof** –

**from** *exec-to-execn* [*OF exec*]

**obtain** *n* **where**  $\Gamma \vdash \langle c, s \rangle = n \Rightarrow t..$

**from** *execn-to-execn-sequence-flatten* [OF *this*]  
**show** *?thesis*  
**by** (rule *execn-to-exec*)  
**qed**

**lemma** *exec-sequence-flatten-to-exec*:  
**assumes** *exec-seq*:  $\Gamma \vdash \langle \text{sequence Seq (flatten } c), s \rangle \Rightarrow t$   
**shows**  $\Gamma \vdash \langle c, s \rangle \Rightarrow t$   
**proof** –  
**from** *exec-to-execn* [OF *exec-seq*]  
**obtain** *n* **where**  $\Gamma \vdash \langle \text{sequence Seq (flatten } c), s \rangle = n \Rightarrow t..$   
**from** *execn-sequence-flatten-to-execn* [OF *this*]  
**show** *?thesis*  
**by** (rule *execn-to-exec*)  
**qed**

**lemma** *exec-to-exec-normalize*:  
**assumes** *exec*:  $\Gamma \vdash \langle c, s \rangle \Rightarrow t$   
**shows**  $\Gamma \vdash \langle \text{normalize } c, s \rangle \Rightarrow t$   
**proof** –  
**from** *exec-to-execn* [OF *exec*] **obtain** *n* **where**  $\Gamma \vdash \langle c, s \rangle = n \Rightarrow t..$   
**hence**  $\Gamma \vdash \langle \text{normalize } c, s \rangle = n \Rightarrow t$   
**by** (rule *execn-to-execn-normalize*)  
**thus** *?thesis*  
**by** (rule *execn-to-exec*)  
**qed**

**lemma** *exec-normalize-to-exec*:  
**assumes** *exec*:  $\Gamma \vdash \langle \text{normalize } c, s \rangle \Rightarrow t$   
**shows**  $\Gamma \vdash \langle c, s \rangle \Rightarrow t$   
**proof** –  
**from** *exec-to-execn* [OF *exec*] **obtain** *n* **where**  $\Gamma \vdash \langle \text{normalize } c, s \rangle = n \Rightarrow t..$   
**hence**  $\Gamma \vdash \langle c, s \rangle = n \Rightarrow t$   
**by** (rule *execn-normalize-to-execn*)  
**thus** *?thesis*  
**by** (rule *execn-to-exec*)  
**qed**

**lemma** *exec-normalize-iff-exec*:  
 $\Gamma \vdash \langle \text{normalize } c, s \rangle \Rightarrow t = \Gamma \vdash \langle c, s \rangle \Rightarrow t$   
**by** (auto intro: *exec-to-exec-normalize exec-normalize-to-exec*)

### 3.4 Lemmas about $c_1 \subseteq_g c_2$

**lemma** *execn-to-execn-subseteq-guards*:  $\bigwedge c \ s \ t \ n. \llbracket c \subseteq_g c'; \Gamma \vdash \langle c, s \rangle = n \Rightarrow t \rrbracket$   
 $\implies \exists t'. \Gamma \vdash \langle c', s \rangle = n \Rightarrow t' \wedge$   
 $(\text{isFault } t \longrightarrow \text{isFault } t') \wedge (\neg \text{isFault } t' \longrightarrow t'=t)$   
**proof** (*induct c'*)  
**case** *Skip* **thus** *?case*

```

    by (fastforce dest: subseteq-guardsD elim: execn-elim-cases)
next
  case Basic thus ?case
    by (fastforce dest: subseteq-guardsD elim: execn-elim-cases)
next
  case Spec thus ?case
    by (fastforce dest: subseteq-guardsD elim: execn-elim-cases)
next
  case (Seq c1' c2')
  have  $c \subseteq_g \text{Seq } c1' c2'$  by fact
  from subseteq-guards-Seq [OF this]
  obtain c1 c2 where
    c:  $c = \text{Seq } c1 c2$  and
    c1-c1':  $c1 \subseteq_g c1'$  and
    c2-c2':  $c2 \subseteq_g c2'$ 
  by blast
  have exec:  $\Gamma \vdash \langle c, s \rangle = n \Rightarrow t$  by fact
  with c obtain w where
    exec-c1:  $\Gamma \vdash \langle c1, s \rangle = n \Rightarrow w$  and
    exec-c2:  $\Gamma \vdash \langle c2, w \rangle = n \Rightarrow t$ 
  by (auto elim: execn-elim-cases)
  from exec-c1 Seq.hyps c1-c1'
  obtain w' where
    exec-c1':  $\Gamma \vdash \langle c1', s \rangle = n \Rightarrow w'$  and
    w-Fault:  $\text{isFault } w \longrightarrow \text{isFault } w'$  and
    w'-noFault:  $\neg \text{isFault } w' \longrightarrow w' = w$ 
  by blast
  show ?case
  proof (cases s)
  case (Fault f)
  with exec have t=Fault f
    by (auto dest: execn-Fault-end)
  with Fault show ?thesis
    by auto
  next
  case Stuck
  with exec have t=Stuck
    by (auto dest: execn-Stuck-end)
  with Stuck show ?thesis
    by auto
  next
  case (Abrupt s')
  with exec have t=Abrupt s'
    by (auto dest: execn-Abrupt-end)
  with Abrupt show ?thesis
    by auto
  next
  case (Normal s')
  show ?thesis

```

```

proof (cases isFault w)
  case True
    then obtain f where w': w=Fault f..
    moreover with exec-c2
    have t: t=Fault f
      by (auto dest: execn-Fault-end)
    ultimately show ?thesis
      using Normal w-Fault exec-c1'
      by (fastforce intro: execn.intros elim: isFaultE)
  next
    case False
    note noFault-w = this
    show ?thesis
    proof (cases isFault w')
      case True
        then obtain f' where w': w'=Fault f'..
        with Normal exec-c1'
        have exec:  $\Gamma \vdash \langle \text{Seq } c1' \ c2', s \rangle =n \Rightarrow \text{Fault } f'$ 
          by (auto intro: execn.intros)
        then show ?thesis
          by auto
      next
        case False
        with w'-noFault have w': w'=w by simp
        from Seq.hyps exec-c2 c2-c2'
        obtain t' where
           $\Gamma \vdash \langle c2', w \rangle =n \Rightarrow t'$  and
          isFault t  $\longrightarrow$  isFault t' and
           $\neg \text{isFault } t' \longrightarrow t'=t$ 
          by blast
        with Normal exec-c1' w'
        show ?thesis
          by (fastforce intro: execn.intros)
    qed
  qed
qed
next
  case (Cond b c1' c2')
  have exec:  $\Gamma \vdash \langle c, s \rangle =n \Rightarrow t$  by fact
  have  $c \subseteq_g \text{Cond } b \ c1' \ c2'$  by fact
  from subseteq-guards-Cond [OF this]
  obtain c1 c2 where
    c:  $c = \text{Cond } b \ c1 \ c2$  and
    c1-c1':  $c1 \subseteq_g c1'$  and
    c2-c2':  $c2 \subseteq_g c2'$ 
    by blast
  show ?case
  proof (cases s)
    case (Fault f)

```

```

with exec have  $t = \text{Fault } f$ 
  by (auto dest: execn-Fault-end)
with Fault show ?thesis
  by auto
next
case Stuck
with exec have  $t = \text{Stuck}$ 
  by (auto dest: execn-Stuck-end)
with Stuck show ?thesis
  by auto
next
case (Abrupt s')
with exec have  $t = \text{Abrupt } s'$ 
  by (auto dest: execn-Abrupt-end)
with Abrupt show ?thesis
  by auto
next
case (Normal s')
from exec [simplified c Normal]
show ?thesis
proof (cases)
  assume s'-in-b:  $s' \in b$ 
  assume  $\Gamma \vdash \langle c1, \text{Normal } s' \rangle = n \Rightarrow t$ 
  with c1-c1' Normal Cond.hyps obtain  $t'$  where
     $\Gamma \vdash \langle c1', \text{Normal } s' \rangle = n \Rightarrow t'$ 
     $\text{isFault } t \longrightarrow \text{isFault } t'$ 
     $\neg \text{isFault } t' \longrightarrow t' = t$ 
  by blast
  with s'-in-b Normal show ?thesis
  by (fastforce intro: execn.intros)
next
  assume s'-notin-b:  $s' \notin b$ 
  assume  $\Gamma \vdash \langle c2, \text{Normal } s' \rangle = n \Rightarrow t$ 
  with c2-c2' Normal Cond.hyps obtain  $t'$  where
     $\Gamma \vdash \langle c2', \text{Normal } s' \rangle = n \Rightarrow t'$ 
     $\text{isFault } t \longrightarrow \text{isFault } t'$ 
     $\neg \text{isFault } t' \longrightarrow t' = t$ 
  by blast
  with s'-notin-b Normal show ?thesis
  by (fastforce intro: execn.intros)
qed
qed
next
case (While b c')
have exec:  $\Gamma \vdash \langle c, s \rangle = n \Rightarrow t$  by fact
have  $c \subseteq_g \text{While } b \ c'$  by fact
from subseq-guards-While [OF this]
obtain  $c''$  where
   $c: c = \text{While } b \ c''$  and

```

```

  c''-c': c''  $\subseteq_g$  c'
  by blast
{
  fix c r w
  assume exec:  $\Gamma \vdash \langle c, r \rangle = n \Rightarrow w$ 
  assume c: c = While b c''
  have  $\exists w'. \Gamma \vdash \langle \text{While } b \ c', r \rangle = n \Rightarrow w' \wedge$ 
    (isFault w  $\longrightarrow$  isFault w')  $\wedge$  ( $\neg$  isFault w'  $\longrightarrow$  w'=w)
  using exec c
  proof (induct)
    case (WhileTrue r b' ca n u w)
    have eqs: While b' ca = While b c'' by fact
    from WhileTrue have r-in-b: r  $\in$  b by simp
    from WhileTrue have exec-c'':  $\Gamma \vdash \langle c'', \text{Normal } r \rangle = n \Rightarrow u$  by simp
    from While.hyps [OF c''-c' exec-c''] obtain u' where
      exec-c':  $\Gamma \vdash \langle c', \text{Normal } r \rangle = n \Rightarrow u'$  and
      u-Fault: isFault u  $\longrightarrow$  isFault u' and
      u'-noFault:  $\neg$  isFault u'  $\longrightarrow$  u' = u
    by blast
    from WhileTrue obtain w' where
      exec-w:  $\Gamma \vdash \langle \text{While } b \ c', u \rangle = n \Rightarrow w'$  and
      w-Fault: isFault w  $\longrightarrow$  isFault w' and
      w'-noFault:  $\neg$  isFault w'  $\longrightarrow$  w' = w
    by blast
    show ?case
    proof (cases isFault u')
      case True
      with exec-c' r-in-b
      show ?thesis
      by (fastforce intro: execn.intros elim: isFaultE)
    next
      case False
      with exec-c' r-in-b u'-noFault exec-w w-Fault w'-noFault
      show ?thesis
      by (fastforce intro: execn.intros)
    qed
  next
  case WhileFalse thus ?case by (fastforce intro: execn.intros)
  qed auto
}
from this [OF exec c]
show ?case .
next
case Call thus ?case
  by (fastforce dest: subseteq-guardsD elim: execn-elim-cases)
next
case (DynCom C')
  have exec:  $\Gamma \vdash \langle c, s \rangle = n \Rightarrow t$  by fact
  have c  $\subseteq_g$  DynCom C' by fact

```

```

from subsetq-guards-DynCom [OF this] obtain  $C$  where
   $c = \text{DynCom } C$  and
   $C-C': \forall s. C s \subseteq_g C' s$ 
  by blast
show ?case
proof (cases s)
  case (Fault f)
    with exec have  $t = \text{Fault } f$ 
      by (auto dest: execn-Fault-end)
    with Fault show ?thesis
      by auto
  next
    case Stuck
      with exec have  $t = \text{Stuck}$ 
        by (auto dest: execn-Stuck-end)
      with Stuck show ?thesis
        by auto
  next
    case (Abrupt s')
      with exec have  $t = \text{Abrupt } s'$ 
        by (auto dest: execn-Abrupt-end)
      with Abrupt show ?thesis
        by auto
  next
    case (Normal s')
      from exec [simplified c Normal]
      have  $\Gamma \vdash \langle C s', \text{Normal } s' \rangle = n \Rightarrow t$ 
        by cases
      from DynCom.hyps  $C-C'$  [rule-format] this obtain  $t'$  where
         $\Gamma \vdash \langle C' s', \text{Normal } s' \rangle = n \Rightarrow t'$ 
         $\text{isFault } t \longrightarrow \text{isFault } t'$ 
         $\neg \text{isFault } t' \longrightarrow t' = t$ 
        by blast
      with Normal show ?thesis
        by (fastforce intro: execn.intros)
  qed
next
  case (Guard f' g' c')
  have exec:  $\Gamma \vdash \langle c, s \rangle = n \Rightarrow t$  by fact
  have  $c \subseteq_g \text{Guard } f' g' c'$  by fact
  hence subset-cases:  $(c \subseteq_g c') \vee (\exists c''. c = \text{Guard } f' g' c'' \wedge (c'' \subseteq_g c'))$ 
    by (rule subsetq-guards-Guard)
  show ?case
  proof (cases s)
    case (Fault f)
      with exec have  $t = \text{Fault } f$ 
        by (auto dest: execn-Fault-end)
      with Fault show ?thesis
        by auto

```



```

next
  case Stuck
  with exec have  $t = \text{Stuck}$ 
    by (auto dest: execn-Stuck-end)
  with Stuck show ?thesis
    by auto
next
  case (Abrupt  $s'$ )
  with exec have  $t = \text{Abrupt } s'$ 
    by (auto dest: execn-Abrupt-end)
  with Abrupt show ?thesis
    by auto
next
  case (Normal  $s'$ )
  from subset-cases show ?thesis
  proof
    assume  $c - c': c \subseteq_g c'$ 
    from Guard.hyps [OF this exec] Normal obtain  $t'$  where
      exec-c':  $\Gamma \vdash \langle c', \text{Normal } s' \rangle = n \Rightarrow t'$  and
      t-Fault:  $\text{isFault } t \longrightarrow \text{isFault } t'$  and
      t-noFault:  $\neg \text{isFault } t' \longrightarrow t' = t$ 
    by blast
    with Normal
    show ?thesis
      by (cases  $s' \in g'$ ) (fastforce intro: execn.intros) +
next
  assume  $\exists c''. c = \text{Guard } f' g' c'' \wedge (c'' \subseteq_g c')$ 
  then obtain  $c''$  where
     $c: c = \text{Guard } f' g' c''$  and
     $c'' - c': c'' \subseteq_g c'$ 
  by blast
  from  $c$  exec Normal
  have exec-Guard':  $\Gamma \vdash \langle \text{Guard } f' g' c'', \text{Normal } s' \rangle = n \Rightarrow t$ 
  by simp
  thus ?thesis
  proof (cases)
    assume  $s' - \text{in-}g': s' \in g'$ 
    assume exec-c'':  $\Gamma \vdash \langle c'', \text{Normal } s' \rangle = n \Rightarrow t$ 
    from Guard.hyps [OF  $c'' - c'$  exec-c''] obtain  $t'$  where
      exec-c':  $\Gamma \vdash \langle c', \text{Normal } s' \rangle = n \Rightarrow t'$  and
      t-Fault:  $\text{isFault } t \longrightarrow \text{isFault } t'$  and
      t-noFault:  $\neg \text{isFault } t' \longrightarrow t' = t$ 
    by blast
    with Normal  $s' - \text{in-}g'$ 
    show ?thesis
      by (fastforce intro: execn.intros)
  next
  assume  $s' \notin g'$   $t = \text{Fault } f'$ 
  with Normal show ?thesis

```

```

      by (fastforce intro: execn.intros)
    qed
  qed
next
case Throw thus ?case
  by (fastforce dest: subseteq-guardsD intro: execn.intros
      elim: execn-elim-cases)
next
case (Catch  $c1' c2'$ )
have  $c \subseteq_g \text{Catch } c1' c2'$  by fact
from subseteq-guards-Catch [OF this]
obtain  $c1 c2$  where
   $c = \text{Catch } c1 c2$  and
   $c1-c1'$ :  $c1 \subseteq_g c1'$  and
   $c2-c2'$ :  $c2 \subseteq_g c2'$ 
by blast
have  $\text{exec}: \Gamma \vdash \langle c, s \rangle =n \Rightarrow t$  by fact
show ?thesis
proof (cases  $s$ )
  case (Fault  $f$ )
  with  $\text{exec}$  have  $t = \text{Fault } f$ 
  by (auto dest: execn-Fault-end)
  with Fault show ?thesis
  by auto
next
case Stuck
with  $\text{exec}$  have  $t = \text{Stuck}$ 
by (auto dest: execn-Stuck-end)
with Stuck show ?thesis
by auto
next
case (Abrupt  $s'$ )
with  $\text{exec}$  have  $t = \text{Abrupt } s'$ 
by (auto dest: execn-Abrupt-end)
with Abrupt show ?thesis
by auto
next
case (Normal  $s'$ )
from  $\text{exec}$  [simplified  $c$  Normal]
show ?thesis
proof (cases)
  fix  $w$ 
  assume  $\text{exec-c1}: \Gamma \vdash \langle c1, \text{Normal } s' \rangle =n \Rightarrow \text{Abrupt } w$ 
  assume  $\text{exec-c2}: \Gamma \vdash \langle c2, \text{Normal } w \rangle =n \Rightarrow t$ 
  from Normal  $\text{exec-c1}$   $c1-c1'$  Catch.hyps obtain  $w'$  where
     $\text{exec-c1}' : \Gamma \vdash \langle c1', \text{Normal } s' \rangle =n \Rightarrow w'$  and
     $w'\text{-noFault}: \neg \text{isFault } w' \longrightarrow w' = \text{Abrupt } w$ 
  by blast

```

```

show ?thesis
proof (cases isFault w')
  case True
  with exec-c1' Normal show ?thesis
  by (fastforce intro: execn.intros elim: isFaultE)
next
  case False
  with w'-noFault have w': w'=Abrupt w by simp
  from Normal exec-c2 c2-c2' Catch.hyps obtain t' where
     $\Gamma \vdash \langle c2', Normal w \rangle = n \Rightarrow t'$ 
    isFault t  $\longrightarrow$  isFault t'
     $\neg$  isFault t'  $\longrightarrow$  t' = t
  by blast
  with exec-c1' w' Normal
  show ?thesis
  by (fastforce intro: execn.intros )
qed
next
  assume exec-c1:  $\Gamma \vdash \langle c1, Normal s \rangle = n \Rightarrow t$ 
  assume t:  $\neg$  isAbr t
  from Normal exec-c1 c1-c1' Catch.hyps obtain t' where
    exec-c1':  $\Gamma \vdash \langle c1', Normal s \rangle = n \Rightarrow t'$  and
    t-Fault: isFault t  $\longrightarrow$  isFault t' and
    t'-noFault:  $\neg$  isFault t'  $\longrightarrow$  t' = t
  by blast
  show ?thesis
  proof (cases isFault t')
    case True
    with exec-c1' Normal show ?thesis
    by (fastforce intro: execn.intros elim: isFaultE)
  next
    case False
    with exec-c1' Normal t-Fault t'-noFault t
    show ?thesis
    by (fastforce intro: execn.intros)
  qed
qed
qed
qed

```

**lemma** *exec-to-exec-subseteq-guards*:

**assumes**  $c-c'$ :  $c \subseteq_g c'$

**assumes** *exec*:  $\Gamma \vdash \langle c, s \rangle \Rightarrow t$

**shows**  $\exists t'. \Gamma \vdash \langle c', s \rangle \Rightarrow t' \wedge$

$(isFault t \longrightarrow isFault t') \wedge (\neg isFault t' \longrightarrow t'=t)$

**proof** –

**from** *exec-to-execn* [OF *exec*] **obtain**  $n$  **where**

$\Gamma \vdash \langle c, s \rangle = n \Rightarrow t$  ..

**from** *execn-to-execn-subseteq-guards* [OF  $c-c'$  *this*]

```

show ?thesis
  by (blast intro: execn-to-exec)
qed

```

### 3.5 Lemmas about *merge-guards*

```

theorem execn-to-execn-merge-guards:
  assumes exec-c:  $\Gamma \vdash \langle c, s \rangle = n \Rightarrow t$ 
  shows  $\Gamma \vdash \langle \text{merge-guards } c, s \rangle = n \Rightarrow t$ 
using exec-c
proof (induct)
  case (Guard s g c n t f)
  have s-in-g:  $s \in g$  by fact
  have exec-merge-c:  $\Gamma \vdash \langle \text{merge-guards } c, \text{Normal } s \rangle = n \Rightarrow t$  by fact
  show ?case
  proof (cases  $\exists f' g' c'. \text{merge-guards } c = \text{Guard } f' g' c'$ )
    case False
    with exec-merge-c s-in-g
    show ?thesis
    by (cases merge-guards c) (auto intro: execn.intros simp add: Let-def)
  next
  case True
  then obtain f' g' c' where
    merge-guards-c:  $\text{merge-guards } c = \text{Guard } f' g' c'$ 
    by iprover
  show ?thesis
  proof (cases f=f')
    case False
    from exec-merge-c s-in-g merge-guards-c False show ?thesis
    by (auto intro: execn.intros simp add: Let-def)
  next
  case True
  from exec-merge-c s-in-g merge-guards-c True show ?thesis
  by (fastforce intro: execn.intros elim: execn.cases)
  qed
qed
next
  case (GuardFault s g f c n)
  have s-notin-g:  $s \notin g$  by fact
  show ?case
  proof (cases  $\exists f' g' c'. \text{merge-guards } c = \text{Guard } f' g' c'$ )
    case False
    with s-notin-g
    show ?thesis
    by (cases merge-guards c) (auto intro: execn.intros simp add: Let-def)
  next
  case True
  then obtain f' g' c' where
    merge-guards-c:  $\text{merge-guards } c = \text{Guard } f' g' c'$ 

```

```

    by iprover
  show ?thesis
proof (cases f=f')
  case False
  from s-notin-g merge-guards-c False show ?thesis
    by (auto intro: execn.intros simp add: Let-def)
next
  case True
  from s-notin-g merge-guards-c True show ?thesis
    by (fastforce intro: execn.intros)
qed
qed (fastforce intro: execn.intros)+

```

lemma *execn-merge-guards-to-execn-Normal*:

```

 $\bigwedge s n t. \Gamma \vdash \langle \text{merge-guards } c, \text{Normal } s \rangle = n \Rightarrow t \implies \Gamma \vdash \langle c, \text{Normal } s \rangle = n \Rightarrow t$ 
proof (induct c)
  case Skip thus ?case by auto
next
  case Basic thus ?case by auto
next
  case Spec thus ?case by auto
next
  case (Seq c1 c2)
  have  $\Gamma \vdash \langle \text{merge-guards } (\text{Seq } c1 \ c2), \text{Normal } s \rangle = n \Rightarrow t$  by fact
  hence exec-merge:  $\Gamma \vdash \langle \text{Seq } (\text{merge-guards } c1) \ (\text{merge-guards } c2), \text{Normal } s \rangle = n \Rightarrow t$ 
  by simp
  then obtain s' where
    exec-merge-c1:  $\Gamma \vdash \langle \text{merge-guards } c1, \text{Normal } s \rangle = n \Rightarrow s'$  and
    exec-merge-c2:  $\Gamma \vdash \langle \text{merge-guards } c2, s' \rangle = n \Rightarrow t$ 
  by cases
  from exec-merge-c1
  have exec-c1:  $\Gamma \vdash \langle c1, \text{Normal } s \rangle = n \Rightarrow s'$ 
    by (rule Seq.hyps)
  show ?case
  proof (cases s')
    case (Normal s'')
    with exec-merge-c2
    have  $\Gamma \vdash \langle c2, s' \rangle = n \Rightarrow t$ 
      by (auto intro: Seq.hyps)
    with exec-c1 show ?thesis
      by (auto intro: execn.intros)
  next
    case (Abrupt s'')
    with exec-merge-c2 have  $t = \text{Abrupt } s''$ 
      by (auto dest: execn-Abrupt-end)
    with exec-c1 Abrupt
    show ?thesis

```

```

    by (auto intro: execn.intros)
next
  case (Fault f)
  with exec-merge-c2 have t=Fault f
  by (auto dest: execn-Fault-end)
  with exec-c1 Fault
  show ?thesis
  by (auto intro: execn.intros)
next
  case Stuck
  with exec-merge-c2 have t=Stuck
  by (auto dest: execn-Stuck-end)
  with exec-c1 Stuck
  show ?thesis
  by (auto intro: execn.intros)
qed
next
  case Cond thus ?case
  by (fastforce intro: execn.intros elim: execn-Normal-elim-cases)
next
  case (While b c)
  {
    fix c' r w
    assume exec-c':  $\Gamma \vdash \langle c', r \rangle = n \Rightarrow w$ 
    assume c':  $c' = \text{While } b \text{ (merge-guards } c)$ 
    have  $\Gamma \vdash \langle \text{While } b \text{ } c, r \rangle = n \Rightarrow w$ 
      using exec-c' c'
    proof (induct)
      case (WhileTrue r b' c'' n u w)
      have eqs:  $\text{While } b' \text{ } c'' = \text{While } b \text{ (merge-guards } c)$  by fact
      from WhileTrue
      have r-in-b:  $r \in b$ 
        by simp
      from WhileTrue While.hyps have exec-c:  $\Gamma \vdash \langle c, \text{Normal } r \rangle = n \Rightarrow u$ 
        by simp
      from WhileTrue have exec-w:  $\Gamma \vdash \langle \text{While } b \text{ } c, u \rangle = n \Rightarrow w$ 
        by simp
      from r-in-b exec-c exec-w
      show ?case
        by (rule execn.WhileTrue)
    next
      case WhileFalse thus ?case by (auto intro: execn.WhileFalse)
    qed auto
  }
  with While.premis show ?case
  by (auto)
next
  case Call thus ?case by simp
next

```

```

case DynCom thus ?case
  by (fastforce intro: execn.intros elim: execn-Normal-elim-cases)
next
case (Guard f g c)
have exec-merge:  $\Gamma \vdash \langle \text{merge-guards } (Guard\ f\ g\ c), Normal\ s \rangle =n \Rightarrow t$  by fact
show ?case
proof (cases  $s \in g$ )
  case False
  with exec-merge have  $t = Fault\ f$ 
    by (auto split: com.splits if-split-asm elim: execn-Normal-elim-cases
      simp add: Let-def is-Guard-def)
  with False show ?thesis
    by (auto intro: execn.intros)
next
case True
note  $s\text{-in-}g = \text{this}$ 
show ?thesis
proof (cases  $\exists f'\ g'\ c'. \text{merge-guards } c = Guard\ f'\ g'\ c'$ )
  case False
  then
  have  $\text{merge-guards } (Guard\ f\ g\ c) = Guard\ f\ g\ (\text{merge-guards } c)$ 
    by (cases  $\text{merge-guards } c$ ) (auto simp add: Let-def)
  with exec-merge  $s\text{-in-}g$ 
  obtain  $\Gamma \vdash \langle \text{merge-guards } c, Normal\ s \rangle =n \Rightarrow t$ 
    by (auto elim: execn-Normal-elim-cases)
  from Guard.hyps [OF this]  $s\text{-in-}g$ 
  show ?thesis
    by (auto intro: execn.intros)
next
case True
then obtain  $f'\ g'\ c'$  where
   $\text{merge-guards-}c: \text{merge-guards } c = Guard\ f'\ g'\ c'$ 
  by iprover
show ?thesis
proof (cases  $f = f'$ )
  case False
  with  $\text{merge-guards-}c$ 
  have  $\text{merge-guards } (Guard\ f\ g\ c) = Guard\ f\ g\ (\text{merge-guards } c)$ 
    by (simp add: Let-def)
  with exec-merge  $s\text{-in-}g$ 
  obtain  $\Gamma \vdash \langle \text{merge-guards } c, Normal\ s \rangle =n \Rightarrow t$ 
    by (auto elim: execn-Normal-elim-cases)
  from Guard.hyps [OF this]  $s\text{-in-}g$ 
  show ?thesis
    by (auto intro: execn.intros)
next
case True
note  $f\text{-eq-}f' = \text{this}$ 
with  $\text{merge-guards-}c$  have

```

```

merge-guards-Guard: merge-guards (Guard f g c) = Guard f (g  $\cap$  g') c'
by simp
show ?thesis
proof (cases s  $\in$  g')
case True
with exec-merge merge-guards-Guard merge-guards-c s-in-g
have  $\Gamma \vdash \langle \text{merge-guards } c, \text{Normal } s \rangle = n \Rightarrow t$ 
by (auto intro: execn.intros elim: execn-Normal-elim-cases)
with Guard.hyps [OF this] s-in-g
show ?thesis
by (auto intro: execn.intros)
next
case False
with exec-merge merge-guards-Guard
have t=Fault f
by (auto elim: execn-Normal-elim-cases)
with merge-guards-c f-eq-f' False
have  $\Gamma \vdash \langle \text{merge-guards } c, \text{Normal } s \rangle = n \Rightarrow t$ 
by (auto intro: execn.intros)
from Guard.hyps [OF this] s-in-g
show ?thesis
by (auto intro: execn.intros)
qed
qed
qed
qed
next
case Throw thus ?case by simp
next
case (Catch c1 c2)
have  $\Gamma \vdash \langle \text{merge-guards } (\text{Catch } c1 c2), \text{Normal } s \rangle = n \Rightarrow t$  by fact
hence  $\Gamma \vdash \langle \text{Catch } (\text{merge-guards } c1) (\text{merge-guards } c2), \text{Normal } s \rangle = n \Rightarrow t$  by simp
thus ?case
by cases (auto intro: execn.intros Catch.hyps)
qed

```

**theorem** *execn-merge-guards-to-execn:*

```

 $\Gamma \vdash \langle \text{merge-guards } c, s \rangle = n \Rightarrow t \implies \Gamma \vdash \langle c, s \rangle = n \Rightarrow t$ 
apply (cases s)
apply (fastforce intro: execn-merge-guards-to-execn-Normal)
apply (fastforce dest: execn-Abrupt-end)
apply (fastforce dest: execn-Fault-end)
apply (fastforce dest: execn-Stuck-end)
done

```

**corollary** *execn-iff-execn-merge-guards:*

```

 $\Gamma \vdash \langle c, s \rangle = n \Rightarrow t = \Gamma \vdash \langle \text{merge-guards } c, s \rangle = n \Rightarrow t$ 
by (blast intro: execn-merge-guards-to-execn execn-to-execn-merge-guards)

```



**theorem** *exec-iff-exec-merge-guards*:  
 $\Gamma \vdash \langle c, s \rangle \Rightarrow t = \Gamma \vdash \langle \text{merge-guards } c, s \rangle \Rightarrow t$   
**by** (*blast dest: exec-to-execn intro: execn-to-exec*  
*intro: execn-to-execn-merge-guards*  
*execn-merge-guards-to-execn*)

**corollary** *exec-to-exec-merge-guards*:  
 $\Gamma \vdash \langle c, s \rangle \Rightarrow t \Longrightarrow \Gamma \vdash \langle \text{merge-guards } c, s \rangle \Rightarrow t$   
**by** (*rule iffD1 [OF exec-iff-exec-merge-guards]*)

**corollary** *exec-merge-guards-to-exec*:  
 $\Gamma \vdash \langle \text{merge-guards } c, s \rangle \Rightarrow t \Longrightarrow \Gamma \vdash \langle c, s \rangle \Rightarrow t$   
**by** (*rule iffD2 [OF exec-iff-exec-merge-guards]*)

### 3.6 Lemmas about *mark-guards*

**lemma** *execn-to-execn-mark-guards*:  
**assumes** *exec-c*:  $\Gamma \vdash \langle c, s \rangle =n \Rightarrow t$   
**assumes** *t-not-Fault*:  $\neg \text{isFault } t$   
**shows**  $\Gamma \vdash \langle \text{mark-guards } f \ c, s \rangle =n \Rightarrow t$   
**using** *exec-c t-not-Fault [simplified not-isFault-iff]*  
**by** (*induct*) (*auto intro: execn.intros dest: noFaultn-startD'*)

**lemma** *execn-to-execn-mark-guards-Fault*:  
**assumes** *exec-c*:  $\Gamma \vdash \langle c, s \rangle =n \Rightarrow t$   
**shows**  $\bigwedge f. \llbracket t = \text{Fault } f \rrbracket \Longrightarrow \exists f'. \Gamma \vdash \langle \text{mark-guards } x \ c, s \rangle =n \Rightarrow \text{Fault } f'$   
**using** *exec-c*  
**proof** (*induct*)  
  *case Skip thus ?case by auto*  
**next**  
  *case Guard thus ?case by (fastforce intro: execn.intros)*  
**next**  
  *case GuardFault thus ?case by (fastforce intro: execn.intros)*  
**next**  
  *case FaultProp thus ?case by auto*  
**next**  
  *case Basic thus ?case by auto*  
**next**  
  *case Spec thus ?case by auto*  
**next**  
  *case SpecStuck thus ?case by auto*  
**next**  
  *case (Seq c1 s n w c2 t)*  
  **have** *exec-c1*:  $\Gamma \vdash \langle c1, \text{Normal } s \rangle =n \Rightarrow w$  **by fact**  
  **have** *exec-c2*:  $\Gamma \vdash \langle c2, w \rangle =n \Rightarrow t$  **by fact**  
  **have** *t*:  $t = \text{Fault } f$  **by fact**  
  **show** *?case*  
  **proof** (*cases w*)  
    *case (Fault f')*

```

with exec-c2 t have  $f'=f$ 
  by (auto dest: execn-Fault-end)
with Fault Seq.hyps obtain  $f''$  where
   $\Gamma \langle \text{mark-guards } x \ c1, \text{Normal } s \rangle =n \Rightarrow \text{Fault } f''$ 
  by auto
moreover have  $\Gamma \langle \text{mark-guards } x \ c2, \text{Fault } f' \rangle =n \Rightarrow \text{Fault } f''$ 
  by auto
ultimately show ?thesis
  by (auto intro: execn.intros)
next
case (Normal s')
with execn-to-execn-mark-guards [OF exec-c1]
have exec-mark-c1:  $\Gamma \langle \text{mark-guards } x \ c1, \text{Normal } s \rangle =n \Rightarrow w$ 
  by simp
with Seq.hyps t obtain  $f'$  where
   $\Gamma \langle \text{mark-guards } x \ c2, w \rangle =n \Rightarrow \text{Fault } f'$ 
  by blast
with exec-mark-c1 show ?thesis
  by (auto intro: execn.intros)
next
case (Abrupt s')
with execn-to-execn-mark-guards [OF exec-c1]
have exec-mark-c1:  $\Gamma \langle \text{mark-guards } x \ c1, \text{Normal } s \rangle =n \Rightarrow w$ 
  by simp
with Seq.hyps t obtain  $f'$  where
   $\Gamma \langle \text{mark-guards } x \ c2, w \rangle =n \Rightarrow \text{Fault } f'$ 
  by (auto intro: execn.intros)
with exec-mark-c1 show ?thesis
  by (auto intro: execn.intros)
next
case Stuck
with exec-c2 have  $t=\text{Stuck}$ 
  by (auto dest: execn-Stuck-end)
with  $t$  show ?thesis by simp
qed
next
case CondTrue thus ?case by (fastforce intro: execn.intros)
next
case CondFalse thus ?case by (fastforce intro: execn.intros)
next
case (WhileTrue s b c n w t)
have exec-c:  $\Gamma \langle c, \text{Normal } s \rangle =n \Rightarrow w$  by fact
have exec-w:  $\Gamma \langle \text{While } b \ c, w \rangle =n \Rightarrow t$  by fact
have  $t = \text{Fault } f$  by fact
have s-in-b:  $s \in b$  by fact
show ?case
proof (cases w)
  case (Fault f')
    with exec-w t have  $f'=f$ 

```

```

    by (auto dest: execn-Fault-end)
  with WhileTrue.hyps obtain  $f''$  where
     $\Gamma \langle \text{mark-guards } x \ c, \text{Normal } s \rangle = n \Rightarrow \text{Fault } f''$ 
  by auto
  moreover have  $\Gamma \langle \text{mark-guards } x \ (\text{While } b \ c), \text{Fault } f'' \rangle = n \Rightarrow \text{Fault } f''$ 
  by auto
  ultimately show ?thesis
    using s-in-b by (auto intro: execn.intros)
next
  case (Normal s')
  with execn-to-execn-mark-guards [OF exec-c]
  have exec-mark-c:  $\Gamma \langle \text{mark-guards } x \ c, \text{Normal } s \rangle = n \Rightarrow w$ 
  by simp
  with WhileTrue.hyps t obtain  $f'$  where
     $\Gamma \langle \text{mark-guards } x \ (\text{While } b \ c), w \rangle = n \Rightarrow \text{Fault } f'$ 
  by blast
  with exec-mark-c s-in-b show ?thesis
    by (auto intro: execn.intros)
next
  case (Abrupt s')
  with execn-to-execn-mark-guards [OF exec-c]
  have exec-mark-c:  $\Gamma \langle \text{mark-guards } x \ c, \text{Normal } s \rangle = n \Rightarrow w$ 
  by simp
  with WhileTrue.hyps t obtain  $f'$  where
     $\Gamma \langle \text{mark-guards } x \ (\text{While } b \ c), w \rangle = n \Rightarrow \text{Fault } f'$ 
  by (auto intro: execn.intros)
  with exec-mark-c s-in-b show ?thesis
    by (auto intro: execn.intros)
next
  case Stuck
  with exec-w have  $t = \text{Stuck}$ 
  by (auto dest: execn-Stuck-end)
  with  $t$  show ?thesis by simp
qed
next
  case WhileFalse thus ?case by (fastforce intro: execn.intros)
next
  case Call thus ?case by (fastforce intro: execn.intros)
next
  case CallUndefined thus ?case by simp
next
  case StuckProp thus ?case by simp
next
  case DynCom thus ?case by (fastforce intro: execn.intros)
next
  case Throw thus ?case by simp
next
  case AbruptProp thus ?case by simp
next

```

**case** (*CatchMatch*  $c1\ s\ n\ w\ c2\ t$ )  
**have**  $exec-c1: \Gamma \vdash \langle c1, Normal\ s \rangle =n\Rightarrow Abrupt\ w$  **by** *fact*  
**have**  $exec-c2: \Gamma \vdash \langle c2, Normal\ w \rangle =n\Rightarrow t$  **by** *fact*  
**have**  $t = Fault\ f$  **by** *fact*  
**from** *execn-to-execn-mark-guards* [*OF exec-c1*]  
**have**  $exec-mark-c1: \Gamma \vdash \langle mark-guards\ x\ c1, Normal\ s \rangle =n\Rightarrow Abrupt\ w$   
**by** *simp*  
**with** *CatchMatch.hyps*  $t$  **obtain**  $f'$  **where**  
 $\Gamma \vdash \langle mark-guards\ x\ c2, Normal\ w \rangle =n\Rightarrow Fault\ f'$   
**by** *blast*  
**with**  $exec-mark-c1$  **show** *?case*  
**by** (*auto intro: execn.intros*)  
**next**  
**case** *CatchMiss* **thus** *?case* **by** (*fastforce intro: execn.intros*)  
**qed**

**lemma** *execn-mark-guards-to-execn*:

$\bigwedge s\ n\ t. \Gamma \vdash \langle mark-guards\ f\ c, s \rangle =n\Rightarrow t$   
 $\implies \exists t'. \Gamma \vdash \langle c, s \rangle =n\Rightarrow t' \wedge$   
 $(isFault\ t \longrightarrow isFault\ t') \wedge$   
 $(t' = Fault\ f \longrightarrow t'=t) \wedge$   
 $(isFault\ t' \longrightarrow isFault\ t) \wedge$   
 $(\neg isFault\ t' \longrightarrow t'=t)$

**proof** (*induct c*)

**case** *Skip* **thus** *?case* **by** *auto*  
**next**  
**case** *Basic* **thus** *?case* **by** *auto*  
**next**  
**case** *Spec* **thus** *?case* **by** *auto*  
**next**

**case** (*Seq*  $c1\ c2\ s\ n\ t$ )

**have**  $exec-mark: \Gamma \vdash \langle mark-guards\ f\ (Seq\ c1\ c2), s \rangle =n\Rightarrow t$  **by** *fact*

**then obtain**  $w$  **where**

$exec-mark-c1: \Gamma \vdash \langle mark-guards\ f\ c1, s \rangle =n\Rightarrow w$  **and**

$exec-mark-c2: \Gamma \vdash \langle mark-guards\ f\ c2, w \rangle =n\Rightarrow t$

**by** (*auto elim: execn-elim-cases*)

**from** *Seq.hyps*  $exec-mark-c1$

**obtain**  $w'$  **where**

$exec-c1: \Gamma \vdash \langle c1, s \rangle =n\Rightarrow w'$  **and**

$w-Fault: isFault\ w \longrightarrow isFault\ w'$  **and**

$w'-Fault-f: w' = Fault\ f \longrightarrow w'=w$  **and**

$w'-Fault: isFault\ w' \longrightarrow isFault\ w$  **and**

$w'-noFault: \neg isFault\ w' \longrightarrow w'=w$

**by** *blast*

**show** *?case*

**proof** (*cases s*)

**case** (*Fault f*)

**with**  $exec-mark$  **have**  $t = Fault\ f$

**by** (*auto dest: execn-Fault-end*)

```

with Fault show ?thesis
  by auto
next
case Stuck
with exec-mark have  $t = \text{Stuck}$ 
  by (auto dest: execn-Stuck-end)
with Stuck show ?thesis
  by auto
next
case (Abrupt  $s'$ )
with exec-mark have  $t = \text{Abrupt } s'$ 
  by (auto dest: execn-Abrupt-end)
with Abrupt show ?thesis
  by auto
next
case (Normal  $s'$ )
show ?thesis
proof (cases isFault  $w$ )
  case True
  then obtain  $f$  where  $w' = \text{Fault } f..$ 
  moreover with exec-mark-c2
  have  $t = \text{Fault } f$ 
    by (auto dest: execn-Fault-end)
  ultimately show ?thesis
    using Normal w-Fault w'-Fault-f exec-c1
    by (fastforce intro: execn.intros elim: isFaultE)
next
case False
note noFault-w = this
show ?thesis
proof (cases isFault  $w'$ )
  case True
  then obtain  $f'$  where  $w' = \text{Fault } f'..$ 
  with Normal exec-c1
  have exec:  $\Gamma \vdash \langle \text{Seq } c1 \ c2, s \rangle = n \Rightarrow \text{Fault } f'$ 
    by (auto intro: execn.intros)
  from w'-Fault-f w' noFault-w
  have  $f' \neq f$ 
    by (cases  $w$ ) auto
  moreover
  from w' w'-Fault exec-mark-c2 have isFault  $t$ 
    by (auto dest: execn-Fault-end elim: isFaultE)
  ultimately
  show ?thesis
    using exec
    by auto
next
case False
with w'-noFault have  $w' = w$  by simp

```

```

from Seq.hyps exec-mark-c2
obtain  $t'$  where
   $\Gamma \vdash \langle c2, w \rangle = n \Rightarrow t'$  and
   $isFault\ t \longrightarrow isFault\ t'$  and
   $t' = Fault\ f \longrightarrow t' = t$  and
   $isFault\ t' \longrightarrow isFault\ t$  and
   $\neg isFault\ t' \longrightarrow t' = t$ 
  by blast
with Normal exec-c1 w'
show ?thesis
  by (fastforce intro: execn.intros)
qed
qed
qed
next
case (Cond b c1 c2 s n t)
have exec-mark:  $\Gamma \vdash \langle mark\text{-guards}\ f\ (Cond\ b\ c1\ c2), s \rangle = n \Rightarrow t$  by fact
show ?case
proof (cases s)
  case (Fault f)
    with exec-mark have  $t = Fault\ f$ 
      by (auto dest: execn-Fault-end)
    with Fault show ?thesis
      by auto
  next
    case Stuck
      with exec-mark have  $t = Stuck$ 
        by (auto dest: execn-Stuck-end)
      with Stuck show ?thesis
        by auto
  next
    case (Abrupt s')
      with exec-mark have  $t = Abrupt\ s'$ 
        by (auto dest: execn-Abrupt-end)
      with Abrupt show ?thesis
        by auto
  next
    case (Normal s')
      show ?thesis
      proof (cases s' ∈ b)
        case True
          with Normal exec-mark
          have  $\Gamma \vdash \langle mark\text{-guards}\ f\ c1\ , Normal\ s' \rangle = n \Rightarrow t$ 
            by (auto elim: execn-Normal-elim-cases)
          with Normal True Cond.hyps obtain  $t'$ 
            where  $\Gamma \vdash \langle c1, Normal\ s' \rangle = n \Rightarrow t'$ 
               $isFault\ t \longrightarrow isFault\ t'$ 
               $t' = Fault\ f \longrightarrow t' = t$ 
               $isFault\ t' \longrightarrow isFault\ t$ 

```

```

       $\neg \text{isFault } t' \longrightarrow t' = t$ 
    by blast
  with Normal True
  show ?thesis
    by (blast intro: execn.intros)
next
case False
with Normal exec-mark
have  $\Gamma \vdash \langle \text{mark-guards } f \ c2 \ , \text{Normal } s' \rangle = n \Rightarrow t$ 
  by (auto elim: execn-Normal-elim-cases)
with Normal False Cond.hyps obtain t'
  where  $\Gamma \vdash \langle c2, \text{Normal } s' \rangle = n \Rightarrow t'$ 
     $\text{isFault } t \longrightarrow \text{isFault } t'$ 
     $t' = \text{Fault } f \longrightarrow t' = t$ 
     $\text{isFault } t' \longrightarrow \text{isFault } t$ 
     $\neg \text{isFault } t' \longrightarrow t' = t$ 
  by blast
with Normal False
show ?thesis
  by (blast intro: execn.intros)
qed
next
case (While b c s n t)
have exec-mark:  $\Gamma \vdash \langle \text{mark-guards } f \ (\text{While } b \ c), s \rangle = n \Rightarrow t$  by fact
show ?case
proof (cases s)
  case (Fault f)
  with exec-mark have  $t = \text{Fault } f$ 
  by (auto dest: execn-Fault-end)
  with Fault show ?thesis
  by auto
next
case Stuck
with exec-mark have  $t = \text{Stuck}$ 
  by (auto dest: execn-Stuck-end)
with Stuck show ?thesis
  by auto
next
case (Abrupt s')
with exec-mark have  $t = \text{Abrupt } s'$ 
  by (auto dest: execn-Abrupt-end)
with Abrupt show ?thesis
  by auto
next
case (Normal s')
{
  fix c' r w
  assume exec-c':  $\Gamma \vdash \langle c', r \rangle = n \Rightarrow w$ 

```

**assume**  $c'$ :  $c' = \text{While } b \text{ (mark-guards } f \text{ } c)$   
**have**  $\exists w'$ .  $\Gamma \vdash \langle \text{While } b \text{ } c, r \rangle = n \Rightarrow w' \wedge (\text{isFault } w \longrightarrow \text{isFault } w') \wedge$   
 $(w' = \text{Fault } f \longrightarrow w' = w) \wedge (\text{isFault } w' \longrightarrow \text{isFault } w) \wedge$   
 $(\neg \text{isFault } w' \longrightarrow w' = w)$   
**using**  $\text{exec-c}' \text{ } c'$   
**proof** (*induct*)  
**case** ( $\text{WhileTrue } r \text{ } b' \text{ } c'' \text{ } n \text{ } u \text{ } w$ )  
**have**  $\text{eqs}$ :  $\text{While } b' \text{ } c'' = \text{While } b \text{ (mark-guards } f \text{ } c)$  **by fact**  
**from**  $\text{WhileTrue.hyps eqs}$   
**have**  $r\text{-in-}b$ :  $r \in b$  **by simp**  
**from**  $\text{WhileTrue.hyps eqs}$   
**have**  $\text{exec-mark-c}$ :  $\Gamma \vdash \langle \text{mark-guards } f \text{ } c, \text{Normal } r \rangle = n \Rightarrow u$  **by simp**  
**from**  $\text{WhileTrue.hyps eqs}$   
**have**  $\text{exec-mark-w}$ :  $\Gamma \vdash \langle \text{While } b \text{ (mark-guards } f \text{ } c), u \rangle = n \Rightarrow w$   
**by simp**  
**show** *?case*  
**proof** –  
**from**  $\text{WhileTrue.hyps eqs}$  **have**  $\Gamma \vdash \langle \text{mark-guards } f \text{ } c, \text{Normal } r \rangle = n \Rightarrow u$   
**by simp**  
**with**  $\text{While.hyps}$   
**obtain**  $u'$  **where**  
 $\text{exec-c}$ :  $\Gamma \vdash \langle c, \text{Normal } r \rangle = n \Rightarrow u'$  **and**  
 $u\text{-Fault}$ :  $\text{isFault } u \longrightarrow \text{isFault } u'$  **and**  
 $u'\text{-Fault-f}$ :  $u' = \text{Fault } f \longrightarrow u' = u$  **and**  
 $u'\text{-Fault}$ :  $\text{isFault } u' \longrightarrow \text{isFault } u$  **and**  
 $u'\text{-noFault}$ :  $\neg \text{isFault } u' \longrightarrow u' = u$   
**by blast**  
**show** *?thesis*  
**proof** (*cases isFault u'*)  
**case** *False*  
**with**  $u'\text{-noFault}$  **have**  $u'$ :  $u' = u$  **by simp**  
**from**  $\text{WhileTrue.hyps eqs}$  **obtain**  $w'$  **where**  
 $\Gamma \vdash \langle \text{While } b \text{ } c, u \rangle = n \Rightarrow w'$   
 $\text{isFault } w \longrightarrow \text{isFault } w'$   
 $w' = \text{Fault } f \longrightarrow w' = w$   
 $\text{isFault } w' \longrightarrow \text{isFault } w$   
 $\neg \text{isFault } w' \longrightarrow w' = w$   
**by blast**  
**with**  $u'$   $\text{exec-c}$   $r\text{-in-}b$   
**show** *?thesis*  
**by** (*blast intro: execn.WhileTrue*)  
**next**  
**case** *True*  
**then obtain**  $f'$  **where**  $u'$ :  $u' = \text{Fault } f'$ .  
**with**  $\text{exec-c}$   $r\text{-in-}b$   
**have**  $\text{exec}$ :  $\Gamma \vdash \langle \text{While } b \text{ } c, \text{Normal } r \rangle = n \Rightarrow \text{Fault } f'$   
**by** (*blast intro: execn.intros*)  
**from**  $\text{True}$   $u'\text{-Fault}$  **have**  $\text{isFault } u$   
**by simp**



```

    then obtain  $f$  where  $u: u = \text{Fault } f..$ 
    with  $\text{exec-mark-}w$  have  $w = \text{Fault } f$ 
      by (auto dest:  $\text{execn-Fault-end}$ )
    with  $\text{exec } u' u u' \text{-Fault-}f$ 
    show ?thesis
      by auto
  qed
next
case (WhileFalse  $r b' c'' n$ )
have eqs:  $\text{While } b' c'' = \text{While } b (\text{mark-guards } f c)$  by fact
from WhileFalse.hyps eqs
have  $r \text{-not-in-}b: r \notin b$  by simp
show ?case
proof -
  from  $r \text{-not-in-}b$ 
  have  $\Gamma \vdash \langle \text{While } b c, \text{Normal } r \rangle = n \Rightarrow \text{Normal } r$ 
    by (rule  $\text{execn.WhileFalse}$ )
  thus ?thesis
    by blast
qed
qed auto
} note  $\text{hyp-while} = \text{this}$ 
show ?thesis
proof (cases  $s' \in b$ )
  case False
  with Normal  $\text{exec-mark}$ 
  have  $t = s$ 
    by (auto elim:  $\text{execn-Normal-elim-cases}$ )
  with Normal False show ?thesis
    by (auto intro:  $\text{execn.intros}$ )
next
case True note  $s' \text{-in-}b = \text{this}$ 
with Normal  $\text{exec-mark}$  obtain  $r$  where
   $\text{exec-mark-c}: \Gamma \vdash \langle \text{mark-guards } f c, \text{Normal } s' \rangle = n \Rightarrow r$  and
   $\text{exec-mark-w}: \Gamma \vdash \langle \text{While } b (\text{mark-guards } f c), r \rangle = n \Rightarrow t$ 
  by (auto elim:  $\text{execn-Normal-elim-cases}$ )
from While.hyps  $\text{exec-mark-c}$  obtain  $r'$  where
   $\text{exec-c}: \Gamma \vdash \langle c, \text{Normal } s' \rangle = n \Rightarrow r'$  and
   $r \text{-Fault}: \text{isFault } r \longrightarrow \text{isFault } r'$  and
   $r' \text{-Fault-f}: r' = \text{Fault } f \longrightarrow r' = r$  and
   $r' \text{-Fault}: \text{isFault } r' \longrightarrow \text{isFault } r$  and
   $r' \text{-noFault}: \neg \text{isFault } r' \longrightarrow r' = r$ 
  by blast
show ?thesis
proof (cases  $\text{isFault } r'$ )
  case False
  with  $r' \text{-noFault}$  have  $r': r' = r$  by simp
  from  $\text{hyp-while } \text{exec-mark-}w$ 

```

```

obtain  $t'$  where
   $\Gamma \vdash \langle \text{While } b \ c, r \rangle = n \Rightarrow t'$ 
   $\text{isFault } t \longrightarrow \text{isFault } t'$ 
   $t' = \text{Fault } f \longrightarrow t' = t$ 
   $\text{isFault } t' \longrightarrow \text{isFault } t$ 
   $\neg \text{isFault } t' \longrightarrow t' = t$ 
  by blast
with  $r'$  exec-c Normal s'-in-b
show ?thesis
  by (blast intro: execn.intros)
next
case True
then obtain  $f'$  where  $r'$ :  $r' = \text{Fault } f'$ ..
hence  $\Gamma \vdash \langle \text{While } b \ c, r' \rangle = n \Rightarrow \text{Fault } f'$ 
  by auto
with Normal s'-in-b exec-c
have exec:  $\Gamma \vdash \langle \text{While } b \ c, \text{Normal } s' \rangle = n \Rightarrow \text{Fault } f'$ 
  by (auto intro: execn.intros)
from True r'-Fault
have  $\text{isFault } r$ 
  by simp
then obtain  $f$  where  $r$ :  $r = \text{Fault } f$ ..
with exec-mark-w have  $t = \text{Fault } f$ 
  by (auto dest: execn-Fault-end)
with Normal exec r' r r'-Fault-f
show ?thesis
  by auto
qed
qed
qed
next
case Call thus ?case by auto
next
case DynCom thus ?case
  by (fastforce elim!: execn-elim-cases intro: execn.intros)
next
case (Guard f' g c s n t)
have exec-mark:  $\Gamma \vdash \langle \text{mark-guards } f \ (\text{Guard } f' \ g \ c), s \rangle = n \Rightarrow t$  by fact
show ?case
proof (cases s)
  case (Fault f)
  with exec-mark have  $t = \text{Fault } f$ 
    by (auto dest: execn-Fault-end)
  with Fault show ?thesis
    by auto
next
case Stuck
with exec-mark have  $t = \text{Stuck}$ 
  by (auto dest: execn-Stuck-end)

```

```

with Stuck show ?thesis
  by auto
next
case (Abrupt s')
with exec-mark have  $t = \text{Abrupt } s'$ 
  by (auto dest: execn-Abrupt-end)
with Abrupt show ?thesis
  by auto
next
case (Normal s')
show ?thesis
proof (cases s' ∈ g)
  case False
with Normal exec-mark have  $t = \text{Fault } f$ 
  by (auto elim: execn-Normal-elim-cases)
from False
have  $\Gamma \vdash \langle \text{Guard } f' \ g \ c, \text{Normal } s' \rangle = n \Rightarrow \text{Fault } f'$ 
  by (blast intro: execn.intros)
with Normal t show ?thesis
  by auto
next
case True
with exec-mark Normal
have  $\Gamma \vdash \langle \text{mark-guards } f \ c, \text{Normal } s' \rangle = n \Rightarrow t$ 
  by (auto elim: execn-Normal-elim-cases)
with Guard.hyps obtain  $t'$  where
   $\Gamma \vdash \langle c, \text{Normal } s' \rangle = n \Rightarrow t'$  and
   $\text{isFault } t \longrightarrow \text{isFault } t'$  and
   $t' = \text{Fault } f \longrightarrow t' = t$  and
   $\text{isFault } t' \longrightarrow \text{isFault } t$  and
   $\neg \text{isFault } t' \longrightarrow t' = t$ 
  by blast
with Normal True
show ?thesis
  by (blast intro: execn.intros)
qed
qed
next
case Throw thus ?case by auto
next
case (Catch c1 c2 s n t)
have exec-mark:  $\Gamma \vdash \langle \text{mark-guards } f \ (\text{Catch } c1 \ c2), s \rangle = n \Rightarrow t$  by fact
show ?case
proof (cases s)
  case (Fault f)
with exec-mark have  $t = \text{Fault } f$ 
  by (auto dest: execn-Fault-end)
with Fault show ?thesis
  by auto

```

```

next
  case Stuck
  with exec-mark have  $t = \text{Stuck}$ 
  by (auto dest: execn-Stuck-end)
  with Stuck show ?thesis
  by auto
next
  case (Abrupt  $s'$ )
  with exec-mark have  $t = \text{Abrupt } s'$ 
  by (auto dest: execn-Abrupt-end)
  with Abrupt show ?thesis
  by auto
next
  case (Normal  $s'$ ) note  $s = \text{this}$ 
  with exec-mark have
     $\Gamma \vdash \langle \text{Catch } (\text{mark-guards } f \ c1) \ (\text{mark-guards } f \ c2), \text{Normal } s' \rangle = n \Rightarrow t$  by simp
  thus ?thesis
  proof (cases)
    fix  $w$ 
    assume exec-mark-c1:  $\Gamma \vdash \langle \text{mark-guards } f \ c1, \text{Normal } s' \rangle = n \Rightarrow \text{Abrupt } w$ 
    assume exec-mark-c2:  $\Gamma \vdash \langle \text{mark-guards } f \ c2, \text{Normal } w \rangle = n \Rightarrow t$ 
    from exec-mark-c1 Catch.hyps
    obtain  $w'$  where
      exec-c1:  $\Gamma \vdash \langle c1, \text{Normal } s' \rangle = n \Rightarrow w'$  and
      w'-Fault-f:  $w' = \text{Fault } f \longrightarrow w' = \text{Abrupt } w$  and
      w'-Fault:  $\text{isFault } w' \longrightarrow \text{isFault } (\text{Abrupt } w)$  and
      w'-noFault:  $\neg \text{isFault } w' \longrightarrow w' = \text{Abrupt } w$ 
    by fastforce
  show ?thesis
  proof (cases  $w'$ )
    case (Fault  $f'$ )
    with Normal exec-c1 have  $\Gamma \vdash \langle \text{Catch } c1 \ c2, s \rangle = n \Rightarrow \text{Fault } f'$ 
    by (auto intro: execn.intros)
    with w'-Fault Fault show ?thesis
    by auto
  next
  case Stuck
  with w'-noFault have False
  by simp
  thus ?thesis ..
  next
  case (Normal  $w''$ )
  with w'-noFault have False by simp thus ?thesis ..
  next
  case (Abrupt  $w''$ )
  with w'-noFault have  $w'' = w$  by simp
  from exec-mark-c2 Catch.hyps
  obtain  $t'$  where
     $\Gamma \vdash \langle c2, \text{Normal } w \rangle = n \Rightarrow t'$ 

```

```

    isFault t  $\longrightarrow$  isFault t'
    t' = Fault f  $\longrightarrow$  t'=t
    isFault t'  $\longrightarrow$  isFault t
     $\neg$  isFault t'  $\longrightarrow$  t'=t
    by blast
  with w'' Abrupt s exec-c1
  show ?thesis
    by (blast intro: execn.intros)
qed
next
  assume t:  $\neg$  isAbr t
  assume  $\Gamma \vdash \langle \text{mark-guards } f \ c1, \text{Normal } s^\wedge \rangle = n \Rightarrow t$ 
  with Catch.hyps
  obtain t' where
    exec-c1:  $\Gamma \vdash \langle c1, \text{Normal } s^\wedge \rangle = n \Rightarrow t'$  and
    t-Fault: isFault t  $\longrightarrow$  isFault t' and
    t'-Fault-f: t' = Fault f  $\longrightarrow$  t'=t and
    t'-Fault: isFault t'  $\longrightarrow$  isFault t and
    t'-noFault:  $\neg$  isFault t'  $\longrightarrow$  t'=t
    by blast
  show ?thesis
  proof (cases isFault t')
    case True
      then obtain f' where t': t'=Fault f'..
      with exec-c1 have  $\Gamma \vdash \langle \text{Catch } c1 \ c2, \text{Normal } s^\wedge \rangle = n \Rightarrow \text{Fault } f'$ 
        by (auto intro: execn.intros)
      with t'-Fault-f t'-Fault t' s show ?thesis
        by auto
    next
      case False
        with t'-noFault have t'=t by simp
        with t exec-c1 s show ?thesis
          by (blast intro: execn.intros)
  qed
qed
qed
qed

lemma exec-to-exec-mark-guards:
  assumes exec-c:  $\Gamma \vdash \langle c, s \rangle \Rightarrow t$ 
  assumes t-not-Fault:  $\neg$  isFault t
  shows  $\Gamma \vdash \langle \text{mark-guards } f \ c, s \rangle \Rightarrow t$ 
proof -
  from exec-to-execn [OF exec-c] obtain n where
     $\Gamma \vdash \langle c, s \rangle = n \Rightarrow t$  ..
  from execn-to-execn-mark-guards [OF this t-not-Fault]
  show ?thesis
    by (blast intro: execn-to-exec)
qed

```

**lemma** *exec-to-exec-mark-guards-Fault*:  
**assumes** *exec-c*:  $\Gamma \vdash \langle c, s \rangle \Rightarrow \text{Fault } f$   
**shows**  $\exists f'. \Gamma \vdash \langle \text{mark-guards } x \ c, s \rangle \Rightarrow \text{Fault } f'$   
**proof** –  
**from** *exec-to-execn* [OF *exec-c*] **obtain** *n* **where**  
 $\Gamma \vdash \langle c, s \rangle = n \Rightarrow \text{Fault } f$  ..  
**from** *execn-to-execn-mark-guards-Fault* [OF *this*]  
**show** ?thesis  
**by** (*blast intro: execn-to-exec*)  
**qed**

**lemma** *exec-mark-guards-to-exec*:  
**assumes** *exec-mark*:  $\Gamma \vdash \langle \text{mark-guards } f \ c, s \rangle \Rightarrow t$   
**shows**  $\exists t'. \Gamma \vdash \langle c, s \rangle \Rightarrow t' \wedge$   
 $(\text{isFault } t \longrightarrow \text{isFault } t') \wedge$   
 $(t' = \text{Fault } f \longrightarrow t' = t) \wedge$   
 $(\text{isFault } t' \longrightarrow \text{isFault } t) \wedge$   
 $(\neg \text{isFault } t' \longrightarrow t' = t)$

**proof** –  
**from** *exec-to-execn* [OF *exec-mark*] **obtain** *n* **where**  
 $\Gamma \vdash \langle \text{mark-guards } f \ c, s \rangle = n \Rightarrow t$  ..  
**from** *execn-mark-guards-to-execn* [OF *this*]  
**show** ?thesis  
**by** (*blast intro: execn-to-exec*)  
**qed**

### 3.7 Lemmas about *strip-guards*

**lemma** *execn-to-execn-strip-guards*:  
**assumes** *exec-c*:  $\Gamma \vdash \langle c, s \rangle = n \Rightarrow t$   
**assumes** *t-not-Fault*:  $\neg \text{isFault } t$   
**shows**  $\Gamma \vdash \langle \text{strip-guards } F \ c, s \rangle = n \Rightarrow t$   
**using** *exec-c t-not-Fault* [*simplified not-isFault-iff*]  
**by** (*induct*) (*auto intro: execn.intros dest: noFaultn-startD*)

**lemma** *execn-to-execn-strip-guards-Fault*:  
**assumes** *exec-c*:  $\Gamma \vdash \langle c, s \rangle = n \Rightarrow t$   
**shows**  $\bigwedge f. \llbracket t = \text{Fault } f; f \notin F \rrbracket \Longrightarrow \Gamma \vdash \langle \text{strip-guards } F \ c, s \rangle = n \Rightarrow \text{Fault } f$   
**using** *exec-c*  
**proof** (*induct*)  
**case** *Skip* **thus** ?case **by** *auto*  
**next**  
**case** *Guard* **thus** ?case **by** (*fastforce intro: execn.intros*)  
**next**  
**case** *GuardFault* **thus** ?case **by** (*fastforce intro: execn.intros*)  
**next**

```

  case FaultProp thus ?case by auto
next
  case Basic thus ?case by auto
next
  case Spec thus ?case by auto
next
  case SpecStuck thus ?case by auto
next
  case (Seq c1 s n w c2 t)
  have exec-c1:  $\Gamma \vdash \langle c1, Normal\ s \rangle =n \Rightarrow w$  by fact
  have exec-c2:  $\Gamma \vdash \langle c2, w \rangle =n \Rightarrow t$  by fact
  have t:  $t = Fault\ f$  by fact
  have notinF:  $f \notin F$  by fact
  show ?case
  proof (cases w)
    case (Fault f')
      with exec-c2 t have  $f' = f$ 
        by (auto dest: execn-Fault-end)
      with Fault notinF Seq.hyps
      have  $\Gamma \vdash \langle strip\text{-}guards\ F\ c1, Normal\ s \rangle =n \Rightarrow Fault\ f$ 
        by auto
      moreover have  $\Gamma \vdash \langle strip\text{-}guards\ F\ c2, Fault\ f \rangle =n \Rightarrow Fault\ f$ 
        by auto
      ultimately show ?thesis
        by (auto intro: execn.intros)
  next
    case (Normal s')
      with execn-to-execn-strip-guards [OF exec-c1]
      have exec-strip-c1:  $\Gamma \vdash \langle strip\text{-}guards\ F\ c1, Normal\ s \rangle =n \Rightarrow w$ 
        by simp
      with Seq.hyps t notinF
      have  $\Gamma \vdash \langle strip\text{-}guards\ F\ c2, w \rangle =n \Rightarrow Fault\ f$ 
        by blast
      with exec-strip-c1 show ?thesis
        by (auto intro: execn.intros)
  next
    case (Abrupt s')
      with execn-to-execn-strip-guards [OF exec-c1]
      have exec-strip-c1:  $\Gamma \vdash \langle strip\text{-}guards\ F\ c1, Normal\ s \rangle =n \Rightarrow w$ 
        by simp
      with Seq.hyps t notinF
      have  $\Gamma \vdash \langle strip\text{-}guards\ F\ c2, w \rangle =n \Rightarrow Fault\ f$ 
        by (auto intro: execn.intros)
      with exec-strip-c1 show ?thesis
        by (auto intro: execn.intros)
  next
    case Stuck
      with exec-c2 have  $t = Stuck$ 
        by (auto dest: execn-Stuck-end)

```

```

    with t show ?thesis by simp
  qed
next
  case CondTrue thus ?case by (fastforce intro: execn.intros)
next
  case CondFalse thus ?case by (fastforce intro: execn.intros)
next
  case (WhileTrue s b c n w t)
  have exec-c:  $\Gamma \vdash \langle c, Normal\ s \rangle = n \Rightarrow w$  by fact
  have exec-w:  $\Gamma \vdash \langle While\ b\ c, w \rangle = n \Rightarrow t$  by fact
  have t:  $t = Fault\ f$  by fact
  have notinF:  $f \notin F$  by fact
  have s-in-b:  $s \in b$  by fact
  show ?case
  proof (cases w)
    case (Fault f')
    with exec-w t have f'=f
      by (auto dest: execn-Fault-end)
    with Fault notinF WhileTrue.hyps
    have  $\Gamma \vdash \langle strip\text{-}guards\ F\ c, Normal\ s \rangle = n \Rightarrow Fault\ f$ 
      by auto
    moreover have  $\Gamma \vdash \langle strip\text{-}guards\ F\ (While\ b\ c), Fault\ f \rangle = n \Rightarrow Fault\ f$ 
      by auto
    ultimately show ?thesis
      using s-in-b by (auto intro: execn.intros)
  next
    case (Normal s')
    with execn-to-execn-strip-guards [OF exec-c]
    have exec-strip-c:  $\Gamma \vdash \langle strip\text{-}guards\ F\ c, Normal\ s \rangle = n \Rightarrow w$ 
      by simp
    with WhileTrue.hyps t notinF
    have  $\Gamma \vdash \langle strip\text{-}guards\ F\ (While\ b\ c), w \rangle = n \Rightarrow Fault\ f$ 
      by blast
    with exec-strip-c s-in-b show ?thesis
      by (auto intro: execn.intros)
  next
    case (Abrupt s')
    with execn-to-execn-strip-guards [OF exec-c]
    have exec-strip-c:  $\Gamma \vdash \langle strip\text{-}guards\ F\ c, Normal\ s \rangle = n \Rightarrow w$ 
      by simp
    with WhileTrue.hyps t notinF
    have  $\Gamma \vdash \langle strip\text{-}guards\ F\ (While\ b\ c), w \rangle = n \Rightarrow Fault\ f$ 
      by (auto intro: execn.intros)
    with exec-strip-c s-in-b show ?thesis
      by (auto intro: execn.intros)
  next
    case Stuck
    with exec-w have t=Stuck
      by (auto dest: execn-Stuck-end)

```



```

    with  $t$  show ?thesis by simp
  qed
next
  case WhileFalse thus ?case by (fastforce intro: execn.intros)
next
  case Call thus ?case by (fastforce intro: execn.intros)
next
  case CallUndefined thus ?case by simp
next
  case StuckProp thus ?case by simp
next
  case DynCom thus ?case by (fastforce intro: execn.intros)
next
  case Throw thus ?case by simp
next
  case AbruptProp thus ?case by simp
next
  case (CatchMatch  $c1$   $s$   $n$   $w$   $c2$   $t$ )
  have exec-c1:  $\Gamma \vdash \langle c1, Normal\ s \rangle =n\Rightarrow Abrupt\ w$  by fact
  have exec-c2:  $\Gamma \vdash \langle c2, Normal\ w \rangle =n\Rightarrow t$  by fact
  have  $t = Fault\ f$  by fact
  have notinF:  $f \notin F$  by fact
  from execn-to-execn-strip-guards [OF exec-c1]
  have exec-strip-c1:  $\Gamma \vdash \langle strip-guards\ F\ c1, Normal\ s \rangle =n\Rightarrow Abrupt\ w$ 
    by simp
  with CatchMatch.hyps  $t$  notinF
  have  $\Gamma \vdash \langle strip-guards\ F\ c2, Normal\ w \rangle =n\Rightarrow Fault\ f$ 
    by blast
  with exec-strip-c1 show ?case
    by (auto intro: execn.intros)
next
  case CatchMiss thus ?case by (fastforce intro: execn.intros)
qed

```

```

lemma execn-to-execn-strip-guards':
  assumes exec-c:  $\Gamma \vdash \langle c, s \rangle =n\Rightarrow t$ 
  assumes t-not-Fault:  $t \notin Fault\ 'F$ 
  shows  $\Gamma \vdash \langle strip-guards\ F\ c, s \rangle =n\Rightarrow t$ 
proof (cases  $t$ )
  case (Fault  $f$ )
  with t-not-Fault exec-c show ?thesis
    by (auto intro: execn-to-execn-strip-guards-Fault)
qed (insert exec-c, auto intro: execn-to-execn-strip-guards)

```

```

lemma execn-strip-guards-to-execn:
   $\bigwedge s\ n\ t. \Gamma \vdash \langle strip-guards\ F\ c, s \rangle =n\Rightarrow t$ 
 $\implies \exists t'. \Gamma \vdash \langle c, s \rangle =n\Rightarrow t' \wedge$ 
  (isFault  $t \longrightarrow isFault\ t'$ )  $\wedge$ 
  ( $t' \in Fault\ '(-\ F) \longrightarrow t'=t$ )  $\wedge$ 

```

```

      ( $\neg$  isFault  $t' \longrightarrow t'=t$ )
proof (induct  $c$ )
  case Skip thus ?case by auto
next
  case Basic thus ?case by auto
next
  case Spec thus ?case by auto
next
  case (Seq  $c1\ c2\ s\ n\ t$ )
have exec-strip:  $\Gamma \vdash \langle \text{strip-guards } F\ (Seq\ c1\ c2), s \rangle = n \Rightarrow t$  by fact
then obtain  $w$  where
  exec-strip-c1:  $\Gamma \vdash \langle \text{strip-guards } F\ c1, s \rangle = n \Rightarrow w$  and
  exec-strip-c2:  $\Gamma \vdash \langle \text{strip-guards } F\ c2, w \rangle = n \Rightarrow t$ 
  by (auto elim: execn-elim-cases)
from Seq.hyps exec-strip-c1
obtain  $w'$  where
  exec-c1:  $\Gamma \vdash \langle c1, s \rangle = n \Rightarrow w'$  and
  w'-Fault: isFault  $w \longrightarrow$  isFault  $w'$  and
  w'-Fault:  $w' \in \text{Fault } (-\ F) \longrightarrow w'=w$  and
  w'-noFault:  $\neg$  isFault  $w' \longrightarrow w'=w$ 
  by blast
show ?case
proof (cases  $s$ )
  case (Fault  $f$ )
    with exec-strip have  $t = \text{Fault } f$ 
      by (auto dest: execn-Fault-end)
    with Fault show ?thesis
      by auto
  next
  case Stuck
    with exec-strip have  $t = \text{Stuck}$ 
      by (auto dest: execn-Stuck-end)
    with Stuck show ?thesis
      by auto
  next
  case (Abrupt  $s'$ )
    with exec-strip have  $t = \text{Abrupt } s'$ 
      by (auto dest: execn-Abrupt-end)
    with Abrupt show ?thesis
      by auto
  next
  case (Normal  $s'$ )
    show ?thesis
    proof (cases isFault  $w$ )
      case True
        then obtain  $f$  where  $w': w = \text{Fault } f..$ 
        moreover with exec-strip-c2
        have  $t: t = \text{Fault } f$ 
          by (auto dest: execn-Fault-end)

```

```

ultimately show ?thesis
  using Normal w-Fault w'-Fault exec-c1
  by (fastforce intro: execn.intros elim: isFaultE)
next
case False
note noFault-w = this
show ?thesis
proof (cases isFault w')
  case True
  then obtain f' where w': w'=Fault f'..
  with Normal exec-c1
  have exec:  $\Gamma \vdash \langle \text{Seq } c1 \ c2, s \rangle =n \Rightarrow \text{Fault } f'$ 
    by (auto intro: execn.intros)
  from w'-Fault w' noFault-w
  have f'  $\in F$ 
    by (cases w) auto
  with exec
  show ?thesis
    by auto
next
case False
with w'-noFault have w': w'=w by simp
from Seq.hyps exec-strip-c2
obtain t' where
   $\Gamma \vdash \langle c2, w \rangle =n \Rightarrow t'$  and
   $\text{isFault } t \longrightarrow \text{isFault } t'$  and
   $t' \in \text{Fault } (-F) \longrightarrow t'=t$  and
   $\neg \text{isFault } t' \longrightarrow t'=t$ 
  by blast
with Normal exec-c1 w'
show ?thesis
  by (fastforce intro: execn.intros)
qed
qed
qed
next
next
case (Cond b c1 c2 s n t)
have exec-strip:  $\Gamma \vdash \langle \text{strip-guards } F \ (\text{Cond } b \ c1 \ c2), s \rangle =n \Rightarrow t$  by fact
show ?case
proof (cases s)
  case (Fault f)
  with exec-strip have t=Fault f
    by (auto dest: execn-Fault-end)
  with Fault show ?thesis
    by auto
next
case Stuck
with exec-strip have t=Stuck

```

```

    by (auto dest: execn-Stuck-end)
  with Stuck show ?thesis
    by auto
next
  case (Abrupt s')
  with exec-strip have t=Abrupt s'
    by (auto dest: execn-Abrupt-end)
  with Abrupt show ?thesis
    by auto
next
  case (Normal s')
  show ?thesis
  proof (cases s' ∈ b)
    case True
    with Normal exec-strip
    have  $\Gamma \vdash \langle \text{strip-guards } F \ c1 \ , \text{Normal } s' \rangle = n \Rightarrow t$ 
      by (auto elim: execn-Normal-elim-cases)
    with Normal True Cond.hyps obtain t'
      where  $\Gamma \vdash \langle c1, \text{Normal } s' \rangle = n \Rightarrow t'$ 
             $\text{isFault } t \longrightarrow \text{isFault } t'$ 
             $t' \in \text{Fault } '(-F) \longrightarrow t' = t$ 
             $\neg \text{isFault } t' \longrightarrow t' = t$ 
      by blast
    with Normal True
    show ?thesis
      by (blast intro: execn.intros)
  next
  case False
  with Normal exec-strip
  have  $\Gamma \vdash \langle \text{strip-guards } F \ c2 \ , \text{Normal } s' \rangle = n \Rightarrow t$ 
    by (auto elim: execn-Normal-elim-cases)
  with Normal False Cond.hyps obtain t'
    where  $\Gamma \vdash \langle c2, \text{Normal } s' \rangle = n \Rightarrow t'$ 
           $\text{isFault } t \longrightarrow \text{isFault } t'$ 
           $t' \in \text{Fault } '(-F) \longrightarrow t' = t$ 
           $\neg \text{isFault } t' \longrightarrow t' = t$ 
    by blast
  with Normal False
  show ?thesis
    by (blast intro: execn.intros)
qed
qed
next
  case (While b c s n t)
  have exec-strip:  $\Gamma \vdash \langle \text{strip-guards } F \ (\text{While } b \ c), s \rangle = n \Rightarrow t$  by fact
  show ?case
  proof (cases s)
    case (Fault f)
    with exec-strip have t=Fault f

```

```

    by (auto dest: execn-Fault-end)
  with Fault show ?thesis
    by auto
next
case Stuck
with exec-strip have t=Stuck
  by (auto dest: execn-Stuck-end)
with Stuck show ?thesis
  by auto
next
case (Abrupt s')
with exec-strip have t=Abrupt s'
  by (auto dest: execn-Abrupt-end)
with Abrupt show ?thesis
  by auto
next
case (Normal s')
{
  fix c' r w
  assume exec-c':  $\Gamma \vdash \langle c', r \rangle = n \Rightarrow w$ 
  assume c':  $c' = \text{While } b \text{ (strip-guards } F \text{ } c)$ 
  have  $\exists w'. \Gamma \vdash \langle \text{While } b \text{ } c, r \rangle = n \Rightarrow w' \wedge (\text{isFault } w \longrightarrow \text{isFault } w') \wedge$ 
     $(w' \in \text{Fault } '(-F) \longrightarrow w'=w) \wedge$ 
     $(\neg \text{isFault } w' \longrightarrow w'=w)$ 
  using exec-c' c'
proof (induct)
  case (WhileTrue r b' c'' n u w)
  have eqs:  $\text{While } b' \text{ } c'' = \text{While } b \text{ (strip-guards } F \text{ } c)$  by fact
  from WhileTrue.hyps eqs
  have r-in-b:  $r \in b$  by simp
  from WhileTrue.hyps eqs
  have exec-strip-c:  $\Gamma \vdash \langle \text{strip-guards } F \text{ } c, \text{Normal } r \rangle = n \Rightarrow u$  by simp
  from WhileTrue.hyps eqs
  have exec-strip-w:  $\Gamma \vdash \langle \text{While } b \text{ (strip-guards } F \text{ } c), u \rangle = n \Rightarrow w$ 
  by simp
  show ?case
proof -
  from WhileTrue.hyps eqs have  $\Gamma \vdash \langle \text{strip-guards } F \text{ } c, \text{Normal } r \rangle = n \Rightarrow u$ 
  by simp
  with While.hyps
  obtain u' where
    exec-c:  $\Gamma \vdash \langle c, \text{Normal } r \rangle = n \Rightarrow u'$  and
    u-Fault:  $\text{isFault } u \longrightarrow \text{isFault } u'$  and
    u'-Fault:  $u' \in \text{Fault } '(-F) \longrightarrow u'=u$  and
    u'-noFault:  $\neg \text{isFault } u' \longrightarrow u'=u$ 
  by blast
  show ?thesis
proof (cases isFault u')
  case False

```

```

with  $u'$ -noFault have  $u'$ :  $u'=u$  by simp
from WhileTrue.hyps eqs obtain  $w'$  where
   $\Gamma \vdash \langle \text{While } b \ c, u \rangle = n \Rightarrow w'$ 
   $\text{isFault } w \longrightarrow \text{isFault } w'$ 
   $w' \in \text{Fault } (-F) \longrightarrow w' = w$ 
   $\neg \text{isFault } w' \longrightarrow w' = w$ 
by blast
with  $u'$  exec-c r-in-b
show ?thesis
by (blast intro: execn.WhileTrue)
next
case True
then obtain  $f'$  where  $u'$ :  $u' = \text{Fault } f'$ ..
with exec-c r-in-b
have exec:  $\Gamma \vdash \langle \text{While } b \ c, \text{Normal } r \rangle = n \Rightarrow \text{Fault } f'$ 
by (blast intro: execn.intros)
show ?thesis
proof (cases isFault u)
case True
then obtain  $f$  where  $u$ :  $u = \text{Fault } f$ ..
with exec-strip-w have  $w = \text{Fault } f$ 
by (auto dest: execn-Fault-end)
with exec u' u u'-Fault
show ?thesis
by auto
next
case False
with  $u'$ -Fault  $u'$  have  $f' \in F$ 
by (cases u) auto
with exec show ?thesis
by auto
qed
qed
qed
next
case (WhileFalse r b' c'' n)
have eqs:  $\text{While } b' \ c'' = \text{While } b \ (\text{strip-guards } F \ c)$  by fact
from WhileFalse.hyps eqs
have  $r$ -not-in- $b$ :  $r \notin b$  by simp
show ?case
proof –
from  $r$ -not-in- $b$ 
have  $\Gamma \vdash \langle \text{While } b \ c, \text{Normal } r \rangle = n \Rightarrow \text{Normal } r$ 
by (rule execn.WhileFalse)
thus ?thesis
by blast
qed
qed auto
} note hyp-while = this

```

**show** *?thesis*  
**proof** (*cases s' ∈ b*)  
    **case** *False*  
    **with** *Normal exec-strip*  
    **have**  $t=s$   
    **by** (*auto elim: execn-Normal-elim-cases*)  
    **with** *Normal False show ?thesis*  
    **by** (*auto intro: execn.intros*)  
**next**  
    **case** *True note s'-in-b = this*  
    **with** *Normal exec-strip obtain r where*  
    *exec-strip-c:  $\Gamma \vdash \langle \text{strip-guards } F \ c, \text{Normal } s \rangle = n \Rightarrow r$  and*  
    *exec-strip-w:  $\Gamma \vdash \langle \text{While } b \ (\text{strip-guards } F \ c), r \rangle = n \Rightarrow t$*   
    **by** (*auto elim: execn-Normal-elim-cases*)  
    **from** *While.hyps exec-strip-c obtain r' where*  
    *exec-c:  $\Gamma \vdash \langle c, \text{Normal } s \rangle = n \Rightarrow r'$  and*  
    *r'-Fault:  $\text{isFault } r \longrightarrow \text{isFault } r'$  and*  
    *r'-Fault:  $r' \in \text{Fault } (-F) \longrightarrow r'=r$  and*  
    *r'-noFault:  $\neg \text{isFault } r' \longrightarrow r'=r$*   
    **by** *blast*  
    **show** *?thesis*  
    **proof** (*cases isFault r'*)  
    **case** *False*  
    **with** *r'-noFault have r': r'=r by simp*  
    **from** *hyp-while exec-strip-w*  
    **obtain** *t' where*  
     *$\Gamma \vdash \langle \text{While } b \ c, r \rangle = n \Rightarrow t'$*   
     *$\text{isFault } t \longrightarrow \text{isFault } t'$*   
     *$t' \in \text{Fault } (-F) \longrightarrow t'=t$*   
     *$\neg \text{isFault } t' \longrightarrow t'=t$*   
    **by** *blast*  
    **with** *r' exec-c Normal s'-in-b*  
    **show** *?thesis*  
    **by** (*blast intro: execn.intros*)  
**next**  
    **case** *True*  
    **then obtain** *f' where r': r'=Fault f'..*  
    **hence**  $\Gamma \vdash \langle \text{While } b \ c, r \rangle = n \Rightarrow \text{Fault } f'$   
    **by** *auto*  
    **with** *Normal s'-in-b exec-c*  
    **have** *exec:  $\Gamma \vdash \langle \text{While } b \ c, \text{Normal } s \rangle = n \Rightarrow \text{Fault } f'$*   
    **by** (*auto intro: execn.intros*)  
    **show** *?thesis*  
    **proof** (*cases isFault r*)  
    **case** *True*  
    **then obtain** *f where r: r=Fault f..*  
    **with** *exec-strip-w have t=Fault f*  
    **by** (*auto dest: execn-Fault-end*)  
    **with** *Normal exec r' r r'-Fault*

```

      show ?thesis
      by auto
    next
      case False
      with  $r'$ -Fault  $r'$  have  $f' \in F$ 
      by (cases  $r$ ) auto
      with Normal exec show ?thesis
      by auto
    qed
  qed
qed
next
  case Call thus ?case by auto
next
  case DynCom thus ?case
  by (fastforce elim!: execn-elim-cases intro: execn.intros)
next
  case (Guard  $f g c s n t$ )
  have exec-strip:  $\Gamma \vdash \langle \text{strip-guards } F \text{ (Guard } f g c), s \rangle =n \Rightarrow t$  by fact
  show ?case
  proof (cases  $s$ )
    case (Fault  $f$ )
    with exec-strip have  $t = \text{Fault } f$ 
    by (auto dest: execn-Fault-end)
    with Fault show ?thesis
    by auto
  next
    case Stuck
    with exec-strip have  $t = \text{Stuck}$ 
    by (auto dest: execn-Stuck-end)
    with Stuck show ?thesis
    by auto
  next
    case (Abrupt  $s'$ )
    with exec-strip have  $t = \text{Abrupt } s'$ 
    by (auto dest: execn-Abrupt-end)
    with Abrupt show ?thesis
    by auto
  next
    case (Normal  $s'$ )
    show ?thesis
    proof (cases  $f \in F$ )
      case True
      with exec-strip Normal
      have exec-strip-c:  $\Gamma \vdash \langle \text{strip-guards } F \text{ } c, \text{Normal } s' \rangle =n \Rightarrow t$ 
      by simp
      with Guard.hyps obtain  $t'$  where
       $\Gamma \vdash \langle c, \text{Normal } s' \rangle =n \Rightarrow t'$  and

```



```

    isFault t  $\longrightarrow$  isFault t' and
    t'  $\in$  Fault ' (-F)  $\longrightarrow$  t'=t and
     $\neg$  isFault t'  $\longrightarrow$  t'=t
  by blast
with Normal True
show ?thesis
  by (cases s'  $\in$  g) (fastforce intro: execn.intros)+
next
case False
note f-notin-F = this
show ?thesis
proof (cases s'  $\in$  g)
  case False
  with Normal exec-strip f-notin-F have t: t=Fault f
  by (auto elim: execn-Normal-elim-cases)
  from False
  have  $\Gamma \vdash \langle \text{Guard } f \ g \ c, \text{Normal } s' \rangle = n \Rightarrow \text{Fault } f$ 
  by (blast intro: execn.intros)
  with False Normal t show ?thesis
  by auto
next
case True
with exec-strip Normal f-notin-F
have  $\Gamma \vdash \langle \text{strip-guards } F \ c, \text{Normal } s' \rangle = n \Rightarrow t$ 
  by (auto elim: execn-Normal-elim-cases)
with Guard.hyps obtain t' where
   $\Gamma \vdash \langle c, \text{Normal } s' \rangle = n \Rightarrow t'$  and
  isFault t  $\longrightarrow$  isFault t' and
  t'  $\in$  Fault ' (-F)  $\longrightarrow$  t'=t and
   $\neg$  isFault t'  $\longrightarrow$  t'=t
  by blast
with Normal True
show ?thesis
  by (blast intro: execn.intros)
qed
qed
qed
next
case Throw thus ?case by auto
next
case (Catch c1 c2 s n t)
have exec-strip:  $\Gamma \vdash \langle \text{strip-guards } F \ (\text{Catch } c1 \ c2), s \rangle = n \Rightarrow t$  by fact
show ?case
proof (cases s)
  case (Fault f)
  with exec-strip have t=Fault f
  by (auto dest: execn-Fault-end)
  with Fault show ?thesis
  by auto

```

```

next
  case Stuck
  with exec-strip have  $t = \text{Stuck}$ 
    by (auto dest: execn-Stuck-end)
  with Stuck show ?thesis
    by auto
next
  case (Abrupt  $s'$ )
  with exec-strip have  $t = \text{Abrupt } s'$ 
    by (auto dest: execn-Abrupt-end)
  with Abrupt show ?thesis
    by auto
next
  case (Normal  $s'$ ) note  $s = \text{this}$ 
  with exec-strip have
     $\Gamma \vdash \langle \text{Catch } (\text{strip-guards } F \ c1) \ (\text{strip-guards } F \ c2), \text{Normal } s' \rangle = n \Rightarrow t$  by simp
  thus ?thesis
  proof (cases)
    fix  $w$ 
    assume exec-strip-c1:  $\Gamma \vdash \langle \text{strip-guards } F \ c1, \text{Normal } s' \rangle = n \Rightarrow \text{Abrupt } w$ 
    assume exec-strip-c2:  $\Gamma \vdash \langle \text{strip-guards } F \ c2, \text{Normal } w \rangle = n \Rightarrow t$ 
    from exec-strip-c1 Catch.hyps
    obtain  $w'$  where
      exec-c1:  $\Gamma \vdash \langle c1, \text{Normal } s' \rangle = n \Rightarrow w'$  and
      w'-Fault:  $w' \in \text{Fault } (-F) \longrightarrow w' = \text{Abrupt } w$  and
      w'-noFault:  $\neg \text{isFault } w' \longrightarrow w' = \text{Abrupt } w$ 
    by blast
    show ?thesis
    proof (cases  $w'$ )
      case (Fault  $f'$ )
        with Normal exec-c1 have  $\Gamma \vdash \langle \text{Catch } c1 \ c2, s \rangle = n \Rightarrow \text{Fault } f'$ 
          by (auto intro: execn.intros)
        with w'-Fault Fault show ?thesis
          by auto
      next
        case (Stuck)
          with w'-noFault have False
            by simp
          thus ?thesis ..
    next
      case (Normal  $w''$ )
        with w'-noFault have False by simp thus ?thesis ..
    next
      case (Abrupt  $w''$ )
        with w'-noFault have  $w'' = w$  by simp
        from exec-strip-c2 Catch.hyps
        obtain  $t'$  where
           $\Gamma \vdash \langle c2, \text{Normal } w \rangle = n \Rightarrow t'$ 
          isFault  $t \longrightarrow \text{isFault } t'$ 

```

```

     $t' \in \text{Fault } '(-F) \longrightarrow t'=t$ 
     $\neg \text{isFault } t' \longrightarrow t'=t$ 
    by blast
  with  $w'' \text{ Abrupt } s \text{ exec-c1}$ 
  show ?thesis
    by (blast intro: execn.intros)
qed
next
  assume  $t: \neg \text{isAbr } t$ 
  assume  $\Gamma \vdash \langle \text{strip-guards } F \ c1, \text{Normal } s^\wedge \rangle =n \Rightarrow t$ 
  with Catch.hyps
  obtain  $t'$  where
     $\text{exec-c1}: \Gamma \vdash \langle c1, \text{Normal } s^\wedge \rangle =n \Rightarrow t'$  and
     $t\text{-Fault}: \text{isFault } t \longrightarrow \text{isFault } t'$  and
     $t'\text{-Fault}: t' \in \text{Fault } '(-F) \longrightarrow t'=t$  and
     $t'\text{-noFault}: \neg \text{isFault } t' \longrightarrow t'=t$ 
    by blast
  show ?thesis
  proof (cases isFault t')
    case True
      then obtain  $f'$  where  $t': t'= \text{Fault } f'..$ 
      with  $\text{exec-c1}$  have  $\Gamma \vdash \langle \text{Catch } c1 \ c2, \text{Normal } s^\wedge \rangle =n \Rightarrow \text{Fault } f'$ 
        by (auto intro: execn.intros)
      with  $t'\text{-Fault } t' \ s$  show ?thesis
        by auto
    next
      case False
      with  $t'\text{-noFault}$  have  $t'=t$  by simp
      with  $t \ \text{exec-c1} \ s$  show ?thesis
        by (blast intro: execn.intros)
  qed
qed
qed
qed

```

```

lemma execn-strip-to-execn:
  assumes  $\text{exec-strip}: \text{strip } F \ \Gamma \vdash \langle c, s \rangle =n \Rightarrow t$ 
  shows  $\exists t'. \Gamma \vdash \langle c, s \rangle =n \Rightarrow t' \wedge$ 
     $(\text{isFault } t \longrightarrow \text{isFault } t') \wedge$ 
     $(t' \in \text{Fault } '(-F) \longrightarrow t'=t) \wedge$ 
     $(\neg \text{isFault } t' \longrightarrow t'=t)$ 
using exec-strip
proof (induct)
  case Skip thus ?case by (blast intro: execn.intros)
next
  case Guard thus ?case by (blast intro: execn.intros)
next
  case GuardFault thus ?case by (blast intro: execn.intros)

```

```

next
  case FaultProp thus ?case by (blast intro: execn.intros)
next
  case Basic thus ?case by (blast intro: execn.intros)
next
  case Spec thus ?case by (blast intro: execn.intros)
next
  case SpecStuck thus ?case by (blast intro: execn.intros)
next
  case Seq thus ?case by (blast intro: execn.intros elim: isFaultE)
next
  case CondTrue thus ?case by (blast intro: execn.intros)
next
  case CondFalse thus ?case by (blast intro: execn.intros)
next
  case WhileTrue thus ?case by (blast intro: execn.intros elim: isFaultE)
next
  case WhileFalse thus ?case by (blast intro: execn.intros)
next
  case Call thus ?case
    by simp (blast intro: execn.intros dest: execn-strip-guards-to-execn)
next
  case CallUndefined thus ?case
    by simp (blast intro: execn.intros)
next
  case StuckProp thus ?case
    by blast
next
  case DynCom thus ?case by (blast intro: execn.intros)
next
  case Throw thus ?case by (blast intro: execn.intros)
next
  case AbruptProp thus ?case by (blast intro: execn.intros)
next
  case (CatchMatch c1 s n r c2 t)
  then obtain r' t' where
    exec-c1:  $\Gamma \vdash \langle c1, Normal\ s \rangle =n \Rightarrow r'$  and
    r'-Fault:  $r' \in Fault\ '(-F) \longrightarrow r' = Abrupt\ r$  and
    r'-noFault:  $\neg isFault\ r' \longrightarrow r' = Abrupt\ r$  and
    exec-c2:  $\Gamma \vdash \langle c2, Normal\ r \rangle =n \Rightarrow t'$  and
    t'-Fault:  $isFault\ t \longrightarrow isFault\ t'$  and
    t'-Fault:  $t' \in Fault\ '(-F) \longrightarrow t' = t$  and
    t'-noFault:  $\neg isFault\ t' \longrightarrow t' = t$ 
    by blast
  show ?case
  proof (cases isFault r')
    case True
      then obtain f' where r':  $r' = Fault\ f'$ ..
      with exec-c1 have  $\Gamma \vdash \langle Catch\ c1\ c2, Normal\ s \rangle =n \Rightarrow Fault\ f'$ 

```

by (auto intro: execn.intros)  
 with  $r'$   $r'$ -Fault show ?thesis  
 by (auto intro: execn.intros)  
 next  
 case False  
 with  $r'$ -noFault have  $r'=Abrupt r$  by simp  
 with exec-c1 exec-c2  $t$ -Fault  $t'$ -noFault  $t'$ -Fault  
 show ?thesis  
 by (blast intro: execn.intros)  
 qed  
 next  
 case CatchMiss thus ?case by (fastforce intro: execn.intros elim: isFaultE)  
 qed

**lemma** *exec-strip-guards-to-exec*:  
 assumes *exec-strip*:  $\Gamma \vdash \langle \text{strip-guards } F \ c, s \rangle \Rightarrow t$   
 shows  $\exists t'. \Gamma \vdash \langle c, s \rangle \Rightarrow t' \wedge$   
      $(isFault\ t \longrightarrow isFault\ t') \wedge$   
      $(t' \in Fault\ '(-F) \longrightarrow t'=t) \wedge$   
      $(\neg isFault\ t' \longrightarrow t'=t)$

**proof** –  
 from *exec-strip* obtain  $n$  where  
     *execn-strip*:  $\Gamma \vdash \langle \text{strip-guards } F \ c, s \rangle =n \Rightarrow t$   
 by (auto simp add: exec-iff-execn)  
 then obtain  $t'$  where  
      $\Gamma \vdash \langle c, s \rangle =n \Rightarrow t'$   
      $isFault\ t \longrightarrow isFault\ t'$   $t' \in Fault\ '(-F) \longrightarrow t'=t \neg isFault\ t' \longrightarrow t'=t$   
 by (blast dest: execn-strip-guards-to-execn)  
 thus ?thesis  
 by (blast intro: execn-to-exec)  
 qed

**lemma** *exec-strip-to-exec*:  
 assumes *exec-strip*:  $\text{strip } F \ \Gamma \vdash \langle c, s \rangle \Rightarrow t$   
 shows  $\exists t'. \Gamma \vdash \langle c, s \rangle \Rightarrow t' \wedge$   
      $(isFault\ t \longrightarrow isFault\ t') \wedge$   
      $(t' \in Fault\ '(-F) \longrightarrow t'=t) \wedge$   
      $(\neg isFault\ t' \longrightarrow t'=t)$

**proof** –  
 from *exec-strip* obtain  $n$  where  
     *execn-strip*:  $\text{strip } F \ \Gamma \vdash \langle c, s \rangle =n \Rightarrow t$   
 by (auto simp add: exec-iff-execn)  
 then obtain  $t'$  where  
      $\Gamma \vdash \langle c, s \rangle =n \Rightarrow t'$   
      $isFault\ t \longrightarrow isFault\ t'$   $t' \in Fault\ '(-F) \longrightarrow t'=t \neg isFault\ t' \longrightarrow t'=t$   
 by (blast dest: execn-strip-to-execn)  
 thus ?thesis  
 by (blast intro: execn-to-exec)  
 qed

**lemma** *exec-to-exec-strip-guards*:  
**assumes** *exec-c*:  $\Gamma \vdash \langle c, s \rangle \Rightarrow t$   
**assumes** *t-not-Fault*:  $\neg \text{isFault } t$   
**shows**  $\Gamma \vdash \langle \text{strip-guards } F \ c, s \rangle \Rightarrow t$   
**proof** –  
**from** *exec-c* **obtain** *n* **where**  $\Gamma \vdash \langle c, s \rangle = n \Rightarrow t$   
**by** (*auto simp add: exec-iff-execn*)  
**from** *this t-not-Fault*  
**have**  $\Gamma \vdash \langle \text{strip-guards } F \ c, s \rangle = n \Rightarrow t$   
**by** (*rule execn-to-execn-strip-guards*)  
**thus**  $\Gamma \vdash \langle \text{strip-guards } F \ c, s \rangle \Rightarrow t$   
**by** (*rule execn-to-exec*)  
**qed**

**lemma** *exec-to-exec-strip-guards'*:  
**assumes** *exec-c*:  $\Gamma \vdash \langle c, s \rangle \Rightarrow t$   
**assumes** *t-not-Fault*:  $t \notin \text{Fault } F$   
**shows**  $\Gamma \vdash \langle \text{strip-guards } F \ c, s \rangle \Rightarrow t$   
**proof** –  
**from** *exec-c* **obtain** *n* **where**  $\Gamma \vdash \langle c, s \rangle = n \Rightarrow t$   
**by** (*auto simp add: exec-iff-execn*)  
**from** *this t-not-Fault*  
**have**  $\Gamma \vdash \langle \text{strip-guards } F \ c, s \rangle = n \Rightarrow t$   
**by** (*rule execn-to-execn-strip-guards'*)  
**thus**  $\Gamma \vdash \langle \text{strip-guards } F \ c, s \rangle \Rightarrow t$   
**by** (*rule execn-to-exec*)  
**qed**

**lemma** *execn-to-execn-strip*:  
**assumes** *exec-c*:  $\Gamma \vdash \langle c, s \rangle = n \Rightarrow t$   
**assumes** *t-not-Fault*:  $\neg \text{isFault } t$   
**shows** *strip F*  $\Gamma \vdash \langle c, s \rangle = n \Rightarrow t$   
**using** *exec-c t-not-Fault*  
**proof** (*induct*)  
**case** (*Call p bdy s n s'*)  
**have** *bdy*:  $\Gamma \vdash p = \text{Some } \text{bdy}$  **by** *fact*  
**from** *Call* **have** *strip F*  $\Gamma \vdash \langle \text{bdy}, \text{Normal } s \rangle = n \Rightarrow s'$   
**by** *blast*  
**from** *execn-to-execn-strip-guards* [*OF this*] *Call*  
**have** *strip F*  $\Gamma \vdash \langle \text{strip-guards } F \ \text{bdy}, \text{Normal } s \rangle = n \Rightarrow s'$   
**by** *simp*  
**moreover** **from** *bdy* **have**  $(\text{strip } F \ \Gamma) \ p = \text{Some } (\text{strip-guards } F \ \text{bdy})$   
**by** *simp*  
**ultimately**  
**show** *?case*  
**by** (*blast intro: execn.intros*)  
**next**

**case** *CallUndefined* **thus** ?*case* **by** (*auto intro: execn.CallUndefined*)  
**qed** (*auto intro: execn.intros dest: noFaultn-startD' simp add: not-isFault-iff*)

**lemma** *execn-to-execn-strip'*:

**assumes** *exec-c*:  $\Gamma \vdash \langle c, s \rangle = n \Rightarrow t$

**assumes** *t-not-Fault*:  $t \notin \text{Fault } F$

**shows** *strip F*  $\Gamma \vdash \langle c, s \rangle = n \Rightarrow t$

**using** *exec-c t-not-Fault*

**proof** (*induct*)

**case** (*Call p bdy s n s'*)

**have** *bdy*:  $\Gamma \vdash p = \text{Some } \text{bdy}$  **by** *fact*

**from** *Call* **have** *strip F*  $\Gamma \vdash \langle \text{bdy}, \text{Normal } s \rangle = n \Rightarrow s'$

**by** *blast*

**from** *execn-to-execn-strip-guards'* [*OF this*] *Call*

**have** *strip F*  $\Gamma \vdash \langle \text{strip-guards } F \text{ bdy}, \text{Normal } s \rangle = n \Rightarrow s'$

**by** *simp*

**moreover from** *bdy* **have** (*strip F*  $\Gamma$ )  $p = \text{Some } (\text{strip-guards } F \text{ bdy})$

**by** *simp*

**ultimately**

**show** ?*case*

**by** (*blast intro: execn.intros*)

**next**

**case** *CallUndefined* **thus** ?*case* **by** (*auto intro: execn.CallUndefined*)

**next**

**case** (*Seq c1 s n s' c2 t*)

**show** ?*case*

**proof** (*cases isFault s'*)

**case** *False*

**with** *Seq* **show** ?*thesis*

**by** (*auto intro: execn.intros simp add: not-isFault-iff*)

**next**

**case** *True*

**then obtain** *f'* **where** *s'*:  $s' = \text{Fault } f'$  **by** (*auto simp add: isFault-def*)

**with** *Seq* **obtain**  $t = \text{Fault } f'$  **and**  $f' \notin F$

**by** (*force dest: execn-Fault-end*)

**with** *Seq s'* **show** ?*thesis*

**by** (*auto intro: execn.intros*)

**qed**

**next**

**case** (*WhileTrue b c s n s' t*)

**show** ?*case*

**proof** (*cases isFault s'*)

**case** *False*

**with** *WhileTrue* **show** ?*thesis*

**by** (*auto intro: execn.intros simp add: not-isFault-iff*)

**next**

**case** *True*

**then obtain** *f'* **where** *s'*:  $s' = \text{Fault } f'$  **by** (*auto simp add: isFault-def*)

**with** *WhileTrue* **obtain**  $t = \text{Fault } f'$  **and**  $f' \notin F$

by (force dest: execn-Fault-end)  
 with WhileTrue s' show ?thesis  
 by (auto intro: execn.intros)  
 qed  
 qed (auto intro: execn.intros)

**lemma** *exec-to-exec-strip*:  
 assumes *exec-c*:  $\Gamma \vdash \langle c, s \rangle \Rightarrow t$   
 assumes *t-not-Fault*:  $\neg \text{isFault } t$   
 shows *strip F*  $\Gamma \vdash \langle c, s \rangle \Rightarrow t$   
**proof** –  
 from *exec-c* obtain *n* where  $\Gamma \vdash \langle c, s \rangle = n \Rightarrow t$   
 by (auto simp add: exec-iff-execn)  
 from *this t-not-Fault*  
 have *strip F*  $\Gamma \vdash \langle c, s \rangle = n \Rightarrow t$   
 by (rule execn-to-execn-strip)  
 thus *strip F*  $\Gamma \vdash \langle c, s \rangle \Rightarrow t$   
 by (rule execn-to-exec)  
 qed

**lemma** *exec-to-exec-strip'*:  
 assumes *exec-c*:  $\Gamma \vdash \langle c, s \rangle \Rightarrow t$   
 assumes *t-not-Fault*:  $t \notin \text{Fault } F$   
 shows *strip F*  $\Gamma \vdash \langle c, s \rangle \Rightarrow t$   
**proof** –  
 from *exec-c* obtain *n* where  $\Gamma \vdash \langle c, s \rangle = n \Rightarrow t$   
 by (auto simp add: exec-iff-execn)  
 from *this t-not-Fault*  
 have *strip F*  $\Gamma \vdash \langle c, s \rangle = n \Rightarrow t$   
 by (rule execn-to-execn-strip')  
 thus *strip F*  $\Gamma \vdash \langle c, s \rangle \Rightarrow t$   
 by (rule execn-to-exec)  
 qed

**lemma** *exec-to-exec-strip-guards-Fault*:  
 assumes *exec-c*:  $\Gamma \vdash \langle c, s \rangle \Rightarrow \text{Fault } f$   
 assumes *f-notin-F*:  $f \notin F$   
 shows  $\Gamma \vdash \langle \text{strip-guards } F \ c, s \rangle \Rightarrow \text{Fault } f$   
**proof** –  
 from *exec-c* obtain *n* where  $\Gamma \vdash \langle c, s \rangle = n \Rightarrow \text{Fault } f$   
 by (auto simp add: exec-iff-execn)  
 from *execn-to-execn-strip-guards-Fault* [OF *this* - *f-notin-F*]  
 have  $\Gamma \vdash \langle \text{strip-guards } F \ c, s \rangle = n \Rightarrow \text{Fault } f$   
 by simp  
 thus  $\Gamma \vdash \langle \text{strip-guards } F \ c, s \rangle \Rightarrow \text{Fault } f$   
 by (rule execn-to-exec)  
 qed



### 3.8 Lemmas about $c_1 \cap_g c_2$

**lemma** *inter-guards-execn-Normal-noFault*:

$$\bigwedge c\ c2\ s\ t\ n. \llbracket (c1 \cap_g c2) = \text{Some } c; \Gamma \vdash \langle c, \text{Normal } s \rangle = n \Rightarrow t; \neg \text{isFault } t \rrbracket \\ \Rightarrow \Gamma \vdash \langle c1, \text{Normal } s \rangle = n \Rightarrow t \wedge \Gamma \vdash \langle c2, \text{Normal } s \rangle = n \Rightarrow t$$

**proof** (*induct c1*)

**case** *Skip*

**have**  $(\text{Skip} \cap_g c2) = \text{Some } c$  **by fact**

**then obtain**  $c2: c2 = \text{Skip}$  **and**  $c: c = \text{Skip}$

**by** (*simp add: inter-guards-Skip*)

**have**  $\Gamma \vdash \langle c, \text{Normal } s \rangle = n \Rightarrow t$  **by fact**

**with**  $c$  **have**  $t = \text{Normal } s$

**by** (*auto elim: execn-Normal-elim-cases*)

**with** *Skip c2*

**show** *?case*

**by** (*auto intro: execn.intros*)

**next**

**case** (*Basic f*)

**have**  $(\text{Basic } f \cap_g c2) = \text{Some } c$  **by fact**

**then obtain**  $c2: c2 = \text{Basic } f$  **and**  $c: c = \text{Basic } f$

**by** (*simp add: inter-guards-Basic*)

**have**  $\Gamma \vdash \langle c, \text{Normal } s \rangle = n \Rightarrow t$  **by fact**

**with**  $c$  **have**  $t = \text{Normal } (f\ s)$

**by** (*auto elim: execn-Normal-elim-cases*)

**with** *Basic c2*

**show** *?case*

**by** (*auto intro: execn.intros*)

**next**

**case** (*Spec r*)

**have**  $(\text{Spec } r \cap_g c2) = \text{Some } c$  **by fact**

**then obtain**  $c2: c2 = \text{Spec } r$  **and**  $c: c = \text{Spec } r$

**by** (*simp add: inter-guards-Spec*)

**have**  $\Gamma \vdash \langle c, \text{Normal } s \rangle = n \Rightarrow t$  **by fact**

**with**  $c$  **have**  $\Gamma \vdash \langle \text{Spec } r, \text{Normal } s \rangle = n \Rightarrow t$  **by simp**

**from** *this Spec c2* **show** *?case*

**by** (*cases*) (*auto intro: execn.intros*)

**next**

**case** (*Seq a1 a2*)

**have** *noFault*:  $\neg \text{isFault } t$  **by fact**

**have**  $(\text{Seq } a1\ a2 \cap_g c2) = \text{Some } c$  **by fact**

**then obtain**  $b1\ b2\ d1\ d2$  **where**

$c2: c2 = \text{Seq } b1\ b2$  **and**

$d1: (a1 \cap_g b1) = \text{Some } d1$  **and**  $d2: (a2 \cap_g b2) = \text{Some } d2$  **and**

$c: c = \text{Seq } d1\ d2$

**by** (*auto simp add: inter-guards-Seq*)

**have**  $\Gamma \vdash \langle c, \text{Normal } s \rangle = n \Rightarrow t$  **by fact**

**with**  $c$  **obtain**  $s'$  **where**

*exec-d1*:  $\Gamma \vdash \langle d1, \text{Normal } s \rangle = n \Rightarrow s'$  **and**

*exec-d2*:  $\Gamma \vdash \langle d2, s' \rangle = n \Rightarrow t$

**by** (*auto elim: execn-Normal-elim-cases*)

```

show ?case
proof (cases s')
  case (Fault f')
    with exec-d2 have t=Fault f'
    by (auto intro: execn-Fault-end)
    with noFault show ?thesis by simp
next
  case (Normal s'')
    with d1 exec-d1 Seq.hyps
    obtain
       $\Gamma \vdash \langle a1, Normal\ s \rangle =n \Rightarrow Normal\ s''$  and  $\Gamma \vdash \langle b1, Normal\ s \rangle =n \Rightarrow Normal\ s''$ 
    by auto
    moreover
      from Normal d2 exec-d2 noFault Seq.hyps
    obtain  $\Gamma \vdash \langle a2, Normal\ s'' \rangle =n \Rightarrow t$  and  $\Gamma \vdash \langle b2, Normal\ s'' \rangle =n \Rightarrow t$ 
    by auto
    ultimately
    show ?thesis
      using Normal c2 by (auto intro: execn.intros)
next
  case (Abrupt s'')
    with exec-d2 have t=Abrupt s''
    by (auto simp add: execn-Abrupt-end)
    moreover
      from Abrupt d1 exec-d1 Seq.hyps
    obtain  $\Gamma \vdash \langle a1, Normal\ s \rangle =n \Rightarrow Abrupt\ s''$  and  $\Gamma \vdash \langle b1, Normal\ s \rangle =n \Rightarrow Abrupt$ 
s''
    by auto
    moreover
    obtain
       $\Gamma \vdash \langle a2, Abrupt\ s'' \rangle =n \Rightarrow Abrupt\ s''$  and  $\Gamma \vdash \langle b2, Abrupt\ s'' \rangle =n \Rightarrow Abrupt\ s''$ 
    by auto
    ultimately
    show ?thesis
      using Abrupt c2 by (auto intro: execn.intros)
next
  case Stuck
    with exec-d2 have t=Stuck
    by (auto simp add: execn-Stuck-end)
    moreover
      from Stuck d1 exec-d1 Seq.hyps
    obtain  $\Gamma \vdash \langle a1, Normal\ s \rangle =n \Rightarrow Stuck$  and  $\Gamma \vdash \langle b1, Normal\ s \rangle =n \Rightarrow Stuck$ 
    by auto
    moreover
    obtain
       $\Gamma \vdash \langle a2, Stuck \rangle =n \Rightarrow Stuck$  and  $\Gamma \vdash \langle b2, Stuck \rangle =n \Rightarrow Stuck$ 
    by auto
    ultimately
    show ?thesis

```

```

    using Stuck c2 by (auto intro: execn.intros)
  qed
next
case (Cond b t1 e1)
have noFault:  $\neg$  isFault t by fact
have (Cond b t1 e1  $\cap_g$  c2) = Some c by fact
then obtain t2 e2 t3 e3 where
  c2: c2=Cond b t2 e2 and
  t3: (t1  $\cap_g$  t2) = Some t3 and
  e3: (e1  $\cap_g$  e2) = Some e3 and
  c: c=Cond b t3 e3
  by (auto simp add: inter-guards-Cond)
have  $\Gamma \vdash \langle c, \text{Normal } s \rangle = n \Rightarrow t$  by fact
with c have  $\Gamma \vdash \langle \text{Cond } b \text{ t3 e3}, \text{Normal } s \rangle = n \Rightarrow t$ 
  by simp
then show ?case
proof (cases)
  assume s-in-b:  $s \in b$ 
  assume  $\Gamma \vdash \langle t3, \text{Normal } s \rangle = n \Rightarrow t$ 
  with Cond.hyps t3 noFault
  obtain  $\Gamma \vdash \langle t1, \text{Normal } s \rangle = n \Rightarrow t$   $\Gamma \vdash \langle t2, \text{Normal } s \rangle = n \Rightarrow t$ 
    by auto
  with s-in-b c2 show ?thesis
    by (auto intro: execn.intros)
next
  assume s-notin-b:  $s \notin b$ 
  assume  $\Gamma \vdash \langle e3, \text{Normal } s \rangle = n \Rightarrow t$ 
  with Cond.hyps e3 noFault
  obtain  $\Gamma \vdash \langle e1, \text{Normal } s \rangle = n \Rightarrow t$   $\Gamma \vdash \langle e2, \text{Normal } s \rangle = n \Rightarrow t$ 
    by auto
  with s-notin-b c2 show ?thesis
    by (auto intro: execn.intros)
qed
next
case (While b bdy1)
have noFault:  $\neg$  isFault t by fact
have (While b bdy1  $\cap_g$  c2) = Some c by fact
then obtain bdy2 bdy where
  c2: c2=While b bdy2 and
  bdy: (bdy1  $\cap_g$  bdy2) = Some bdy and
  c: c=While b bdy
  by (auto simp add: inter-guards-While)
have exec-c:  $\Gamma \vdash \langle c, \text{Normal } s \rangle = n \Rightarrow t$  by fact
{
  fix s t n w w1 w2
  assume exec-w:  $\Gamma \vdash \langle w, \text{Normal } s \rangle = n \Rightarrow t$ 
  assume w: w=While b bdy
  assume noFault:  $\neg$  isFault t
  from exec-w w noFault

```

```

have  $\Gamma \vdash \langle \text{While } b \text{ bdy1}, \text{Normal } s \rangle =n \Rightarrow t \wedge$ 
   $\Gamma \vdash \langle \text{While } b \text{ bdy2}, \text{Normal } s \rangle =n \Rightarrow t$ 
proof (induct)
  prefer 10
  case (WhileTrue s b' bdy' n s' s'')
  have eqs:  $\text{While } b' \text{ bdy}' = \text{While } b \text{ bdy}$  by fact
  from WhileTrue have s-in-b:  $s \in b$  by simp
  have noFault-s'':  $\neg \text{isFault } s''$  by fact
  from WhileTrue
  have exec-bdy:  $\Gamma \vdash \langle \text{bdy}, \text{Normal } s \rangle =n \Rightarrow s'$  by simp
  from WhileTrue
  have exec-w:  $\Gamma \vdash \langle \text{While } b \text{ bdy}, s^\wedge \rangle =n \Rightarrow s''$  by simp
  show ?case
  proof (cases s^)
    case (Fault f)
    with exec-w have s''=Fault f
    by (auto intro: execn-Fault-end)
    with noFault-s'' show ?thesis by simp
  next
  case (Normal s''')
  with exec-bdy bdy While.hyps
  obtain  $\Gamma \vdash \langle \text{bdy1}, \text{Normal } s \rangle =n \Rightarrow \text{Normal } s'''$ 
     $\Gamma \vdash \langle \text{bdy2}, \text{Normal } s \rangle =n \Rightarrow \text{Normal } s'''$ 
    by auto
  moreover
  from Normal WhileTrue
  obtain
     $\Gamma \vdash \langle \text{While } b \text{ bdy1}, \text{Normal } s'' \rangle =n \Rightarrow s''$ 
     $\Gamma \vdash \langle \text{While } b \text{ bdy2}, \text{Normal } s'' \rangle =n \Rightarrow s''$ 
    by simp
  ultimately show ?thesis
    using s-in-b Normal
    by (auto intro: execn.intros)
  next
  case (Abrupt s''')
  with exec-bdy bdy While.hyps
  obtain  $\Gamma \vdash \langle \text{bdy1}, \text{Normal } s \rangle =n \Rightarrow \text{Abrupt } s'''$ 
     $\Gamma \vdash \langle \text{bdy2}, \text{Normal } s \rangle =n \Rightarrow \text{Abrupt } s'''$ 
    by auto
  moreover
  from Abrupt WhileTrue
  obtain
     $\Gamma \vdash \langle \text{While } b \text{ bdy1}, \text{Abrupt } s'' \rangle =n \Rightarrow s''$ 
     $\Gamma \vdash \langle \text{While } b \text{ bdy2}, \text{Abrupt } s'' \rangle =n \Rightarrow s''$ 
    by simp
  ultimately show ?thesis
    using s-in-b Abrupt
    by (auto intro: execn.intros)
  next

```

```

case Stuck
with exec-bdy bdy While.hyps
obtain  $\Gamma \vdash \langle bdy1, Normal\ s \rangle =n \Rightarrow Stuck$ 
       $\Gamma \vdash \langle bdy2, Normal\ s \rangle =n \Rightarrow Stuck$ 
      by auto
moreover
from Stuck WhileTrue
obtain
       $\Gamma \vdash \langle While\ b\ bdy1, Stuck \rangle =n \Rightarrow s''$ 
       $\Gamma \vdash \langle While\ b\ bdy2, Stuck \rangle =n \Rightarrow s''$ 
      by simp
ultimately show ?thesis
      using s-in-b Stuck
      by (auto intro: execn.intros)
qed
next
      case WhileFalse thus ?case by (auto intro: execn.intros)
qed (simp-all)
}
with this [OF exec-c c noFault] c2
show ?case
      by auto
next
      case Call thus ?case by (simp add: inter-guards-Call)
next
      case (DynCom f1)
      have noFault:  $\neg isFault\ t$  by fact
      have (DynCom f1  $\cap_g$  c2) = Some c by fact
      then obtain f2 f where
        c2: c2 = DynCom f2 and
        f-defined:  $\forall s. ((f1\ s) \cap_g (f2\ s)) \neq None$  and
        c: c = DynCom ( $\lambda s. the\ ((f1\ s) \cap_g (f2\ s))$ )
        by (auto simp add: inter-guards-DynCom)
      have  $\Gamma \vdash \langle c, Normal\ s \rangle =n \Rightarrow t$  by fact
      with c have  $\Gamma \vdash \langle DynCom\ (\lambda s. the\ ((f1\ s) \cap_g (f2\ s))), Normal\ s \rangle =n \Rightarrow t$  by simp
      then show ?case
      proof (cases)
        assume exec-f:  $\Gamma \vdash \langle the\ (f1\ s \cap_g f2\ s), Normal\ s \rangle =n \Rightarrow t$ 
        from f-defined obtain f where (f1 s  $\cap_g$  f2 s) = Some f
        by auto
        with DynCom.hyps this exec-f c2 noFault
        show ?thesis
        using execn.DynCom by fastforce
      qed
next
      case Guard thus ?case
      by (fastforce elim: execn-Normal-elim-cases intro: execn.intros
        simp add: inter-guards-Guard)
next

```

```

case Throw thus ?case
  by (fastforce elim: execn-Normal-elim-cases
      simp add: inter-guards-Throw)
next
case (Catch a1 a2)
have noFault: ¬ isFault t by fact
have (Catch a1 a2  $\cap_g$  c2) = Some c by fact
then obtain b1 b2 d1 d2 where
  c2: c2 = Catch b1 b2 and
  d1: (a1  $\cap_g$  b1) = Some d1 and d2: (a2  $\cap_g$  b2) = Some d2 and
  c: c = Catch d1 d2
  by (auto simp add: inter-guards-Catch)
have  $\Gamma \vdash \langle c, \text{Normal } s \rangle = n \Rightarrow t$  by fact
with c have  $\Gamma \vdash \langle \text{Catch } d1 \ d2, \text{Normal } s \rangle = n \Rightarrow t$  by simp
then show ?case
proof (cases)
  fix s'
  assume  $\Gamma \vdash \langle d1, \text{Normal } s \rangle = n \Rightarrow \text{Abrupt } s'$ 
  with d1 Catch.hyps
  obtain  $\Gamma \vdash \langle a1, \text{Normal } s \rangle = n \Rightarrow \text{Abrupt } s'$  and  $\Gamma \vdash \langle b1, \text{Normal } s \rangle = n \Rightarrow \text{Abrupt } s'$ 
    by auto
  moreover
  assume  $\Gamma \vdash \langle d2, \text{Normal } s' \rangle = n \Rightarrow t$ 
  with d2 Catch.hyps noFault
  obtain  $\Gamma \vdash \langle a2, \text{Normal } s' \rangle = n \Rightarrow t$  and  $\Gamma \vdash \langle b2, \text{Normal } s' \rangle = n \Rightarrow t$ 
    by auto
  ultimately
  show ?thesis
    using c2 by (auto intro: execn.intros)
next
  assume  $\neg \text{isAbr } t$ 
  moreover
  assume  $\Gamma \vdash \langle d1, \text{Normal } s \rangle = n \Rightarrow t$ 
  with d1 Catch.hyps noFault
  obtain  $\Gamma \vdash \langle a1, \text{Normal } s \rangle = n \Rightarrow t$  and  $\Gamma \vdash \langle b1, \text{Normal } s \rangle = n \Rightarrow t$ 
    by auto
  ultimately
  show ?thesis
    using c2 by (auto intro: execn.intros)
qed
qed

```

```

lemma inter-guards-execn-noFault:
  assumes c: (c1  $\cap_g$  c2) = Some c
  assumes exec-c:  $\Gamma \vdash \langle c, s \rangle = n \Rightarrow t$ 
  assumes noFault: ¬ isFault t
  shows  $\Gamma \vdash \langle c1, s \rangle = n \Rightarrow t \wedge \Gamma \vdash \langle c2, s \rangle = n \Rightarrow t$ 

```

```

proof (cases s)
  case (Fault f)
    with exec-c have t = Fault f
      by (auto intro: execn-Fault-end)
      with noFault show ?thesis
      by simp
  next
    case (Abrupt s')
      with exec-c have t=Abrupt s'
        by (simp add: execn-Abrupt-end)
      with Abrupt show ?thesis by auto
  next
    case Stuck
      with exec-c have t=Stuck
        by (simp add: execn-Stuck-end)
      with Stuck show ?thesis by auto
  next
    case (Normal s')
      with exec-c noFault inter-guards-execn-Normal-noFault [OF c]
      show ?thesis
      by blast
qed

```

```

lemma inter-guards-exec-noFault:
  assumes c: (c1  $\cap_g$  c2) = Some c
  assumes exec-c:  $\Gamma \vdash \langle c, s \rangle \Rightarrow t$ 
  assumes noFault:  $\neg$  isFault t
  shows  $\Gamma \vdash \langle c1, s \rangle \Rightarrow t \wedge \Gamma \vdash \langle c2, s \rangle \Rightarrow t$ 
proof -
  from exec-c obtain n where  $\Gamma \vdash \langle c, s \rangle = n \Rightarrow t$ 
    by (auto simp add: exec-iff-execn)
  from c this noFault
  have  $\Gamma \vdash \langle c1, s \rangle = n \Rightarrow t \wedge \Gamma \vdash \langle c2, s \rangle = n \Rightarrow t$ 
    by (rule inter-guards-execn-noFault)
  thus ?thesis
    by (auto intro: execn-to-exec)
qed

```

```

lemma inter-guards-execn-Normal-Fault:
   $\bigwedge c c2 s n. \llbracket (c1 \cap_g c2) = \text{Some } c; \Gamma \vdash \langle c, \text{Normal } s \rangle = n \Rightarrow \text{Fault } f \rrbracket$ 
     $\implies (\Gamma \vdash \langle c1, \text{Normal } s \rangle = n \Rightarrow \text{Fault } f \vee \Gamma \vdash \langle c2, \text{Normal } s \rangle = n \Rightarrow \text{Fault } f)$ 
proof (induct c1)
  case Skip thus ?case by (fastforce simp add: inter-guards-Skip)
next
  case (Basic f) thus ?case by (fastforce simp add: inter-guards-Basic)
next
  case (Spec r) thus ?case by (fastforce simp add: inter-guards-Spec)
next

```

**case** (*Seq a1 a2*)  
**have** (*Seq a1 a2*  $\cap_g$  *c2*) = *Some c* **by fact**  
**then obtain** *b1 b2 d1 d2* **where**  
*c2*: *c2=Seq b1 b2* **and**  
*d1*: (*a1*  $\cap_g$  *b1*) = *Some d1* **and** *d2*: (*a2*  $\cap_g$  *b2*) = *Some d2* **and**  
*c*: *c=Seq d1 d2*  
**by** (*auto simp add: inter-guards-Seq*)  
**have**  $\Gamma \vdash \langle c, \text{Normal } s \rangle = n \Rightarrow \text{Fault } f$  **by fact**  
**with** *c* **obtain** *s'* **where**  
*exec-d1*:  $\Gamma \vdash \langle d1, \text{Normal } s \rangle = n \Rightarrow s'$  **and**  
*exec-d2*:  $\Gamma \vdash \langle d2, s' \rangle = n \Rightarrow \text{Fault } f$   
**by** (*auto elim: execn-Normal-elim-cases*)  
**show** *?case*  
**proof** (*cases s'*)  
**case** (*Fault f'*)  
**with** *exec-d2* **have** *f'=f*  
**by** (*auto dest: execn-Fault-end*)  
**with** *Fault d1 exec-d1*  
**have**  $\Gamma \vdash \langle a1, \text{Normal } s \rangle = n \Rightarrow \text{Fault } f \vee \Gamma \vdash \langle b1, \text{Normal } s \rangle = n \Rightarrow \text{Fault } f$   
**by** (*auto dest: Seq.hyps*)  
**thus** *?thesis*  
**proof** (*cases rule: disjE [consumes 1]*)  
**assume**  $\Gamma \vdash \langle a1, \text{Normal } s \rangle = n \Rightarrow \text{Fault } f$   
**hence**  $\Gamma \vdash \langle \text{Seq } a1 \ a2, \text{Normal } s \rangle = n \Rightarrow \text{Fault } f$   
**by** (*auto intro: execn.intros*)  
**thus** *?thesis*  
**by** *simp*  
**next**  
**assume**  $\Gamma \vdash \langle b1, \text{Normal } s \rangle = n \Rightarrow \text{Fault } f$   
**hence**  $\Gamma \vdash \langle \text{Seq } b1 \ b2, \text{Normal } s \rangle = n \Rightarrow \text{Fault } f$   
**by** (*auto intro: execn.intros*)  
**with** *c2* **show** *?thesis*  
**by** *simp*  
**qed**  
**next**  
**case** *Abrupt* **with** *exec-d2* **show** *?thesis* **by** (*auto dest: execn-Abrupt-end*)  
**next**  
**case** *Stuck* **with** *exec-d2* **show** *?thesis* **by** (*auto dest: execn-Stuck-end*)  
**next**  
**case** (*Normal s''*)  
**with** *inter-guards-execn-noFault [OF d1 exec-d1]* **obtain**  
*exec-a1*:  $\Gamma \vdash \langle a1, \text{Normal } s \rangle = n \Rightarrow \text{Normal } s''$  **and**  
*exec-b1*:  $\Gamma \vdash \langle b1, \text{Normal } s \rangle = n \Rightarrow \text{Normal } s''$   
**by** *simp*  
**moreover from** *d2 exec-d2 Normal*  
**have**  $\Gamma \vdash \langle a2, \text{Normal } s' \rangle = n \Rightarrow \text{Fault } f \vee \Gamma \vdash \langle b2, \text{Normal } s' \rangle = n \Rightarrow \text{Fault } f$   
**by** (*auto dest: Seq.hyps*)  
**ultimately show** *?thesis*  
**using** *c2* **by** (*auto intro: execn.intros*)



```

qed
next
case (Cond b t1 e1)
have (Cond b t1 e1  $\cap_g$  c2) = Some c by fact
then obtain t2 e2 t e where
  c2: c2=Cond b t2 e2 and
  t: (t1  $\cap_g$  t2) = Some t and
  e: (e1  $\cap_g$  e2) = Some e and
  c: c=Cond b t e
  by (auto simp add: inter-guards-Cond)
have  $\Gamma \vdash \langle c, Normal\ s \rangle = n \Rightarrow Fault\ f$  by fact
with c have  $\Gamma \vdash \langle Cond\ b\ t\ e, Normal\ s \rangle = n \Rightarrow Fault\ f$  by simp
thus ?case
proof (cases)
  assume s  $\in$  b
  moreover assume  $\Gamma \vdash \langle t, Normal\ s \rangle = n \Rightarrow Fault\ f$ 
  with t have  $\Gamma \vdash \langle t1, Normal\ s \rangle = n \Rightarrow Fault\ f \vee \Gamma \vdash \langle t2, Normal\ s \rangle = n \Rightarrow Fault\ f$ 
    by (auto dest: Cond.hyps)
  ultimately show ?thesis using c2 c by (fastforce intro: execn.intros)
next
  assume s  $\notin$  b
  moreover assume  $\Gamma \vdash \langle e, Normal\ s \rangle = n \Rightarrow Fault\ f$ 
  with e have  $\Gamma \vdash \langle e1, Normal\ s \rangle = n \Rightarrow Fault\ f \vee \Gamma \vdash \langle e2, Normal\ s \rangle = n \Rightarrow Fault\ f$ 
    by (auto dest: Cond.hyps)
  ultimately show ?thesis using c2 c by (fastforce intro: execn.intros)
qed
next
case (While b bdy1)
have (While b bdy1  $\cap_g$  c2) = Some c by fact
then obtain bdy2 bdy where
  c2: c2=While b bdy2 and
  bdy: (bdy1  $\cap_g$  bdy2) = Some bdy and
  c: c=While b bdy
  by (auto simp add: inter-guards-While)
have exec-c:  $\Gamma \vdash \langle c, Normal\ s \rangle = n \Rightarrow Fault\ f$  by fact
{
  fix s t n w w1 w2
  assume exec-w:  $\Gamma \vdash \langle w, Normal\ s \rangle = n \Rightarrow t$ 
  assume w: w=While b bdy
  assume Fault: t=Fault f
  from exec-w w Fault
  have  $\Gamma \vdash \langle While\ b\ bdy1, Normal\ s \rangle = n \Rightarrow Fault\ f \vee$ 
     $\Gamma \vdash \langle While\ b\ bdy2, Normal\ s \rangle = n \Rightarrow Fault\ f$ 
  proof (induct)
    case (WhileTrue s b' bdy' n s' s'')
    have eqs: While b' bdy' = While b bdy by fact
    from WhileTrue have s-in-b: s  $\in$  b by simp
    have Fault-s'': s''=Fault f by fact
    from WhileTrue

```

```

have exec-bdy:  $\Gamma \vdash \langle bdy, Normal\ s \rangle =n \Rightarrow s'$  by simp
from WhileTrue
have exec-w:  $\Gamma \vdash \langle While\ b\ bdy, s^\wedge \rangle =n \Rightarrow s''$  by simp
show ?case
proof (cases s^\wedge)
  case (Fault f^\wedge)
    with exec-w Fault-s'' have  $f'=f$ 
      by (auto dest: execn-Fault-end)
    with Fault exec-bdy bdy While.hyps
    have  $\Gamma \vdash \langle bdy1, Normal\ s \rangle =n \Rightarrow Fault\ f \vee \Gamma \vdash \langle bdy2, Normal\ s \rangle =n \Rightarrow Fault\ f$ 
      by auto
    with s-in-b show ?thesis
      by (fastforce intro: execn.intros)
  next
    case (Normal s''')
    with inter-guards-execn-noFault [OF bdy exec-bdy]
    obtain  $\Gamma \vdash \langle bdy1, Normal\ s \rangle =n \Rightarrow Normal\ s'''$ 
       $\Gamma \vdash \langle bdy2, Normal\ s \rangle =n \Rightarrow Normal\ s'''$ 
      by auto
    moreover
    from Normal WhileTrue
    have  $\Gamma \vdash \langle While\ b\ bdy1, Normal\ s''' \rangle =n \Rightarrow Fault\ f \vee$ 
       $\Gamma \vdash \langle While\ b\ bdy2, Normal\ s''' \rangle =n \Rightarrow Fault\ f$ 
      by simp
    ultimately show ?thesis
      using s-in-b by (fastforce intro: execn.intros)
  next
    case (Abrupt s''')
    with exec-w Fault-s'' show ?thesis by (fastforce dest: execn-Abrupt-end)
  next
    case Stuck
    with exec-w Fault-s'' show ?thesis by (fastforce dest: execn-Stuck-end)
  qed
next
  case WhileFalse thus ?case by (auto intro: execn.intros)
qed (simp-all)
}
with this [OF exec-c c] c2
show ?case
  by auto
next
  case Call thus ?case by (fastforce simp add: inter-guards-Call)
next
  case (DynCom f1)
  have (DynCom f1  $\cap_g$  c2) = Some c by fact
  then obtain f2 where
    c2: c2=DynCom f2 and
    F-defined:  $\forall s. ((f1\ s) \cap_g (f2\ s)) \neq None$  and
    c: c=DynCom ( $\lambda s. the\ ((f1\ s) \cap_g (f2\ s))$ )

```

```

    by (auto simp add: inter-guards-DynCom)
  have  $\Gamma \vdash \langle c, Normal\ s \rangle = n \Rightarrow Fault\ f$  by fact
  with c have  $\Gamma \vdash \langle DynCom\ (\lambda s. the\ ((f1\ s) \cap_g (f2\ s))), Normal\ s \rangle = n \Rightarrow Fault\ f$  by
simp
  then show ?case
  proof (cases)
    assume exec-F:  $\Gamma \vdash \langle the\ (f1\ s \cap_g f2\ s), Normal\ s \rangle = n \Rightarrow Fault\ f$ 
    from F-defined obtain F where  $(f1\ s \cap_g f2\ s) = Some\ F$ 
    by auto
    with DynCom.hyps this exec-F c2
    show ?thesis
    by (fastforce intro: execn.intros)
  qed
next
  case (Guard m g1 bdy1)
  have  $(Guard\ m\ g1\ bdy1 \cap_g c2) = Some\ c$  by fact
  then obtain g2 bdy2 bdy where
    c2:  $c2 = Guard\ m\ g2\ bdy2$  and
    bdy:  $(bdy1 \cap_g bdy2) = Some\ bdy$  and
    c:  $c = Guard\ m\ (g1 \cap g2)\ bdy$ 
  by (auto simp add: inter-guards-Guard)
  have  $\Gamma \vdash \langle c, Normal\ s \rangle = n \Rightarrow Fault\ f$  by fact
  with c have  $\Gamma \vdash \langle Guard\ m\ (g1 \cap g2)\ bdy, Normal\ s \rangle = n \Rightarrow Fault\ f$ 
  by simp
  thus ?case
  proof (cases)
    assume f-m:  $Fault\ f = Fault\ m$ 
    assume  $s \notin g1 \cap g2$ 
    hence  $s \notin g1 \vee s \notin g2$ 
    by blast
    with c2 f-m show ?thesis
    by (auto intro: execn.intros)
  next
    assume  $s \in g1 \cap g2$ 
    moreover
    assume  $\Gamma \vdash \langle bdy, Normal\ s \rangle = n \Rightarrow Fault\ f$ 
    with bdy have  $\Gamma \vdash \langle bdy1, Normal\ s \rangle = n \Rightarrow Fault\ f \vee \Gamma \vdash \langle bdy2, Normal\ s \rangle = n \Rightarrow$ 
Fault f
    by (rule Guard.hyps)
    ultimately show ?thesis
    using c2
    by (auto intro: execn.intros)
  qed
next
  case Throw thus ?case by (fastforce simp add: inter-guards-Throw)
next
  case (Catch a1 a2)
  have  $(Catch\ a1\ a2 \cap_g c2) = Some\ c$  by fact
  then obtain b1 b2 d1 d2 where

```

```

    c2: c2=Catch b1 b2 and
    d1: (a1  $\cap_g$  b1) = Some d1 and d2: (a2  $\cap_g$  b2) = Some d2 and
    c: c=Catch d1 d2
  by (auto simp add: inter-guards-Catch)
have  $\Gamma \vdash \langle c, Normal\ s \rangle = n \Rightarrow Fault\ f$  by fact
with c have  $\Gamma \vdash \langle Catch\ d1\ d2, Normal\ s \rangle = n \Rightarrow Fault\ f$  by simp
thus ?case
proof (cases)
  fix s'
  assume  $\Gamma \vdash \langle d1, Normal\ s \rangle = n \Rightarrow Abrupt\ s'$ 
  from inter-guards-execn-noFault [OF d1 this] obtain
    exec-a1:  $\Gamma \vdash \langle a1, Normal\ s \rangle = n \Rightarrow Abrupt\ s'$  and
    exec-b1:  $\Gamma \vdash \langle b1, Normal\ s \rangle = n \Rightarrow Abrupt\ s'$ 
  by simp
  moreover assume  $\Gamma \vdash \langle d2, Normal\ s \rangle = n \Rightarrow Fault\ f$ 
  with d2
  have  $\Gamma \vdash \langle a2, Normal\ s \rangle = n \Rightarrow Fault\ f \vee \Gamma \vdash \langle b2, Normal\ s \rangle = n \Rightarrow Fault\ f$ 
    by (auto dest: Catch.hyps)
  ultimately show ?thesis
    using c2 by (fastforce intro: execn.intros)
next
  assume  $\Gamma \vdash \langle d1, Normal\ s \rangle = n \Rightarrow Fault\ f$ 
  with d1 have  $\Gamma \vdash \langle a1, Normal\ s \rangle = n \Rightarrow Fault\ f \vee \Gamma \vdash \langle b1, Normal\ s \rangle = n \Rightarrow Fault\ f$ 
f
  by (auto dest: Catch.hyps)
  with c2 show ?thesis
  by (fastforce intro: execn.intros)
qed
qed

```

```

lemma inter-guards-execn-Fault:
  assumes c: (c1  $\cap_g$  c2) = Some c
  assumes exec-c:  $\Gamma \vdash \langle c, s \rangle = n \Rightarrow Fault\ f$ 
  shows  $\Gamma \vdash \langle c1, s \rangle = n \Rightarrow Fault\ f \vee \Gamma \vdash \langle c2, s \rangle = n \Rightarrow Fault\ f$ 
proof (cases s)
  case (Fault f)
  with exec-c show ?thesis
  by (auto dest: execn-Fault-end)
next
  case (Abrupt s')
  with exec-c show ?thesis
  by (fastforce dest: execn-Abrupt-end)
next
  case Stuck
  with exec-c show ?thesis
  by (fastforce dest: execn-Stuck-end)
next
  case (Normal s')

```

**with** *exec-c inter-guards-execn-Normal-Fault* [*OF c*]  
**show** *?thesis*  
**by** *blast*  
**qed**

**lemma** *inter-guards-exec-Fault*:  
**assumes** *c: (c1  $\cap_g$  c2) = Some c*  
**assumes** *exec-c:  $\Gamma \vdash \langle c, s \rangle \Rightarrow \text{Fault } f$*   
**shows**  *$\Gamma \vdash \langle c1, s \rangle \Rightarrow \text{Fault } f \vee \Gamma \vdash \langle c2, s \rangle \Rightarrow \text{Fault } f$*   
**proof** –  
**from** *exec-c obtain n where  $\Gamma \vdash \langle c, s \rangle = n \Rightarrow \text{Fault } f$*   
**by** (*auto simp add: exec-iff-execn*)  
**from** *c this*  
**have**  *$\Gamma \vdash \langle c1, s \rangle = n \Rightarrow \text{Fault } f \vee \Gamma \vdash \langle c2, s \rangle = n \Rightarrow \text{Fault } f$*   
**by** (*rule inter-guards-execn-Fault*)  
**thus** *?thesis*  
**by** (*auto intro: execn-to-exec*)  
**qed**

### 3.9 Restriction of Procedure Environment

**lemma** *restrict-SomeD:  $(m|_A) x = \text{Some } y \implies m x = \text{Some } y$*   
**by** (*auto simp add: restrict-map-def split: if-split-asm*)

**lemma** *restrict-dom-same [simp]:  $m|_{\text{dom } m} = m$*   
**apply** (*rule ext*)  
**apply** (*clarsimp simp add: restrict-map-def*)  
**apply** (*simp only: not-None-eq [symmetric]*)  
**apply** *rule*  
**apply** (*drule sym*)  
**apply** *blast*  
**done**

**lemma** *restrict-in-dom:  $x \in A \implies (m|_A) x = m x$*   
**by** (*auto simp add: restrict-map-def*)

**lemma** *exec-restrict-to-exec*:  
**assumes** *exec-restrict:  $\Gamma|_A \vdash \langle c, s \rangle \Rightarrow t$*   
**assumes** *notStuck:  $t \neq \text{Stuck}$*   
**shows**  *$\Gamma \vdash \langle c, s \rangle \Rightarrow t$*   
**using** *exec-restrict notStuck*  
**by** (*induct*) (*auto intro: exec.intros dest: restrict-SomeD Stuck-end*)

**lemma** *execn-restrict-to-execn*:  
**assumes** *exec-restrict:  $\Gamma|_A \vdash \langle c, s \rangle = n \Rightarrow t$*   
**assumes** *notStuck:  $t \neq \text{Stuck}$*   
**shows**  *$\Gamma \vdash \langle c, s \rangle = n \Rightarrow t$*

```

using exec-restrict notStuck
by (induct) (auto intro: execn.intros dest: restrict-SomeD execn-Stuck-end)

lemma restrict-NoneD:  $m\ x = \text{None} \implies (m|_A)\ x = \text{None}$ 
  by (auto simp add: restrict-map-def split: if-split-asm)

lemma execn-to-execn-restrict:
  assumes execn:  $\Gamma \vdash \langle c, s \rangle = n \Rightarrow t$ 
  shows  $\exists t'. \Gamma |_{P} \vdash \langle c, s \rangle = n \Rightarrow t' \wedge (t = \text{Stuck} \longrightarrow t' = \text{Stuck}) \wedge$ 
     $(\forall f. t = \text{Fault } f \longrightarrow t' \in \{\text{Fault } f, \text{Stuck}\}) \wedge (t' \neq \text{Stuck} \longrightarrow t' = t)$ 
using execn
proof (induct)
  case Skip show ?case by (blast intro: execn.Skip)
next
  case Guard thus ?case by (auto intro: execn.Guard)
next
  case GuardFault thus ?case by (auto intro: execn.GuardFault)
next
  case FaultProp thus ?case by (auto intro: execn.FaultProp)
next
  case Basic thus ?case by (auto intro: execn.Basic)
next
  case Spec thus ?case by (auto intro: execn.Spec)
next
  case SpecStuck thus ?case by (auto intro: execn.SpecStuck)
next
  case Seq thus ?case by (metis insertCI execn.Seq StuckProp)
next
  case CondTrue thus ?case by (auto intro: execn.CondTrue)
next
  case CondFalse thus ?case by (auto intro: execn.CondFalse)
next
  case WhileTrue thus ?case by (metis insertCI execn.WhileTrue StuckProp)
next
  case WhileFalse thus ?case by (auto intro: execn.WhileFalse)
next
  case (Call p bdy n s s')
  have  $\Gamma\ p = \text{Some } \textit{bdy}$  by fact
  show ?case
  proof (cases p ∈ P)
    case True
    with Call have  $(\Gamma|_P)\ p = \text{Some } \textit{bdy}$ 
      by (simp)
    with Call show ?thesis
      by (auto intro: execn.intros)
  next
  case False
  hence  $(\Gamma|_P)\ p = \text{None}$  by simp
  thus ?thesis

```

```

    by (auto intro: execn.CallUndefined)
  qed
next
  case (CallUndefined p n s)
  have  $\Gamma \ p = \text{None}$  by fact
  hence  $(\Gamma|_P) \ p = \text{None}$  by (rule restrict-NoneD)
  thus ?case by (auto intro: execn.CallUndefined)
next
  case StuckProp thus ?case by (auto intro: execn.StuckProp)
next
  case DynCom thus ?case by (auto intro: execn.DynCom)
next
  case Throw thus ?case by (auto intro: execn.Throw)
next
  case AbruptProp thus ?case by (auto intro: execn.AbruptProp)
next
  case (CatchMatch c1 s n s' c2 s'')
  from CatchMatch.hyps
  obtain  $t' \ t''$  where
    exec-res-c1:  $\Gamma|_P \vdash \langle c1, \text{Normal } s \rangle =_n \Rightarrow t'$  and
    t'-notStuck:  $t' \neq \text{Stuck} \longrightarrow t' = \text{Abrupt } s'$  and
    exec-res-c2:  $\Gamma|_P \vdash \langle c2, \text{Normal } s' \rangle =_n \Rightarrow t''$  and
    s''-Stuck:  $s'' = \text{Stuck} \longrightarrow t'' = \text{Stuck}$  and
    s''-Fault:  $\forall f. s'' = \text{Fault } f \longrightarrow t'' \in \{\text{Fault } f, \text{Stuck}\}$  and
    t''-notStuck:  $t'' \neq \text{Stuck} \longrightarrow t'' = s''$ 
  by auto
  show ?case
  proof (cases  $t' = \text{Stuck}$ )
    case True
    with exec-res-c1
    have  $\Gamma|_P \vdash \langle \text{Catch } c1 \ c2, \text{Normal } s \rangle =_n \Rightarrow \text{Stuck}$ 
    by (auto intro: execn.CatchMiss)
    thus ?thesis
    by auto
  next
    case False
    with t'-notStuck have  $t' = \text{Abrupt } s'$ 
    by simp
    with exec-res-c1 exec-res-c2
    have  $\Gamma|_P \vdash \langle \text{Catch } c1 \ c2, \text{Normal } s \rangle =_n \Rightarrow t''$ 
    by (auto intro: execn.CatchMatch)
    with s''-Stuck s''-Fault t''-notStuck
    show ?thesis
    by blast
  qed
next
  case (CatchMiss c1 s n w c2)
  have exec-c1:  $\Gamma \vdash \langle c1, \text{Normal } s \rangle =_n \Rightarrow w$  by fact
  from CatchMiss.hyps obtain  $w'$  where

```

$exec-c1': \Gamma \mid p \vdash \langle c1, Normal\ s \rangle = n \Rightarrow w'$  **and**  
 $w\text{-Stuck}: w = Stuck \longrightarrow w' = Stuck$  **and**  
 $w\text{-Fault}: \forall f. w = Fault\ f \longrightarrow w' \in \{Fault\ f, Stuck\}$  **and**  
 $w'\text{-noStuck}: w' \neq Stuck \longrightarrow w' = w$   
**by auto**  
**have**  $noAbr\text{-}w: \neg isAbr\ w$  **by fact**  
**show**  $?case$   
**proof** ( $cases\ w'$ )  
**case** ( $Normal\ s'$ )  
**with**  $w'\text{-noStuck}$  **have**  $w'=w$   
**by simp**  
**with**  $exec-c1'\ Normal\ w\text{-Stuck}\ w\text{-Fault}\ w'\text{-noStuck}$   
**show**  $?thesis$   
**by** ( $fastforce\ intro: execn.CatchMiss$ )  
**next**  
**case** ( $Abrupt\ s'$ )  
**with**  $w'\text{-noStuck}$  **have**  $w'=w$   
**by simp**  
**with**  $noAbr\text{-}w\ Abrupt$  **show**  $?thesis$  **by simp**  
**next**  
**case** ( $Fault\ f$ )  
**with**  $w'\text{-noStuck}$  **have**  $w'=w$   
**by simp**  
**with**  $exec-c1'\ Fault\ w\text{-Stuck}\ w\text{-Fault}\ w'\text{-noStuck}$   
**show**  $?thesis$   
**by** ( $fastforce\ intro: execn.CatchMiss$ )  
**next**  
**case**  $Stuck$   
**with**  $exec-c1'\ w\text{-Stuck}\ w\text{-Fault}\ w'\text{-noStuck}$   
**show**  $?thesis$   
**by** ( $fastforce\ intro: execn.CatchMiss$ )  
**qed**  
**qed**

**lemma**  $exec\text{-to}\text{-}exec\text{-restrict}$ :

**assumes**  $exec: \Gamma \vdash \langle c, s \rangle \Rightarrow t$

**shows**  $\exists t'. \Gamma \mid p \vdash \langle c, s \rangle \Rightarrow t' \wedge (t = Stuck \longrightarrow t' = Stuck) \wedge$   
 $(\forall f. t = Fault\ f \longrightarrow t' \in \{Fault\ f, Stuck\}) \wedge (t \neq Stuck \longrightarrow t' = t)$

**proof** –

**from**  $exec$  **obtain**  $n$  **where**

$execn\text{-strip}: \Gamma \vdash \langle c, s \rangle = n \Rightarrow t$

**by** ( $auto\ simp\ add: exec\text{-iff}\text{-}execn$ )

**from**  $execn\text{-to}\text{-}execn\text{-restrict}$  [**where**  $P=P, OF\ this$ ]

**obtain**  $t'$  **where**

$\Gamma \mid p \vdash \langle c, s \rangle = n \Rightarrow t'$

$t = Stuck \longrightarrow t' = Stuck \ \forall f. t = Fault\ f \longrightarrow t' \in \{Fault\ f, Stuck\} \ t \neq Stuck \longrightarrow t' = t$

**by blast**

**thus**  $?thesis$



by (*blast intro: execn-to-exec*)  
**qed**

**lemma notStuck-GuardD:**

$\llbracket \Gamma \vdash \langle \text{Guard } m \ g \ c, \text{Normal } s \rangle \Rightarrow \notin \{ \text{Stuck} \}; s \in g \rrbracket \Longrightarrow \Gamma \vdash \langle c, \text{Normal } s \rangle \Rightarrow \notin \{ \text{Stuck} \}$   
**by** (*auto simp add: final-notin-def dest: exec.Guard*)

**lemma notStuck-SeqD1:**

$\llbracket \Gamma \vdash \langle \text{Seq } c1 \ c2, \text{Normal } s \rangle \Rightarrow \notin \{ \text{Stuck} \} \rrbracket \Longrightarrow \Gamma \vdash \langle c1, \text{Normal } s \rangle \Rightarrow \notin \{ \text{Stuck} \}$   
**by** (*auto simp add: final-notin-def dest: exec.Seq*)

**lemma notStuck-SeqD2:**

$\llbracket \Gamma \vdash \langle \text{Seq } c1 \ c2, \text{Normal } s \rangle \Rightarrow \notin \{ \text{Stuck} \}; \Gamma \vdash \langle c1, \text{Normal } s \rangle \Rightarrow s' \rrbracket \Longrightarrow \Gamma \vdash \langle c2, s' \rangle \Rightarrow \notin \{ \text{Stuck} \}$   
**by** (*auto simp add: final-notin-def dest: exec.Seq*)

**lemma notStuck-SeqD:**

$\llbracket \Gamma \vdash \langle \text{Seq } c1 \ c2, \text{Normal } s \rangle \Rightarrow \notin \{ \text{Stuck} \} \rrbracket \Longrightarrow$   
 $\Gamma \vdash \langle c1, \text{Normal } s \rangle \Rightarrow \notin \{ \text{Stuck} \} \wedge (\forall s'. \Gamma \vdash \langle c1, \text{Normal } s \rangle \Rightarrow s' \longrightarrow \Gamma \vdash \langle c2, s' \rangle \Rightarrow \notin \{ \text{Stuck} \})$   
**by** (*auto simp add: final-notin-def dest: exec.Seq*)

**lemma notStuck-CondTrueD:**

$\llbracket \Gamma \vdash \langle \text{Cond } b \ c1 \ c2, \text{Normal } s \rangle \Rightarrow \notin \{ \text{Stuck} \}; s \in b \rrbracket \Longrightarrow \Gamma \vdash \langle c1, \text{Normal } s \rangle \Rightarrow \notin \{ \text{Stuck} \}$   
**by** (*auto simp add: final-notin-def dest: exec.CondTrue*)

**lemma notStuck-CondFalseD:**

$\llbracket \Gamma \vdash \langle \text{Cond } b \ c1 \ c2, \text{Normal } s \rangle \Rightarrow \notin \{ \text{Stuck} \}; s \notin b \rrbracket \Longrightarrow \Gamma \vdash \langle c2, \text{Normal } s \rangle \Rightarrow \notin \{ \text{Stuck} \}$   
**by** (*auto simp add: final-notin-def dest: exec.CondFalse*)

**lemma notStuck-WhileTrueD1:**

$\llbracket \Gamma \vdash \langle \text{While } b \ c, \text{Normal } s \rangle \Rightarrow \notin \{ \text{Stuck} \}; s \in b \rrbracket$   
 $\Longrightarrow \Gamma \vdash \langle c, \text{Normal } s \rangle \Rightarrow \notin \{ \text{Stuck} \}$   
**by** (*auto simp add: final-notin-def dest: exec.WhileTrue*)

**lemma notStuck-WhileTrueD2:**

$\llbracket \Gamma \vdash \langle \text{While } b \ c, \text{Normal } s \rangle \Rightarrow \notin \{ \text{Stuck} \}; \Gamma \vdash \langle c, \text{Normal } s \rangle \Rightarrow s'; s \in b \rrbracket$   
 $\Longrightarrow \Gamma \vdash \langle \text{While } b \ c, s' \rangle \Rightarrow \notin \{ \text{Stuck} \}$   
**by** (*auto simp add: final-notin-def dest: exec.WhileTrue*)

**lemma notStuck-CallD:**

$\llbracket \Gamma \vdash \langle \text{Call } p, \text{Normal } s \rangle \Rightarrow \notin \{ \text{Stuck} \}; \Gamma \ p = \text{Some } \text{bdy} \rrbracket$   
 $\Longrightarrow \Gamma \vdash \langle \text{bdy}, \text{Normal } s \rangle \Rightarrow \notin \{ \text{Stuck} \}$   
**by** (*auto simp add: final-notin-def dest: exec.Call*)

**lemma notStuck-CallDefinedD:**

$\llbracket \Gamma \vdash \langle \text{Call } p, \text{Normal } s \rangle \Rightarrow \notin \{ \text{Stuck} \} \rrbracket$   
 $\Longrightarrow \Gamma \ p \neq \text{None}$

**by** (*cases*  $\Gamma$   $p$ )  
 (*auto simp add: final-notin-def dest: exec.CallUndefined*)

**lemma** *notStuck-DynComD*:  
 $\llbracket \Gamma \vdash \langle \text{DynCom } c, \text{Normal } s \rangle \Rightarrow \notin \{ \text{Stuck} \} \rrbracket$   
 $\implies \Gamma \vdash \langle (c \ s), \text{Normal } s \rangle \Rightarrow \notin \{ \text{Stuck} \}$   
**by** (*auto simp add: final-notin-def dest: exec.DynCom*)

**lemma** *notStuck-CatchD1*:  
 $\llbracket \Gamma \vdash \langle \text{Catch } c1 \ c2, \text{Normal } s \rangle \Rightarrow \notin \{ \text{Stuck} \} \rrbracket \implies \Gamma \vdash \langle c1, \text{Normal } s \rangle \Rightarrow \notin \{ \text{Stuck} \}$   
**by** (*auto simp add: final-notin-def dest: exec.CatchMatch exec.CatchMiss*)

**lemma** *notStuck-CatchD2*:  
 $\llbracket \Gamma \vdash \langle \text{Catch } c1 \ c2, \text{Normal } s \rangle \Rightarrow \notin \{ \text{Stuck} \}; \Gamma \vdash \langle c1, \text{Normal } s \rangle \Rightarrow \text{Abrupt } s' \rrbracket$   
 $\implies \Gamma \vdash \langle c2, \text{Normal } s' \rangle \Rightarrow \notin \{ \text{Stuck} \}$   
**by** (*auto simp add: final-notin-def dest: exec.CatchMatch*)

### 3.10 Miscellaneous

**lemma** *execn-noguards-no-Fault*:  
**assumes** *execn*:  $\Gamma \vdash \langle c, s \rangle = n \Rightarrow t$   
**assumes** *noguards-c*: *noguards*  $c$   
**assumes** *noguards- $\Gamma$* :  $\forall p \in \text{dom } \Gamma. \text{noguards } (\text{the } (\Gamma \ p))$   
**assumes** *s-no-Fault*:  $\neg \text{isFault } s$   
**shows**  $\neg \text{isFault } t$   
**using** *execn noguards-c s-no-Fault*  
**proof** (*induct*)  
**case** (*Call*  $p$  *bdy*  $n$   $s$   $t$ ) **with** *noguards- $\Gamma$*  **show** *?case*  
**apply**  $-$   
**apply** (*drule* *bspec* [**where**  $x=p$ ])  
**apply** *auto*  
**done**  
**qed** (*auto*)

**lemma** *exec-noguards-no-Fault*:  
**assumes** *exec*:  $\Gamma \vdash \langle c, s \rangle \Rightarrow t$   
**assumes** *noguards-c*: *noguards*  $c$   
**assumes** *noguards- $\Gamma$* :  $\forall p \in \text{dom } \Gamma. \text{noguards } (\text{the } (\Gamma \ p))$   
**assumes** *s-no-Fault*:  $\neg \text{isFault } s$   
**shows**  $\neg \text{isFault } t$   
**using** *exec noguards-c s-no-Fault*  
**proof** (*induct*)  
**case** (*Call*  $p$  *bdy*  $s$   $t$ ) **with** *noguards- $\Gamma$*  **show** *?case*  
**apply**  $-$   
**apply** (*drule* *bspec* [**where**  $x=p$ ])  
**apply** *auto*  
**done**  
**qed** *auto*





**by** (*auto simp add: nvalid-def*)

**lemma validI:**

$\llbracket \bigwedge s t. \llbracket \Gamma \vdash \langle c, Normal\ s \rangle \Rightarrow t; s \in P; t \notin Fault\ ' F \rrbracket \Longrightarrow t \in Normal\ ' Q \cup Abrupt\ ' A \rrbracket$

$\Longrightarrow \Gamma \models_{/F} P\ c\ Q, A$

**by** (*auto simp add: valid-def*)

**lemma cvalidI:**

$\llbracket \bigwedge s t. \llbracket \forall (P, p, Q, A) \in \Theta. \Gamma \models_{/F} P\ (Call\ p)\ Q, A; \Gamma \vdash \langle c, Normal\ s \rangle \Rightarrow t; s \in P; t \notin Fault\ ' F \rrbracket$

$\Longrightarrow t \in Normal\ ' Q \cup Abrupt\ ' A \rrbracket$

$\Longrightarrow \Gamma, \Theta \models_{/F} P\ c\ Q, A$

**by** (*auto simp add: cvalid-def valid-def*)

**lemma cvalidD:**

$\llbracket \Gamma, \Theta \models_{/F} P\ c\ Q, A; \forall (P, p, Q, A) \in \Theta. \Gamma \models_{/F} P\ (Call\ p)\ Q, A; \Gamma \vdash \langle c, Normal\ s \rangle \Rightarrow t; s \in P; t \notin Fault\ ' F \rrbracket$

$\Longrightarrow t \in Normal\ ' Q \cup Abrupt\ ' A$

**by** (*auto simp add: cvalid-def valid-def*)

**lemma cnvalidI:**

$\llbracket \bigwedge s t. \llbracket \forall (P, p, Q, A) \in \Theta. \Gamma \models_{/n} P\ (Call\ p)\ Q, A;$

$\Gamma \vdash \langle c, Normal\ s \rangle =n \Rightarrow t; s \in P; t \notin Fault\ ' F \rrbracket$

$\Longrightarrow t \in Normal\ ' Q \cup Abrupt\ ' A \rrbracket$

$\Longrightarrow \Gamma, \Theta \models_{/n} P\ c\ Q, A$

**by** (*auto simp add: cnvalid-def nvalid-def*)

**lemma cnvalidD:**

$\llbracket \Gamma, \Theta \models_{/n} P\ c\ Q, A; \forall (P, p, Q, A) \in \Theta. \Gamma \models_{/n} P\ (Call\ p)\ Q, A;$

$\Gamma \vdash \langle c, Normal\ s \rangle =n \Rightarrow t; s \in P;$

$t \notin Fault\ ' F \rrbracket$

$\Longrightarrow t \in Normal\ ' Q \cup Abrupt\ ' A$

**by** (*auto simp add: cnvalid-def nvalid-def*)

**lemma nvalid-augment-Faults:**

**assumes** *validn*:  $\Gamma \models_{/n} P\ c\ Q, A$

**assumes** *F'*:  $F \subseteq F'$

**shows**  $\Gamma \models_{/n} P\ c\ Q, A$

**proof** (*rule nvalidI*)

**fix** *s t*

**assume** *exec*:  $\Gamma \vdash \langle c, Normal\ s \rangle =n \Rightarrow t$

**assume** *P*:  $s \in P$

**assume** *F*:  $t \notin Fault\ ' F'$

**with** *F'* **have**  $t \notin Fault\ ' F$

**by** *blast*

**with** *exec P validn*

**show**  $t \in \text{Normal} \text{ ' } Q \cup \text{Abrupt} \text{ ' } A$   
**by** (*auto simp add: nvalid-def*)  
**qed**

**lemma** *valid-augment-Faults*:

**assumes**  $\text{validn}:\Gamma \models_{/F} P \text{ c } Q, A$

**assumes**  $F': F \subseteq F'$

**shows**  $\Gamma \models_{/F'} P \text{ c } Q, A$

**proof** (*rule validI*)

**fix**  $s \ t$

**assume**  $\text{exec}:\Gamma \vdash \langle c, \text{Normal } s \rangle \Rightarrow t$

**assume**  $P: s \in P$

**assume**  $F: t \notin \text{Fault} \text{ ' } F'$

**with**  $F'$  **have**  $t \notin \text{Fault} \text{ ' } F$

**by** *blast*

**with**  $\text{exec } P \ \text{validn}$

**show**  $t \in \text{Normal} \text{ ' } Q \cup \text{Abrupt} \text{ ' } A$

**by** (*auto simp add: valid-def*)

**qed**

**lemma** *nvalid-to-nvalid-strip*:

**assumes**  $\text{validn}:\Gamma \models_{/F} P \text{ c } Q, A$

**assumes**  $F': F' \subseteq -F$

**shows**  $\text{strip } F' \ \Gamma \models_{/F} P \text{ c } Q, A$

**proof** (*rule nvalidI*)

**fix**  $s \ t$

**assume**  $\text{exec-strip}:\text{strip } F' \ \Gamma \vdash \langle c, \text{Normal } s \rangle =n \Rightarrow t$

**assume**  $P: s \in P$

**assume**  $F: t \notin \text{Fault} \text{ ' } F$

**from**  $\text{exec-strip}$  **obtain**  $t'$  **where**

$\text{exec}:\Gamma \vdash \langle c, \text{Normal } s \rangle =n \Rightarrow t'$  **and**

$t': t' \in \text{Fault} \text{ ' } (-F') \longrightarrow t'=t \neg \text{isFault } t' \longrightarrow t'=t$

**by** (*blast dest: execn-strip-to-execn*)

**show**  $t \in \text{Normal} \text{ ' } Q \cup \text{Abrupt} \text{ ' } A$

**proof** (*cases*  $t' \in \text{Fault} \text{ ' } F$ )

**case** *True*

**with**  $t' \ F \ F'$  **have** *False*

**by** *blast*

**thus** *?thesis ..*

**next**

**case** *False*

**with**  $\text{exec } P \ \text{validn}$

**have**  $*$ :  $t' \in \text{Normal} \text{ ' } Q \cup \text{Abrupt} \text{ ' } A$

**by** (*auto simp add: nvalid-def*)

**with**  $t'$  **have**  $t'=t$

**by** *auto*

**with**  $*$  **show** *?thesis*

**by** *simp*

**qed**

qed

**lemma** *valid-to-valid-strip*:

**assumes** *valid*:  $\Gamma \models_{/F} P \ c \ Q, A$

**assumes** *F'*:  $F' \subseteq -F$

**shows** *strip* *F'*  $\Gamma \models_{/F} P \ c \ Q, A$

**proof** (*rule validI*)

**fix** *s t*

**assume** *exec-strip*: *strip F' Γ ⊢ ⟨c, Normal s⟩ ⇒ t*

**assume** *P*:  $s \in P$

**assume** *F*:  $t \notin \text{Fault } F$

**from** *exec-strip* **obtain** *t'* **where**

*exec*:  $\Gamma \vdash \langle c, \text{Normal } s \rangle \Rightarrow t'$  **and**

*t'*:  $t' \in \text{Fault } (-F') \longrightarrow t'=t \wedge \text{isFault } t' \longrightarrow t'=t$

**by** (*blast dest: exec-strip-to-exec*)

**show**  $t \in \text{Normal } Q \cup \text{Abrupt } A$

**proof** (*cases t' ∈ Fault F'*)

**case** *True*

**with** *t' F F'* **have** *False*

**by** *blast*

**thus** *?thesis ..*

**next**

**case** *False*

**with** *exec P valid*

**have** *\**:  $t' \in \text{Normal } Q \cup \text{Abrupt } A$

**by** (*auto simp add: valid-def*)

**with** *t'* **have**  $t'=t$

**by** *auto*

**with** *\** **show** *?thesis*

**by** *simp*

qed

qed

### 4.3 The Hoare Rules: $\Gamma, \Theta \vdash_{/F} P \ c \ Q, A$

**lemma** *mono-WeakenContext*:  $A \subseteq B \implies$

$(\lambda(P, c, Q, A'). (\Gamma, \Theta, F, P, c, Q, A') \in A) \ x \longrightarrow$

$(\lambda(P, c, Q, A'). (\Gamma, \Theta, F, P, c, Q, A') \in B) \ x$

**apply** *blast*

**done**

**inductive** *hoarep*:: $(('s, 'p, 'f) \text{ body}, ('s, 'p) \text{ quadruple set}, 'f \text{ set},$

$'s \text{ assn}, ('s, 'p, 'f) \text{ com}, 's \text{ assn}, 's \text{ assn}] \implies \text{bool}$

$(\langle (\exists -, - \vdash' / \_ \ - / \ (-) / \ - , \ -) \rangle [60, 60, 60, 1000, 20, 1000, 1000] 60)$

**for**  $\Gamma :: ('s, 'p, 'f) \text{ body}$

**where**

*Skip*:  $\Gamma, \Theta \vdash_{/F} Q \ \text{Skip} \ Q, A$

- | *Basic*:  $\Gamma, \Theta \vdash_F \{s. f s \in Q\}$  (*Basic f*)  $Q, A$
- | *Spec*:  $\Gamma, \Theta \vdash_F \{s. (\forall t. (s, t) \in r \longrightarrow t \in Q) \wedge (\exists t. (s, t) \in r)\}$  (*Spec r*)  $Q, A$
- | *Seq*:  $\llbracket \Gamma, \Theta \vdash_F P \ c_1 \ R, A; \Gamma, \Theta \vdash_F R \ c_2 \ Q, A \rrbracket$   
 $\implies$   
 $\Gamma, \Theta \vdash_F P \ (\text{Seq } c_1 \ c_2) \ Q, A$
- | *Cond*:  $\llbracket \Gamma, \Theta \vdash_F (P \cap b) \ c_1 \ Q, A; \Gamma, \Theta \vdash_F (P \cap - \ b) \ c_2 \ Q, A \rrbracket$   
 $\implies$   
 $\Gamma, \Theta \vdash_F P \ (\text{Cond } b \ c_1 \ c_2) \ Q, A$
- | *While*:  $\Gamma, \Theta \vdash_F (P \cap b) \ c \ P, A$   
 $\implies$   
 $\Gamma, \Theta \vdash_F P \ (\text{While } b \ c) \ (P \cap - \ b), A$
- | *Guard*:  $\Gamma, \Theta \vdash_F (g \cap P) \ c \ Q, A$   
 $\implies$   
 $\Gamma, \Theta \vdash_F (g \cap P) \ (\text{Guard } f \ g \ c) \ Q, A$
- | *Guarantee*:  $\llbracket f \in F; \Gamma, \Theta \vdash_F (g \cap P) \ c \ Q, A \rrbracket$   
 $\implies$   
 $\Gamma, \Theta \vdash_F P \ (\text{Guard } f \ g \ c) \ Q, A$
- | *CallRec*:  
 $\llbracket (P, p, Q, A) \in \text{Specs};$   
 $\forall (P, p, Q, A) \in \text{Specs}. p \in \text{dom } \Gamma \wedge \Gamma, \Theta \cup \text{Specs} \vdash_F P \ (\text{the } (\Gamma \ p)) \ Q, A \rrbracket$   
 $\implies \Gamma, \Theta \vdash_F P \ (\text{Call } p) \ Q, A$
- | *DynCom*:  
 $\forall s \in P. \Gamma, \Theta \vdash_F P \ (c \ s) \ Q, A$   
 $\implies$   
 $\Gamma, \Theta \vdash_F P \ (\text{DynCom } c) \ Q, A$
- | *Throw*:  $\Gamma, \Theta \vdash_F A \ \text{Throw} \ Q, A$
- | *Catch*:  $\llbracket \Gamma, \Theta \vdash_F P \ c_1 \ Q, R; \Gamma, \Theta \vdash_F R \ c_2 \ Q, A \rrbracket \implies \Gamma, \Theta \vdash_F P \ \text{Catch } c_1 \ c_2 \ Q, A$
- | *Conseq*:  $\forall s \in P. \exists P' \ Q' \ A'. \Gamma, \Theta \vdash_F P' \ c \ Q', A' \wedge s \in P' \wedge Q' \subseteq Q \wedge A' \subseteq A$   
 $\implies \Gamma, \Theta \vdash_F P \ c \ Q, A$
- | *Asm*:  $\llbracket (P, p, Q, A) \in \Theta \rrbracket$   
 $\implies$   
 $\Gamma, \Theta \vdash_F P \ (\text{Call } p) \ Q, A$
- | *ExFalso*:  $\llbracket \forall n. \Gamma, \Theta \models_n \vdash_F P \ c \ Q, A; \neg \Gamma \models \vdash_F P \ c \ Q, A \rrbracket \implies \Gamma, \Theta \vdash_F P \ c \ Q, A$



— This is a hack rule that enables us to derive completeness for an arbitrary context  $\Theta$ , from completeness for an empty context.

Does not work, because of rule `ExFalso`, the context  $\Theta$  is to blame. A weaker version with empty context can be derived from soundness and completeness later on.

```

lemma hoare-strip-Γ:
  assumes deriv:  $\Gamma, \Theta \vdash /_F P \ p \ Q, A$ 
  shows strip  $(-F) \ \Gamma, \Theta \vdash /_F P \ p \ Q, A$ 
using deriv
proof induct
  case Skip thus ?case by (iprover intro: hoarep.Skip)
next
  case Basic thus ?case by (iprover intro: hoarep.Basic)
next
  case Spec thus ?case by (iprover intro: hoarep.Spec)
next
  case Seq thus ?case by (iprover intro: hoarep.Seq)
next
  case Cond thus ?case by (iprover intro: hoarep.Cond)
next
  case While thus ?case by (iprover intro: hoarep.While)
next
  case Guard thus ?case by (iprover intro: hoarep.Guard)

next
  case DynCom
  thus ?case
  by — (rule hoarep.DynCom, best elim!: ballE exE)
next
  case Throw thus ?case by (iprover intro: hoarep.Throw)
next
  case Catch thus ?case by (iprover intro: hoarep.Catch)

next
  case Asm thus ?case by (iprover intro: hoarep.Asm)
next
  case ExFalso
  thus ?case
  oops

```

```

lemma hoare-augment-context:
  assumes deriv:  $\Gamma, \Theta \vdash /_F P \ p \ Q, A$ 
  shows  $\bigwedge \Theta'. \ \Theta \subseteq \Theta' \implies \Gamma, \Theta' \vdash /_F P \ p \ Q, A$ 
using deriv
proof (induct)
  case CallRec
  case (CallRec  $P \ p \ Q \ A \ \text{Specs} \ \Theta \ F \ \Theta'$ )

```

**from** *CallRec.prem*s  
**have**  $\Theta \cup \text{Specs}$   
 $\subseteq \Theta' \cup \text{Specs}$   
**by** *blast*  
**with** *CallRec.hyps* (2)  
**have**  $\forall (P, p, Q, A) \in \text{Specs}. p \in \text{dom } \Gamma \wedge \Gamma, \Theta' \cup \text{Specs} \vdash_{/F} P$  (the  $(\Gamma p)$ )  $Q, A$   
**by** *fastforce*  
  
**with** *CallRec* **show** *?case* **by** – (rule *hoarep.CallRec*)  
**next**  
**case** *DynCom* **thus** *?case* **by** (*blast intro: hoarep.DynCom*)  
**next**  
**case** (*Conseq*  $P \Theta F c Q A \Theta'$ )  
**from** *Conseq*  
**have**  $\forall s \in P.$   
 $(\exists P' Q' A'. \Gamma, \Theta' \vdash_{/F} P' c Q', A' \wedge s \in P' \wedge Q' \subseteq Q \wedge A' \subseteq A)$   
**by** *blast*  
**with** *Conseq* **show** *?case* **by** – (rule *hoarep.Conseq*)  
**next**  
**case** (*ExFalso*  $\Theta F P c Q A \Theta'$ )  
**have** *valid-ctxt*:  $\forall n. \Gamma, \Theta \models_n \vdash_{/F} P c Q, A \Theta \subseteq \Theta'$  **by** *fact+*  
**hence**  $\forall n. \Gamma, \Theta' \models_n \vdash_{/F} P c Q, A$   
**by** (*simp add: cinvalid-def*) *blast*  
**moreover** **have** *invalid*:  $\neg \Gamma \models_{/F} P c Q, A$  **by** *fact*  
**ultimately** **show** *?case*  
**by** (*rule hoarep.ExFalso*)  
**qed** (*blast intro: hoarep.intros*)+

#### 4.4 Some Derived Rules

**lemma** *Conseq'*:  $\forall s. s \in P \longrightarrow$   
 $(\exists P' Q' A'.$   
 $(\forall Z. \Gamma, \Theta \vdash_{/F} (P' Z) c (Q' Z), (A' Z)) \wedge$   
 $(\exists Z. s \in P' Z \wedge (Q' Z \subseteq Q) \wedge (A' Z \subseteq A)))$   
 $\implies$   
 $\Gamma, \Theta \vdash_{/F} P c Q, A$

**apply** (*rule Conseq*)  
**apply** (*rule ballI*)  
**apply** (*erule-tac x=s in allE*)  
**apply** (*clarify*)  
**apply** (*rule-tac x=P' Z in exI*)  
**apply** (*rule-tac x=Q' Z in exI*)  
**apply** (*rule-tac x=A' Z in exI*)  
**apply** *blast*  
**done**

**lemma** *conseq*:  $\llbracket \forall Z. \Gamma, \Theta \vdash_{/F} (P' Z) c (Q' Z), (A' Z);$   
 $\forall s. s \in P \longrightarrow (\exists Z. s \in P' Z \wedge (Q' Z \subseteq Q) \wedge (A' Z \subseteq A)) \rrbracket$

$\Longrightarrow$   
 $\Gamma, \Theta \vdash_{/F} P \text{ c } Q, A$   
**by** (rule *Conseq*) *blast*

**theorem** *conseqPrePost* [trans]:

$\Gamma, \Theta \vdash_{/F} P' \text{ c } Q', A' \Longrightarrow P \subseteq P' \Longrightarrow Q' \subseteq Q \Longrightarrow A' \subseteq A \Longrightarrow \Gamma, \Theta \vdash_{/F} P \text{ c } Q, A$   
**by** (rule *conseq* [where ?P'= $\lambda$ Z. P' and ?Q'= $\lambda$ Z. Q']) *auto*

**lemma** *conseqPre* [trans]:  $\Gamma, \Theta \vdash_{/F} P' \text{ c } Q, A \Longrightarrow P \subseteq P' \Longrightarrow \Gamma, \Theta \vdash_{/F} P \text{ c } Q, A$

**by** (rule *conseq*) *auto*

**lemma** *conseqPost* [trans]:  $\Gamma, \Theta \vdash_{/F} P \text{ c } Q', A' \Longrightarrow Q' \subseteq Q \Longrightarrow A' \subseteq A$

$\Longrightarrow \Gamma, \Theta \vdash_{/F} P \text{ c } Q, A$

**by** (rule *conseq*) *auto*

**lemma** *CallRec'*:

$\llbracket p \in \text{Procs}; \text{Procs} \subseteq \text{dom } \Gamma;$

$\forall p \in \text{Procs}.$

$\forall Z. \Gamma, \Theta \cup (\bigcup p \in \text{Procs}. \bigcup Z. \{((P \text{ p } Z), p, Q \text{ p } Z, A \text{ p } Z)\})$

$\vdash_{/F} (P \text{ p } Z) (\text{the } (\Gamma \text{ p})) (Q \text{ p } Z), (A \text{ p } Z)]$

$\Longrightarrow$

$\Gamma, \Theta \vdash_{/F} (P \text{ p } Z) (\text{Call } p) (Q \text{ p } Z), (A \text{ p } Z)$

**apply** (rule *CallRec* [where *Specs*= $\bigcup p \in \text{Procs}. \bigcup Z. \{((P \text{ p } Z), p, Q \text{ p } Z, A \text{ p } Z)\}$ ])

**apply** *blast*

**apply** *blast*

**done**

**end**

## 5 Properties of Partial Correctness Hoare Logic

**theory** *HoarePartialProps* **imports** *HoarePartialDef* **begin**

### 5.1 Soundness

**lemma** *hoare-cnvalid*:

**assumes** *hoare*:  $\Gamma, \Theta \vdash_{/F} P \text{ c } Q, A$

**shows**  $\bigwedge n. \Gamma, \Theta \models n:_{/F} P \text{ c } Q, A$

**using** *hoare*

**proof** (*induct*)

**case** (*Skip*  $\Theta \text{ F } P \text{ A}$ )

**show**  $\Gamma, \Theta \models n:_{/F} P \text{ Skip } P, A$

**proof** (rule *cnvalidI*)

**fix** *s t*

**assume**  $\Gamma \vdash \langle \text{Skip}, \text{Normal } s \rangle = n \Rightarrow t \text{ s } \in P$

**thus**  $t \in \text{Normal } \text{' } P \cup \text{Abrupt } \text{' } A$

```

    by cases auto
  qed
next
case (Basic  $\Theta$   $F$   $f$   $P$   $A$ )
show  $\Gamma, \Theta \models_{/F} \{s. f\ s \in P\}$  (Basic  $f$ )  $P, A$ 
proof (rule cvalidI)
  fix  $s\ t$ 
  assume  $\Gamma \vdash \langle \text{Basic } f, \text{Normal } s \rangle =_{n \Rightarrow} t\ s \in \{s. f\ s \in P\}$ 
  thus  $t \in \text{Normal } ' P \cup \text{Abrupt } ' A$ 
  by cases auto
qed
next
case (Spec  $\Theta$   $F$   $r$   $Q$   $A$ )
show  $\Gamma, \Theta \models_{/F} \{s. (\forall t. (s, t) \in r \longrightarrow t \in Q) \wedge (\exists t. (s, t) \in r)\}$  Spec  $r$   $Q, A$ 
proof (rule cvalidI)
  fix  $s\ t$ 
  assume exec:  $\Gamma \vdash \langle \text{Spec } r, \text{Normal } s \rangle =_{n \Rightarrow} t$ 
  assume  $P: s \in \{s. (\forall t. (s, t) \in r \longrightarrow t \in Q) \wedge (\exists t. (s, t) \in r)\}$ 
  from exec  $P$ 
  show  $t \in \text{Normal } ' Q \cup \text{Abrupt } ' A$ 
  by cases auto
qed
next
case (Seq  $\Theta$   $F$   $P$   $c1$   $R$   $A$   $c2$   $Q$ )
have valid-c1:  $\bigwedge n. \Gamma, \Theta \models_{/F} P\ c1\ R, A$  by fact
have valid-c2:  $\bigwedge n. \Gamma, \Theta \models_{/F} R\ c2\ Q, A$  by fact
show  $\Gamma, \Theta \models_{/F} P\ \text{Seq } c1\ c2\ Q, A$ 
proof (rule cvalidI)
  fix  $s\ t$ 
  assume ctxt:  $\forall (P, p, Q, A) \in \Theta. \Gamma \models_{/F} P\ (\text{Call } p)\ Q, A$ 
  assume exec:  $\Gamma \vdash \langle \text{Seq } c1\ c2, \text{Normal } s \rangle =_{n \Rightarrow} t$ 
  assume t-notin-F:  $t \notin \text{Fault } ' F$ 
  assume  $P: s \in P$ 
  from exec  $P$  obtain  $r$  where
    exec-c1:  $\Gamma \vdash \langle c1, \text{Normal } s \rangle =_{n \Rightarrow} r$  and exec-c2:  $\Gamma \vdash \langle c2, r \rangle =_{n \Rightarrow} t$ 
  by cases auto
  with t-notin-F have  $r \notin \text{Fault } ' F$ 
  by (auto dest: execn-Fault-end)
  with valid-c1 ctxt exec-c1  $P$ 
  have  $r: r \in \text{Normal } ' R \cup \text{Abrupt } ' A$ 
  by (rule cvalidD)
  show  $t \in \text{Normal } ' Q \cup \text{Abrupt } ' A$ 
  proof (cases  $r$ )
    case (Normal  $r'$ )
    with exec-c2  $r$ 
    show  $t \in \text{Normal } ' Q \cup \text{Abrupt } ' A$ 
    apply -
    apply (rule cvalidD [OF valid-c2 ctxt - - t-notin-F])

```

```

    apply auto
  done
next
  case (Abrupt r')
  with exec-c2 have t=Abrupt r'
  by (auto elim: execn-elim-cases)
  with Abrupt r show ?thesis
  by auto
next
  case Fault with r show ?thesis by blast
next
  case Stuck with r show ?thesis by blast
qed
qed
next
  case (Cond  $\Theta$  F P b c1 Q A c2)
  have valid-c1:  $\bigwedge n. \Gamma, \Theta \models_{n: /F} (P \cap b) c1 Q, A$  by fact
  have valid-c2:  $\bigwedge n. \Gamma, \Theta \models_{n: /F} (P \cap \neg b) c2 Q, A$  by fact
  show  $\Gamma, \Theta \models_{n: /F} P \text{ Cond } b c1 c2 Q, A$ 
  proof (rule cinvalidI)
    fix s t
    assume ctxt:  $\forall (P, p, Q, A) \in \Theta. \Gamma \models_{n: /F} P (\text{Call } p) Q, A$ 
    assume exec:  $\Gamma \vdash \langle \text{Cond } b c1 c2, \text{Normal } s \rangle =_{n \Rightarrow} t$ 
    assume P:  $s \in P$ 
    assume t-notin-F:  $t \notin \text{Fault } F$ 
    show  $t \in \text{Normal } Q \cup \text{Abrupt } A$ 
    proof (cases s ∈ b)
      case True
      with exec have  $\Gamma \vdash \langle c1, \text{Normal } s \rangle =_{n \Rightarrow} t$ 
      by cases auto
      with P True
      show ?thesis
      by - (rule cinvalidD [OF valid-c1 ctxt - - t-notin-F], auto)
    next
      case False
      with exec P have  $\Gamma \vdash \langle c2, \text{Normal } s \rangle =_{n \Rightarrow} t$ 
      by cases auto
      with P False
      show ?thesis
      by - (rule cinvalidD [OF valid-c2 ctxt - - t-notin-F], auto)
    qed
  qed
next
  case (While  $\Theta$  F P b c A n)
  have valid-c:  $\bigwedge n. \Gamma, \Theta \models_{n: /F} (P \cap b) c P, A$  by fact
  show  $\Gamma, \Theta \models_{n: /F} P \text{ While } b c (P \cap \neg b), A$ 
  proof (rule cinvalidI)
    fix s t

```

```

assume ctxt:  $\forall (P, p, Q, A) \in \Theta. \Gamma \models_n: /_F P (Call\ p) Q, A$ 
assume exec:  $\Gamma \vdash \langle While\ b\ c, Normal\ s \rangle =n \Rightarrow t$ 
assume P:  $s \in P$ 
assume t-notin-F:  $t \notin Fault\ 'F$ 
show  $t \in Normal\ '(P \cap -\ b) \cup Abrupt\ 'A$ 
proof (cases  $s \in b$ )
  case True
  {
    fix d::('b,'a,'c) com fix s t
    assume exec:  $\Gamma \vdash \langle d, s \rangle =n \Rightarrow t$ 
    assume d:  $d = While\ b\ c$ 
    assume ctxt:  $\forall (P, p, Q, A) \in \Theta. \Gamma \models_n: /_F P (Call\ p) Q, A$ 
    from exec d ctxt
    have  $\llbracket s \in Normal\ 'P; t \notin Fault\ 'F \rrbracket$ 
       $\Longrightarrow t \in Normal\ '(P \cap -\ b) \cup Abrupt\ 'A$ 
    proof (induct)
      case (WhileTrue  $s\ b'\ c'\ n\ r\ t$ )
      have t-notin-F:  $t \notin Fault\ 'F$  by fact
      have eqs:  $While\ b'\ c' = While\ b\ c$  by fact
      note valid-c
      moreover have ctxt:  $\forall (P, p, Q, A) \in \Theta. \Gamma \models_n: /_F P (Call\ p) Q, A$  by fact
      moreover from WhileTrue
      obtain  $\Gamma \vdash \langle c, Normal\ s \rangle =n \Rightarrow r$  and
         $\Gamma \vdash \langle While\ b\ c, r \rangle =n \Rightarrow t$  and
         $Normal\ s \in Normal\ '(P \cap b)$  by auto
      moreover with t-notin-F have  $r \notin Fault\ 'F$ 
        by (auto dest: execn-Fault-end)
      ultimately
      have r:  $r \in Normal\ 'P \cup Abrupt\ 'A$ 
        by  $-$  (rule cnvalidD,auto)
      from this - ctxt
      show  $t \in Normal\ '(P \cap -\ b) \cup Abrupt\ 'A$ 
      proof (cases r)
        case (Normal  $r'$ )
        with r ctxt eqs t-notin-F
        show ?thesis
        by  $-$  (rule WhileTrue.hyps,auto)
      next
        case (Abrupt  $r'$ )
        have  $\Gamma \vdash \langle While\ b'\ c', r \rangle =n \Rightarrow t$  by fact
        with Abrupt have  $t = r$ 
        by (auto dest: execn-Abrupt-end)
        with r Abrupt show ?thesis
        by blast
      next
        case Fault with r show ?thesis by blast
      next
        case Stuck with r show ?thesis by blast
    qed
  }

```

```

    qed auto
  }
  with exec ctxt P t-notin-F
  show ?thesis
  by auto
next
  case False
  with exec P have t=Normal s
  by cases auto
  with P False
  show ?thesis
  by auto
qed
qed
next
  case (Guard  $\Theta$  F g P c Q A f)
  have valid-c:  $\bigwedge n. \Gamma, \Theta \models n: /_F (g \cap P) c Q, A$  by fact
  show  $\Gamma, \Theta \models n: /_F (g \cap P) \text{Guard } f g c Q, A$ 
  proof (rule cvalidI)
    fix s t
    assume ctxt:  $\forall (P, p, Q, A) \in \Theta. \Gamma \models n: /_F P (\text{Call } p) Q, A$ 
    assume exec:  $\Gamma \vdash \langle \text{Guard } f g c, \text{Normal } s \rangle = n \Rightarrow t$ 
    assume t-notin-F:  $t \notin \text{Fault } ' F$ 
    assume P:  $s \in (g \cap P)$ 
    from exec P have  $\Gamma \vdash \langle c, \text{Normal } s \rangle = n \Rightarrow t$ 
    by cases auto
    from valid-c ctxt this P t-notin-F
    show  $t \in \text{Normal } ' Q \cup \text{Abrupt } ' A$ 
    by (rule cvalidD)
  qed
next
  case (Guarantee f F  $\Theta$  g P c Q A)
  have valid-c:  $\bigwedge n. \Gamma, \Theta \models n: /_F (g \cap P) c Q, A$  by fact
  have f-F:  $f \in F$  by fact
  show  $\Gamma, \Theta \models n: /_F P \text{Guard } f g c Q, A$ 
  proof (rule cvalidI)
    fix s t
    assume ctxt:  $\forall (P, p, Q, A) \in \Theta. \Gamma \models n: /_F P (\text{Call } p) Q, A$ 
    assume exec:  $\Gamma \vdash \langle \text{Guard } f g c, \text{Normal } s \rangle = n \Rightarrow t$ 
    assume t-notin-F:  $t \notin \text{Fault } ' F$ 
    assume P:  $s \in P$ 
    from exec f-F t-notin-F have g:  $s \in g$ 
    by cases auto
    with P have P':  $s \in g \cap P$ 
    by blast
    from exec P g have  $\Gamma \vdash \langle c, \text{Normal } s \rangle = n \Rightarrow t$ 
    by cases auto
    from valid-c ctxt this P' t-notin-F

```

```

  show  $t \in \text{Normal} \text{ ' } Q \cup \text{Abrupt} \text{ ' } A$ 
  by (rule cinvalidD)
qed
next
case (CallRec  $P p Q A \text{ Specs } \Theta F$ )
have  $p: (P,p,Q,A) \in \text{Specs}$  by fact
have valid-body:
 $\forall (P,p,Q,A) \in \text{Specs}. p \in \text{dom } \Gamma \wedge (\forall n. \Gamma, \Theta \cup \text{Specs} \models_n /_F P \text{ (the } (\Gamma p)) Q, A)$ 
  using CallRec.hyps by blast
show  $\Gamma, \Theta \models_n /_F P \text{ Call } p Q, A$ 
proof -
{
  fix  $n$ 
  have  $\forall (P, p, Q, A) \in \Theta. \Gamma \models_n /_F P \text{ (Call } p) Q, A$ 
     $\implies \forall (P,p,Q,A) \in \text{Specs}. \Gamma \models_n /_F P \text{ (Call } p) Q, A$ 
  proof (induct  $n$ )
  case 0
  show  $\forall (P,p,Q,A) \in \text{Specs}. \Gamma \models_0 /_F P \text{ (Call } p) Q, A$ 
    by (fastforce elim!: execn-elim-cases simp add: nvalid-def)
  next
  case (Suc  $m$ )
  have hyp:  $\forall (P, p, Q, A) \in \Theta. \Gamma \models_m /_F P \text{ (Call } p) Q, A$ 
     $\implies \forall (P,p,Q,A) \in \text{Specs}. \Gamma \models_m /_F P \text{ (Call } p) Q, A$  by fact
  have  $\forall (P, p, Q, A) \in \Theta. \Gamma \models_{\text{Suc } m} /_F P \text{ (Call } p) Q, A$  by fact
  hence ctxt-m:  $\forall (P, p, Q, A) \in \Theta. \Gamma \models_m /_F P \text{ (Call } p) Q, A$ 
    by (fastforce simp add: nvalid-def intro: execn-Suc)
  hence valid-Proc:
     $\forall (P,p,Q,A) \in \text{Specs}. \Gamma \models_m /_F P \text{ (Call } p) Q, A$ 
    by (rule hyp)
  let  $?\Theta' = \Theta \cup \text{Specs}$ 
  from valid-Proc ctxt-m
  have  $\forall (P, p, Q, A) \in ?\Theta'. \Gamma \models_m /_F P \text{ (Call } p) Q, A$ 
    by fastforce
  with valid-body
  have valid-body-m:
     $\forall (P,p,Q,A) \in \text{Specs}. \forall n. \Gamma \models_m /_F P \text{ (the } (\Gamma p)) Q, A$ 
    by (fastforce simp add: cinvalid-def)
  show  $\forall (P,p,Q,A) \in \text{Specs}. \Gamma \models_{\text{Suc } m} /_F P \text{ (Call } p) Q, A$ 
  proof (clarify)
  fix  $P p Q A$  assume  $p: (P,p,Q,A) \in \text{Specs}$ 
  show  $\Gamma \models_{\text{Suc } m} /_F P \text{ (Call } p) Q, A$ 
  proof (rule nvalidI)
  fix  $s t$ 
  assume exec-call:
     $\Gamma \vdash \langle \text{Call } p, \text{Normal } s \rangle =_{\text{Suc } m} t$ 
  assume Pre:  $s \in P$ 
  assume t-notin-F:  $t \notin \text{Fault} \text{ ' } F$ 

```



```

    from exec-call
    show  $t \in \text{Normal} \text{ ' } Q \cup \text{Abrupt} \text{ ' } A$ 
    proof (cases)
      fix bdy m'
      assume m:  $\text{Suc } m = \text{Suc } m'$ 
      assume bdy:  $\Gamma \text{ } p = \text{Some } \text{bdy}$ 
      assume exec-body:  $\Gamma \vdash \langle \text{bdy}, \text{Normal } s \rangle = m' \Rightarrow t$ 
      from Pre valid-body-m exec-body bdy m p t-notin-F
      show ?thesis
        by (fastforce simp add: nvalid-def)
    next
      assume  $\Gamma \text{ } p = \text{None}$ 
      with valid-body p have False by auto
      thus ?thesis ..
    qed
  qed
}
with p show ?thesis
  by (fastforce simp add: cvalid-def)
qed
next
case (DynCom P  $\Theta$  F c Q A)
hence valid-c:  $\forall s \in P. (\forall n. \Gamma, \Theta \models n: /_F P (c s) Q, A)$  by auto
show  $\Gamma, \Theta \models n: /_F P \text{ DynCom } c Q, A$ 
proof (rule cvalidI)
  fix s t
  assume ctxt:  $\forall (P, p, Q, A) \in \Theta. \Gamma \models n: /_F P (\text{Call } p) Q, A$ 
  assume exec:  $\Gamma \vdash \langle \text{DynCom } c, \text{Normal } s \rangle = n \Rightarrow t$ 
  assume P:  $s \in P$ 
  assume t-notin-Fault:  $t \notin \text{Fault} \text{ ' } F$ 
  from exec show  $t \in \text{Normal} \text{ ' } Q \cup \text{Abrupt} \text{ ' } A$ 
  proof (cases)
    assume  $\Gamma \vdash \langle c s, \text{Normal } s \rangle = n \Rightarrow t$ 
    from cvalidD [OF valid-c [rule-format, OF P]] ctxt this P t-notin-Fault
    show ?thesis .
  qed
qed
next
case (Throw  $\Theta$  F A Q)
show  $\Gamma, \Theta \models n: /_F A \text{ Throw } Q, A$ 
proof (rule cvalidI)
  fix s t
  assume  $\Gamma \vdash \langle \text{Throw}, \text{Normal } s \rangle = n \Rightarrow t \text{ } s \in A$ 
  then show  $t \in \text{Normal} \text{ ' } Q \cup \text{Abrupt} \text{ ' } A$ 
    by cases simp
qed
next

```

**case** (*Catch*  $\Theta$   $F$   $P$   $c_1$   $Q$   $R$   $c_2$   $A$ )  
**have** *valid-c1*:  $\bigwedge n. \Gamma, \Theta \models_{n:/F} P \ c_1 \ Q, R$  **by fact**  
**have** *valid-c2*:  $\bigwedge n. \Gamma, \Theta \models_{n:/F} R \ c_2 \ Q, A$  **by fact**  
**show**  $\Gamma, \Theta \models_{n:/F} P \ \text{Catch} \ c_1 \ c_2 \ Q, A$   
**proof** (*rule cinvalidI*)  
**fix**  $s \ t$   
**assume** *ctxt*:  $\forall (P, p, Q, A) \in \Theta. \Gamma \models_{n:/F} P \ (\text{Call } p) \ Q, A$   
**assume** *exec*:  $\Gamma \vdash \langle \text{Catch } c_1 \ c_2, \text{Normal } s \rangle =_{n \Rightarrow} t$   
**assume**  $P: s \in P$   
**assume** *t-notin-Fault*:  $t \notin \text{Fault} \ ' F$   
**from** *exec* **show**  $t \in \text{Normal} \ ' Q \cup \text{Abrupt} \ ' A$   
**proof** (*cases*)  
**fix**  $s'$   
**assume** *exec-c1*:  $\Gamma \vdash \langle c_1, \text{Normal } s \rangle =_{n \Rightarrow} \text{Abrupt } s'$   
**assume** *exec-c2*:  $\Gamma \vdash \langle c_2, \text{Normal } s' \rangle =_{n \Rightarrow} t$   
**from** *cinvalidD* [*OF valid-c1 ctxt exec-c1 P* ]  
**have**  $\text{Abrupt } s' \in \text{Abrupt} \ ' R$   
**by auto**  
**with** *cinvalidD* [*OF valid-c2 ctxt - - t-notin-Fault*] *exec-c2*  
**show** *?thesis*  
**by fastforce**  
**next**  
**assume** *exec-c1*:  $\Gamma \vdash \langle c_1, \text{Normal } s \rangle =_{n \Rightarrow} t$   
**assume** *notAbr*:  $\neg \text{isAbr } t$   
**from** *cinvalidD* [*OF valid-c1 ctxt exec-c1 P t-notin-Fault*]  
**have**  $t \in \text{Normal} \ ' Q \cup \text{Abrupt} \ ' R$  .  
**with** *notAbr*  
**show** *?thesis*  
**by auto**  
**qed**  
**qed**  
**next**  
**case** (*Conseq*  $P$   $\Theta$   $F$   $c$   $Q$   $A$ )  
**hence** *adapt*:  $\forall s \in P. (\exists P' \ Q' \ A'. \Gamma, \Theta \models_{n:/F} P' \ c \ Q', A' \wedge$   
 $s \in P' \wedge Q' \subseteq Q \wedge A' \subseteq A)$   
**by blast**  
**show**  $\Gamma, \Theta \models_{n:/F} P \ c \ Q, A$   
**proof** (*rule cinvalidI*)  
**fix**  $s \ t$   
**assume** *ctxt*:  $\forall (P, p, Q, A) \in \Theta. \Gamma \models_{n:/F} P \ (\text{Call } p) \ Q, A$   
**assume** *exec*:  $\Gamma \vdash \langle c, \text{Normal } s \rangle =_{n \Rightarrow} t$   
**assume**  $P: s \in P$   
**assume** *t-notin-F*:  $t \notin \text{Fault} \ ' F$   
**show**  $t \in \text{Normal} \ ' Q \cup \text{Abrupt} \ ' A$   
**proof** –  
**from**  $P$  **adapt** **obtain**  $P' \ Q' \ A' \ Z$  **where**  
*spec*:  $\Gamma, \Theta \models_{n:/F} P' \ c \ Q', A'$  **and**  
 $P': s \in P'$  **and** *strengthen*:  $Q' \subseteq Q \wedge A' \subseteq A$

```

    by auto
  from spec [rule-format] ctxt exec P' t-notin-F
  have t ∈ Normal ' Q' ∪ Abrupt ' A'
    by (rule cinvalidD)
  with strengthen show ?thesis
    by blast
qed
qed
next
case (Asm P p Q A Θ F)
have asm: (P, p, Q, A) ∈ Θ by fact
show  $\Gamma, \Theta \models_{/F} P$  (Call p) Q, A
proof (rule cinvalidI)
  fix s t
  assume ctxt:  $\forall (P, p, Q, A) \in \Theta. \Gamma \models_{/F} P$  (Call p) Q, A
  assume exec:  $\Gamma \vdash \langle \text{Call } p, \text{Normal } s \rangle = n \Rightarrow t$ 
  from asm ctxt have  $\Gamma \models_{/F} P$  Call p Q, A by auto
  moreover
  assume  $s \in P$  t  $\notin$  Fault ' F
  ultimately
  show t ∈ Normal ' Q ∪ Abrupt ' A
    using exec
    by (auto simp add: nvalid-def)
qed
next
case ExFalse thus ?case by iprover
qed

theorem hoare-sound:  $\Gamma, \Theta \vdash_{/F} P$  c Q, A  $\implies \Gamma, \Theta \models_{/F} P$  c Q, A
  by (iprover intro: cinvalid-to-cvalid hoare-cinvalid)

```

## 5.2 Completeness

**lemma** *MGT-valid*:

```

 $\Gamma \models_{/F} \{s. s = Z \wedge \Gamma \vdash \langle c, \text{Normal } s \rangle \Rightarrow \notin (\{\text{Stuck}\} \cup \text{Fault ' } (-F))\} c$ 
  {t.  $\Gamma \vdash \langle c, \text{Normal } Z \rangle \Rightarrow \text{Normal } t$ }, {t.  $\Gamma \vdash \langle c, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t$ }
proof (rule validI)
  fix s t
  assume  $\Gamma \vdash \langle c, \text{Normal } s \rangle \Rightarrow t$ 
  s ∈ {s. s = Z ∧  $\Gamma \vdash \langle c, \text{Normal } s \rangle \Rightarrow \notin (\{\text{Stuck}\} \cup \text{Fault ' } (-F))\}$ 
  t  $\notin$  Fault ' F
  thus t ∈ Normal ' {t.  $\Gamma \vdash \langle c, \text{Normal } Z \rangle \Rightarrow \text{Normal } t$ } ∪
    Abrupt ' {t.  $\Gamma \vdash \langle c, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t$ }
    by (cases t) (auto simp add: final-notin-def)
qed

```

The consequence rule where the existential  $Z$  is instantiated to  $s$ . Useful in proof of *MGT-lemma*.

**lemma** *ConseqMGT*:

**assumes** *modif*:  $\forall Z. \Gamma, \Theta \vdash_{/F} (P' Z) c (Q' Z), (A' Z)$   
**assumes** *impl*:  $\bigwedge s. s \in P \implies s \in P' s \wedge (\forall t. t \in Q' s \longrightarrow t \in Q) \wedge$   
 $(\forall t. t \in A' s \longrightarrow t \in A)$   
**shows**  $\Gamma, \Theta \vdash_{/F} P c Q, A$   
**using** *impl*  
**by** – (*rule conseq [OF modif], blast*)

**lemma** *Seq-NoFaultStuckD1*:  
**assumes** *noabort*:  $\Gamma \vdash \langle \text{Seq } c1 \ c2, s \rangle \Rightarrow \notin (\{Stuck\} \cup \text{Fault } ' F)$   
**shows**  $\Gamma \vdash \langle c1, s \rangle \Rightarrow \notin (\{Stuck\} \cup \text{Fault } ' F)$   
**proof** (*rule final-notinI*)  
**fix** *t*  
**assume** *exec-c1*:  $\Gamma \vdash \langle c1, s \rangle \Rightarrow t$   
**show**  $t \notin \{Stuck\} \cup \text{Fault } ' F$   
**proof**  
**assume**  $t \in \{Stuck\} \cup \text{Fault } ' F$   
**moreover**  
**{**  
**assume**  $t = Stuck$   
**with** *exec-c1*  
**have**  $\Gamma \vdash \langle \text{Seq } c1 \ c2, s \rangle \Rightarrow Stuck$   
**by** (*auto intro: exec-Seq'*)  
**with** *noabort* **have** *False*  
**by** (*auto simp add: final-notin-def*)  
**hence** *False ..*  
**}**  
**moreover**  
**{**  
**assume**  $t \in \text{Fault } ' F$   
**then obtain** *f* **where**  
 $t = \text{Fault } f$  **and**  $f \in F$   
**by** *auto*  
**from** *t exec-c1*  
**have**  $\Gamma \vdash \langle \text{Seq } c1 \ c2, s \rangle \Rightarrow \text{Fault } f$   
**by** (*auto intro: exec-Seq'*)  
**with** *noabort f* **have** *False*  
**by** (*auto simp add: final-notin-def*)  
**hence** *False ..*  
**}**  
**ultimately show** *False* **by** *auto*  
**qed**  
**qed**

**lemma** *Seq-NoFaultStuckD2*:  
**assumes** *noabort*:  $\Gamma \vdash \langle \text{Seq } c1 \ c2, s \rangle \Rightarrow \notin (\{Stuck\} \cup \text{Fault } ' F)$   
**shows**  $\forall t. \Gamma \vdash \langle c1, s \rangle \Rightarrow t \longrightarrow t \notin (\{Stuck\} \cup \text{Fault } ' F) \longrightarrow$   
 $\Gamma \vdash \langle c2, t \rangle \Rightarrow \notin (\{Stuck\} \cup \text{Fault } ' F)$   
**using** *noabort*

by (*auto simp add: final-notin-def intro: exec-Seq*)

**lemma** *MGT-implies-complete:*

**assumes** *MGT*:  $\forall Z. \Gamma, \{\} \vdash_{/F} \{s. s=Z \wedge \Gamma \vdash \langle c, Normal \ s \rangle \Rightarrow \notin(\{Stuck\} \cup \text{Fault } (-F))\} \ c$

$\{t. \Gamma \vdash \langle c, Normal \ Z \rangle \Rightarrow Normal \ t\},$   
 $\{t. \Gamma \vdash \langle c, Normal \ Z \rangle \Rightarrow Abrupt \ t\}$

**assumes** *valid*:  $\Gamma \models_{/F} P \ c \ Q, A$

**shows**  $\Gamma, \{\} \vdash_{/F} P \ c \ Q, A$

**using** *MGT*

**apply** (*rule ConseqMGT*)

**apply** (*insert valid*)

**apply** (*auto simp add: valid-def intro!: final-notinI*)

**done**

Equipped only with the classic consequence rule  $\llbracket ?\Gamma, ?\Theta \vdash_{/?F} ?P' \ ?c \ ?Q', ?A'; ?P \subseteq ?P'; ?Q' \subseteq ?Q; ?A' \subseteq ?A \rrbracket \implies ?\Gamma, ?\Theta \vdash_{/?F} ?P \ ?c \ ?Q, ?A$  we can only derive this syntactically more involved version of completeness. But semantically it is equivalent to the "real" one (see below)

**lemma** *MGT-implies-complete'*:

**assumes** *MGT*:  $\forall Z. \Gamma, \{\} \vdash_{/F}$

$\{s. s=Z \wedge \Gamma \vdash \langle c, Normal \ s \rangle \Rightarrow \notin(\{Stuck\} \cup \text{Fault } (-F))\} \ c$   
 $\{t. \Gamma \vdash \langle c, Normal \ Z \rangle \Rightarrow Normal \ t\},$   
 $\{t. \Gamma \vdash \langle c, Normal \ Z \rangle \Rightarrow Abrupt \ t\}$

**assumes** *valid*:  $\Gamma \models_{/F} P \ c \ Q, A$

**shows**  $\Gamma, \{\} \vdash_{/F} \{s. s=Z \wedge s \in P\} \ c \ \{t. Z \in P \longrightarrow t \in Q\}, \{t. Z \in P \longrightarrow t \in A\}$

**using** *MGT* [*rule-format, of Z*]

**apply** (*rule conseqPrePost*)

**apply** (*insert valid*)

**apply** (*fastforce simp add: valid-def final-notin-def*)

**apply** (*fastforce simp add: valid-def*)

**apply** (*fastforce simp add: valid-def*)

**done**

Semantic equivalence of both kind of formulations

**lemma** *valid-involved-to-valid:*

**assumes** *valid*:

$\forall Z. \Gamma \models_{/F} \{s. s=Z \wedge s \in P\} \ c \ \{t. Z \in P \longrightarrow t \in Q\}, \{t. Z \in P \longrightarrow t \in A\}$

**shows**  $\Gamma \models_{/F} P \ c \ Q, A$

**using** *valid*

**apply** (*simp add: valid-def*)

**apply** *clarsimp*

**apply** (*erule-tac x=x in allE*)

**apply** (*erule-tac x=Normal x in allE*)

**apply** (*erule-tac x=t in allE*)

**apply** *fastforce*  
**done**

The sophisticated consequence rule allow us to do this semantical transformation on the hoare-level, too. The magic is, that it allow us to choose the instance of  $Z$  under the assumption of an state  $s \in P$

**lemma**

**assumes** *deriv*:

$\forall Z. \Gamma, \{\} \vdash_{/F} \{s. s=Z \wedge s \in P\} c \{t. Z \in P \longrightarrow t \in Q\}, \{t. Z \in P \longrightarrow t \in A\}$

**shows**  $\Gamma, \{\} \vdash_{/F} P c Q, A$

**apply** (*rule ConseqMGT [OF deriv]*)

**apply** *auto*

**done**

**lemma** *valid-to-valid-involved*:

$\Gamma \models_{/F} P c Q, A \implies$

$\Gamma \models_{/F} \{s. s=Z \wedge s \in P\} c \{t. Z \in P \longrightarrow t \in Q\}, \{t. Z \in P \longrightarrow t \in A\}$

**by** (*simp add: valid-def Collect-conv-if*)

**lemma**

**assumes** *deriv*:  $\Gamma, \{\} \vdash_{/F} P c Q, A$

**shows**  $\Gamma, \{\} \vdash_{/F} \{s. s=Z \wedge s \in P\} c \{t. Z \in P \longrightarrow t \in Q\}, \{t. Z \in P \longrightarrow t \in A\}$

**apply** (*rule conseqPrePost [OF deriv]*)

**apply** *auto*

**done**

**lemma** *conseq-extract-state-indep-prop*:

**assumes** *state-indep-prop*:  $\forall s \in P. R$

**assumes** *to-show*:  $R \implies \Gamma, \Theta \vdash_{/F} P c Q, A$

**shows**  $\Gamma, \Theta \vdash_{/F} P c Q, A$

**apply** (*rule Conseq*)

**apply** (*clarify*)

**apply** (*rule-tac x=P in exI*)

**apply** (*rule-tac x=Q in exI*)

**apply** (*rule-tac x=A in exI*)

**using** *state-indep-prop to-show*

**by** *blast*

**lemma** *MGT-lemma*:

**assumes** *MGT-Calls*:

$\forall p \in \text{dom } \Gamma. \forall Z. \Gamma, \Theta \vdash_{/F}$

$\{s. s=Z \wedge \Gamma \vdash \langle \text{Call } p, \text{Normal } s \rangle \Rightarrow \notin (\{\text{Stuck}\} \cup \text{Fault } '(-F))\}$   
*(Call p)*

$\{t. \Gamma \vdash \langle \text{Call } p, \text{Normal } Z \rangle \Rightarrow \text{Normal } t\},$

$\{t. \Gamma \vdash \langle \text{Call } p, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\}$

**shows**  $\bigwedge Z. \Gamma, \Theta \vdash /_F \{s. s = Z \wedge \Gamma \vdash \langle c, \text{Normal } s \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F))\} c$   
 $\{t. \Gamma \vdash \langle c, \text{Normal } Z \rangle \Rightarrow \text{Normal } t\}, \{t. \Gamma \vdash \langle c, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\}$

**proof** (*induct c*)

**case** *Skip*

**show**  $\Gamma, \Theta \vdash /_F \{s. s = Z \wedge \Gamma \vdash \langle \text{Skip}, \text{Normal } s \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F))\}$   
*Skip*  
 $\{t. \Gamma \vdash \langle \text{Skip}, \text{Normal } Z \rangle \Rightarrow \text{Normal } t\}, \{t. \Gamma \vdash \langle \text{Skip}, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\}$

**by** (*rule hoarep.Skip [THEN conseqPre]*)  
*(auto elim: exec-elim-cases simp add: final-notin-def intro: exec.intros)*

**next**

**case** (*Basic f*)

**show**  $\Gamma, \Theta \vdash /_F \{s. s = Z \wedge \Gamma \vdash \langle \text{Basic } f, \text{Normal } s \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F))\}$   
*Basic f*  
 $\{t. \Gamma \vdash \langle \text{Basic } f, \text{Normal } Z \rangle \Rightarrow \text{Normal } t\},$   
 $\{t. \Gamma \vdash \langle \text{Basic } f, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\}$

**by** (*rule hoarep.Basic [THEN conseqPre]*)  
*(auto elim: exec-elim-cases simp add: final-notin-def intro: exec.intros)*

**next**

**case** (*Spec r*)

**show**  $\Gamma, \Theta \vdash /_F \{s. s = Z \wedge \Gamma \vdash \langle \text{Spec } r, \text{Normal } s \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F))\}$   
*Spec r*  
 $\{t. \Gamma \vdash \langle \text{Spec } r, \text{Normal } Z \rangle \Rightarrow \text{Normal } t\},$   
 $\{t. \Gamma \vdash \langle \text{Spec } r, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\}$

**apply** (*rule hoarep.Spec [THEN conseqPre]*)  
**apply** (*clarsimp simp add: final-notin-def*)  
**apply** (*case-tac  $\exists t. (Z, t) \in r$* )  
**apply** (*auto elim: exec-elim-cases simp add: final-notin-def intro: exec.intros*)

**done**

**next**

**case** (*Seq c1 c2*)

**have** *hyp-c1*:  $\forall Z. \Gamma, \Theta \vdash /_F \{s. s = Z \wedge \Gamma \vdash \langle c1, \text{Normal } s \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F))\} c1$   
 $\{t. \Gamma \vdash \langle c1, \text{Normal } Z \rangle \Rightarrow \text{Normal } t\},$   
 $\{t. \Gamma \vdash \langle c1, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\}$

**using** *Seq.hyps by iprover*

**have** *hyp-c2*:  $\forall Z. \Gamma, \Theta \vdash /_F \{s. s = Z \wedge \Gamma \vdash \langle c2, \text{Normal } s \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F))\} c2$   
 $\{t. \Gamma \vdash \langle c2, \text{Normal } Z \rangle \Rightarrow \text{Normal } t\},$   
 $\{t. \Gamma \vdash \langle c2, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\}$

**using** *Seq.hyps by iprover*

**from** *hyp-c1*

**have**  $\Gamma, \Theta \vdash /_F \{s. s = Z \wedge \Gamma \vdash \langle \text{Seq } c1 c2, \text{Normal } s \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F))\}$   
*c1*  
 $\{t. \Gamma \vdash \langle c1, \text{Normal } Z \rangle \Rightarrow \text{Normal } t \wedge$   
 $\Gamma \vdash \langle c2, \text{Normal } t \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F))\},$   
 $\{t. \Gamma \vdash \langle \text{Seq } c1 c2, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\}$

**by** (*rule ConseqMGT*)  
*(auto dest: Seq-NoFaultStuckD1 [simplified] Seq-NoFaultStuckD2 [simplified])*

*intro: exec.Seq*  
**thus**  $\Gamma, \Theta \vdash_{/F} \{s. s=Z \wedge \Gamma \vdash \langle \text{Seq } c1 \ c2, \text{Normal } s \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F))\}$   
 $\text{Seq } c1 \ c2$   
 $\{t. \Gamma \vdash \langle \text{Seq } c1 \ c2, \text{Normal } Z \rangle \Rightarrow \text{Normal } t\},$   
 $\{t. \Gamma \vdash \langle \text{Seq } c1 \ c2, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\}$   
**proof** (*rule hoarep.Seq*)  
**show**  $\Gamma, \Theta \vdash_{/F} \{t. \Gamma \vdash \langle c1, \text{Normal } Z \rangle \Rightarrow \text{Normal } t \wedge$   
 $\Gamma \vdash \langle c2, \text{Normal } t \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F))\}$   
 $c2$   
 $\{t. \Gamma \vdash \langle \text{Seq } c1 \ c2, \text{Normal } Z \rangle \Rightarrow \text{Normal } t\},$   
 $\{t. \Gamma \vdash \langle \text{Seq } c1 \ c2, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\}$   
**proof** (*rule ConseqMGT [OF hyp-c2], safe*)  
**fix**  $r \ t$   
**assume**  $\Gamma \vdash \langle c1, \text{Normal } Z \rangle \Rightarrow \text{Normal } r \ \Gamma \vdash \langle c2, \text{Normal } r \rangle \Rightarrow \text{Normal } t$   
**then show**  $\Gamma \vdash \langle \text{Seq } c1 \ c2, \text{Normal } Z \rangle \Rightarrow \text{Normal } t$   
**by** (*iprover intro: exec.intros*)  
**next**  
**fix**  $r \ t$   
**assume**  $\Gamma \vdash \langle c1, \text{Normal } Z \rangle \Rightarrow \text{Normal } r \ \Gamma \vdash \langle c2, \text{Normal } r \rangle \Rightarrow \text{Abrupt } t$   
**then show**  $\Gamma \vdash \langle \text{Seq } c1 \ c2, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t$   
**by** (*iprover intro: exec.intros*)  
**qed**  
**qed**  
**next**  
**case** (*Cond b c1 c2*)  
**have**  $\forall Z. \Gamma, \Theta \vdash_{/F} \{s. s=Z \wedge \Gamma \vdash \langle c1, \text{Normal } s \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F))\} \ c1$   
 $\{t. \Gamma \vdash \langle c1, \text{Normal } Z \rangle \Rightarrow \text{Normal } t\},$   
 $\{t. \Gamma \vdash \langle c1, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\}$   
**using** *Cond.hyps* **by** *iprover*  
**hence**  $\Gamma, \Theta \vdash_{/F} (\{s. s=Z \wedge \Gamma \vdash \langle \text{Cond } b \ c1 \ c2, \text{Normal } s \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F))\} \cap b)$   
 $c1$   
 $\{t. \Gamma \vdash \langle \text{Cond } b \ c1 \ c2, \text{Normal } Z \rangle \Rightarrow \text{Normal } t\},$   
 $\{t. \Gamma \vdash \langle \text{Cond } b \ c1 \ c2, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\}$   
**by** (*rule ConseqMGT*)  
(*fastforce intro: exec.CondTrue simp add: final-notin-def*)  
**moreover**  
**have**  $\forall Z. \Gamma, \Theta \vdash_{/F} \{s. s=Z \wedge \Gamma \vdash \langle c2, \text{Normal } s \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F))\} \ c2$   
 $\{t. \Gamma \vdash \langle c2, \text{Normal } Z \rangle \Rightarrow \text{Normal } t\},$   
 $\{t. \Gamma \vdash \langle c2, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\}$   
**using** *Cond.hyps* **by** *iprover*  
**hence**  $\Gamma, \Theta \vdash_{/F} (\{s. s=Z \wedge \Gamma \vdash \langle \text{Cond } b \ c1 \ c2, \text{Normal } s \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F))\} \cap \neg b)$   
 $c2$   
 $\{t. \Gamma \vdash \langle \text{Cond } b \ c1 \ c2, \text{Normal } Z \rangle \Rightarrow \text{Normal } t\},$   
 $\{t. \Gamma \vdash \langle \text{Cond } b \ c1 \ c2, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\}$   
**by** (*rule ConseqMGT*)  
(*fastforce intro: exec.CondFalse simp add: final-notin-def*)



**ultimately**  
**show**  $\Gamma, \Theta \vdash_{/F} \{s. s=Z \wedge \Gamma \vdash \langle \text{Cond } b \ c1 \ c2, \text{Normal } s \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F)) \}$   
 $(-F))\}$   
 $\text{Cond } b \ c1 \ c2$   
 $\{t. \Gamma \vdash \langle \text{Cond } b \ c1 \ c2, \text{Normal } Z \rangle \Rightarrow \text{Normal } t\},$   
 $\{t. \Gamma \vdash \langle \text{Cond } b \ c1 \ c2, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\}$   
**by** (*rule hoarep.Cond*)  
**next**  
**case** (*While b c*)  
**let**  $?unroll = (\{(s,t). s \in b \wedge \Gamma \vdash \langle c, \text{Normal } s \rangle \Rightarrow \text{Normal } t\})^*$   
**let**  $?P' = \lambda Z. \{t. (Z,t) \in ?unroll \wedge$   
 $(\forall e. (Z,e) \in ?unroll \longrightarrow e \in b$   
 $\longrightarrow \Gamma \vdash \langle c, \text{Normal } e \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F)) \wedge$   
 $(\forall u. \Gamma \vdash \langle c, \text{Normal } e \rangle \Rightarrow \text{Abrupt } u \longrightarrow$   
 $\Gamma \vdash \langle \text{While } b \ c, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } u)\}$   
**let**  $?A' = \lambda Z. \{t. \Gamma \vdash \langle \text{While } b \ c, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\}$   
**show**  $\Gamma, \Theta \vdash_{/F} \{s. s=Z \wedge \Gamma \vdash \langle \text{While } b \ c, \text{Normal } s \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F))\}$   
 $\text{While } b \ c$   
 $\{t. \Gamma \vdash \langle \text{While } b \ c, \text{Normal } Z \rangle \Rightarrow \text{Normal } t\},$   
 $\{t. \Gamma \vdash \langle \text{While } b \ c, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\}$   
**proof** (*rule ConseqMGT [where ?P'=?P'*  
**and**  $?Q' = \lambda Z. ?P' \ Z \cap - \ b$  **and**  $?A' = ?A']$ )  
**show**  $\forall Z. \Gamma, \Theta \vdash_{/F} (?P' \ Z) (\text{While } b \ c) (?P' \ Z \cap - \ b), (?A' \ Z)$   
**proof** (*rule allI, rule hoarep.While*)  
**fix**  $Z$   
**from** *While*  
**have**  $\forall Z. \Gamma, \Theta \vdash_{/F} \{s. s=Z \wedge \Gamma \vdash \langle c, \text{Normal } s \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F))\}$   
 $c$   
 $\{t. \Gamma \vdash \langle c, \text{Normal } Z \rangle \Rightarrow \text{Normal } t\},$   
 $\{t. \Gamma \vdash \langle c, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\}$  **by** *iprover*  
**then show**  $\Gamma, \Theta \vdash_{/F} (?P' \ Z \cap b) \ c \ (?P' \ Z), (?A' \ Z)$   
**proof** (*rule ConseqMGT*)  
**fix**  $s$   
**assume**  $s \in \{t. (Z, t) \in ?unroll \wedge$   
 $(\forall e. (Z,e) \in ?unroll \longrightarrow e \in b$   
 $\longrightarrow \Gamma \vdash \langle c, \text{Normal } e \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F)) \wedge$   
 $(\forall u. \Gamma \vdash \langle c, \text{Normal } e \rangle \Rightarrow \text{Abrupt } u \longrightarrow$   
 $\Gamma \vdash \langle \text{While } b \ c, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } u)\}$   
 $\cap b$   
**then obtain**  
 $Z\text{-}s\text{-unroll}: (Z,s) \in ?unroll$  **and**  
 $noabort: \forall e. (Z,e) \in ?unroll \longrightarrow e \in b$   
 $\longrightarrow \Gamma \vdash \langle c, \text{Normal } e \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F)) \wedge$   
 $(\forall u. \Gamma \vdash \langle c, \text{Normal } e \rangle \Rightarrow \text{Abrupt } u \longrightarrow$   
 $\Gamma \vdash \langle \text{While } b \ c, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } u)$  **and**  
 $s\text{-in-}b: s \in b$   
**by** *blast*  
**show**  $s \in \{t. t = s \wedge \Gamma \vdash \langle c, \text{Normal } t \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F))\} \wedge$   
 $(\forall t. t \in \{t. \Gamma \vdash \langle c, \text{Normal } s \rangle \Rightarrow \text{Normal } t\} \longrightarrow$

$t \in \{t. (Z, t) \in ?unroll \wedge$   
 $(\forall e. (Z, e) \in ?unroll \longrightarrow e \in b$   
 $\longrightarrow \Gamma \vdash \langle c, Normal e \rangle \Rightarrow \notin (\{Stuck\} \cup Fault \text{ ' } (-F)) \wedge$   
 $(\forall u. \Gamma \vdash \langle c, Normal e \rangle \Rightarrow Abrupt u \longrightarrow$   
 $\Gamma \vdash \langle While b c, Normal Z \rangle \Rightarrow Abrupt u))\}$   $\wedge$   
 $(\forall t. t \in \{t. \Gamma \vdash \langle c, Normal s \rangle \Rightarrow Abrupt t\} \longrightarrow$   
 $t \in \{t. \Gamma \vdash \langle While b c, Normal Z \rangle \Rightarrow Abrupt t\})$   
 $(is ?C1 \wedge ?C2 \wedge ?C3)$   
**proof** (*intro conjI*)  
**from** *Z-s-unroll noabort s-in-b* **show** *?C1* **by** *blast*  
**next**  
 $\{$   
**fix** *t*  
**assume** *s-t*:  $\Gamma \vdash \langle c, Normal s \rangle \Rightarrow Normal t$   
**moreover**  
**from** *Z-s-unroll s-t s-in-b*  
**have**  $(Z, t) \in ?unroll$   
**by** (*blast intro: rtrancl-into-rtrancl*)  
**moreover note** *noabort*  
**ultimately**  
**have**  $(Z, t) \in ?unroll \wedge$   
 $(\forall e. (Z, e) \in ?unroll \longrightarrow e \in b$   
 $\longrightarrow \Gamma \vdash \langle c, Normal e \rangle \Rightarrow \notin (\{Stuck\} \cup Fault \text{ ' } (-F)) \wedge$   
 $(\forall u. \Gamma \vdash \langle c, Normal e \rangle \Rightarrow Abrupt u \longrightarrow$   
 $\Gamma \vdash \langle While b c, Normal Z \rangle \Rightarrow Abrupt u))$   
**by** *iprover*  
 $\}$   
**then show** *?C2* **by** *blast*  
**next**  
 $\{$   
**fix** *t*  
**assume** *s-t*:  $\Gamma \vdash \langle c, Normal s \rangle \Rightarrow Abrupt t$   
**from** *Z-s-unroll noabort s-t s-in-b*  
**have**  $\Gamma \vdash \langle While b c, Normal Z \rangle \Rightarrow Abrupt t$   
**by** *blast*  
 $\}$  **thus** *?C3* **by** *simp*  
**qed**  
**qed**  
**qed**  
**next**  
**fix** *s*  
**assume** *P*:  $s \in \{s. s=Z \wedge \Gamma \vdash \langle While b c, Normal s \rangle \Rightarrow \notin (\{Stuck\} \cup Fault \text{ ' } (-F))\}$   
**hence** *WhileNoFault*:  $\Gamma \vdash \langle While b c, Normal Z \rangle \Rightarrow \notin (\{Stuck\} \cup Fault \text{ ' } (-F))$   
**by** *auto*  
**show**  $s \in ?P' s \wedge$   
 $(\forall t. t \in (?P' s \cap - b) \longrightarrow$   
 $t \in \{t. \Gamma \vdash \langle While b c, Normal Z \rangle \Rightarrow Normal t\}) \wedge$   
 $(\forall t. t \in ?A' s \longrightarrow t \in ?A' Z)$

```

proof (intro conjI)
{
  fix e
  assume  $(Z, e) \in ?unroll\ e \in b$ 
  from this WhileNoFault
  have  $\Gamma \vdash \langle c, Normal\ e \rangle \Rightarrow \notin(\{Stuck\} \cup Fault\ '(-F)) \wedge$ 
     $(\forall u. \Gamma \vdash \langle c, Normal\ e \rangle \Rightarrow Abrupt\ u \longrightarrow$ 
       $\Gamma \vdash \langle While\ b\ c, Normal\ Z \rangle \Rightarrow Abrupt\ u)$  (is ?Prop Z e)
  proof (induct rule: converse-rtrancl-induct [consumes 1])
    assume e-in-b: e ∈ b
    assume WhileNoFault:  $\Gamma \vdash \langle While\ b\ c, Normal\ e \rangle \Rightarrow \notin(\{Stuck\} \cup Fault\ '(-F))$ 
    (-F))
    with e-in-b WhileNoFault
    have cNoFault:  $\Gamma \vdash \langle c, Normal\ e \rangle \Rightarrow \notin(\{Stuck\} \cup Fault\ '(-F))$ 
      by (auto simp add: final-notin-def intro: exec.intros)
    moreover
    {
      fix u assume  $\Gamma \vdash \langle c, Normal\ e \rangle \Rightarrow Abrupt\ u$ 
      with e-in-b have  $\Gamma \vdash \langle While\ b\ c, Normal\ e \rangle \Rightarrow Abrupt\ u$ 
      by (blast intro: exec.intros)
    }
    ultimately
    show ?Prop e e
    by iprover
  next
  fix Z r
  assume e-in-b: e ∈ b
  assume WhileNoFault:  $\Gamma \vdash \langle While\ b\ c, Normal\ Z \rangle \Rightarrow \notin(\{Stuck\} \cup Fault\ '(-F))$ 
  (-F))
  assume hyp:  $[[e \in b; \Gamma \vdash \langle While\ b\ c, Normal\ r \rangle \Rightarrow \notin(\{Stuck\} \cup Fault\ '(-F))]]$ 
     $\implies ?Prop\ r\ e$ 
  assume Z-r:
     $(Z, r) \in \{(Z, r). Z \in b \wedge \Gamma \vdash \langle c, Normal\ Z \rangle \Rightarrow Normal\ r\}$ 
  with WhileNoFault
  have  $\Gamma \vdash \langle While\ b\ c, Normal\ r \rangle \Rightarrow \notin(\{Stuck\} \cup Fault\ '(-F))$ 
    by (auto simp add: final-notin-def intro: exec.intros)
  from hyp [OF e-in-b this] obtain
    cNoFault:  $\Gamma \vdash \langle c, Normal\ e \rangle \Rightarrow \notin(\{Stuck\} \cup Fault\ '(-F))$  and
    Abrupt-r:  $\forall u. \Gamma \vdash \langle c, Normal\ e \rangle \Rightarrow Abrupt\ u \longrightarrow$ 
       $\Gamma \vdash \langle While\ b\ c, Normal\ r \rangle \Rightarrow Abrupt\ u$ 
  by simp

  {
    fix u assume  $\Gamma \vdash \langle c, Normal\ e \rangle \Rightarrow Abrupt\ u$ 
    with Abrupt-r have  $\Gamma \vdash \langle While\ b\ c, Normal\ r \rangle \Rightarrow Abrupt\ u$  by simp
    moreover from Z-r obtain
       $Z \in b\ \Gamma \vdash \langle c, Normal\ Z \rangle \Rightarrow Normal\ r$ 
    by simp
    ultimately have  $\Gamma \vdash \langle While\ b\ c, Normal\ Z \rangle \Rightarrow Abrupt\ u$ 
  }
}

```

```

      by (blast intro: exec.intros)
    }
    with cNoFault show ?Prop Z e
      by iprover
  qed
}
with P show s ∈ ?P' s
  by blast
next
{
  fix t
  assume termination: t ∉ b
  assume (Z, t) ∈ ?unroll
  hence  $\Gamma \vdash \langle \text{While } b \ c, \text{Normal } Z \rangle \Rightarrow \text{Normal } t$ 
  proof (induct rule: converse-rtrancl-induct [consumes 1])
    from termination
    show  $\Gamma \vdash \langle \text{While } b \ c, \text{Normal } t \rangle \Rightarrow \text{Normal } t$ 
      by (blast intro: exec.WhileFalse)
  next
  fix Z r
  assume first-body:
     $(Z, r) \in \{(s, t). s \in b \wedge \Gamma \vdash \langle c, \text{Normal } s \rangle \Rightarrow \text{Normal } t\}$ 
  assume (r, t) ∈ ?unroll
  assume rest-loop:  $\Gamma \vdash \langle \text{While } b \ c, \text{Normal } r \rangle \Rightarrow \text{Normal } t$ 
  show  $\Gamma \vdash \langle \text{While } b \ c, \text{Normal } Z \rangle \Rightarrow \text{Normal } t$ 
  proof -
    from first-body obtain
       $Z \in b \ \Gamma \vdash \langle c, \text{Normal } Z \rangle \Rightarrow \text{Normal } r$ 
    by fast
    moreover
    from rest-loop have
       $\Gamma \vdash \langle \text{While } b \ c, \text{Normal } r \rangle \Rightarrow \text{Normal } t$ 
    by fast
    ultimately show  $\Gamma \vdash \langle \text{While } b \ c, \text{Normal } Z \rangle \Rightarrow \text{Normal } t$ 
      by (rule exec.WhileTrue)
  qed
  qed
}
with P
show  $(\forall t. t \in (?P' s \cap - b) \longrightarrow t \in \{t. \Gamma \vdash \langle \text{While } b \ c, \text{Normal } Z \rangle \Rightarrow \text{Normal } t\})$ 
  by blast
next
from P show  $\forall t. t \in ?A' s \longrightarrow t \in ?A' Z$  by simp
qed
qed
next
case (Call p)
let ?P =  $\{s. s = Z \wedge \Gamma \vdash \langle \text{Call } p, \text{Normal } s \rangle \Rightarrow \neg(\{\text{Stuck}\} \cup \text{Fault } '(-F))\}$ 

```

**from** *noStuck-Call* **have**  $\forall s \in ?P. p \in \text{dom } \Gamma$   
**by** (*fastforce simp add: final-notin-def*)  
**then show**  $\Gamma, \Theta \vdash /_F ?P$  (*Call p*)  
 $\{t. \Gamma \vdash \langle \text{Call } p, \text{Normal } Z \rangle \Rightarrow \text{Normal } t\},$   
 $\{t. \Gamma \vdash \langle \text{Call } p, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\}$   
**proof** (*rule conseq-extract-state-indep-prop*)  
**assume** *p-defined*:  $p \in \text{dom } \Gamma$   
**with** *MGT-Calls* **show**  
 $\Gamma, \Theta \vdash /_F \{s. s = Z \wedge$   
 $\Gamma \vdash \langle \text{Call } p, \text{Normal } s \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F))\}$   
 $(\text{Call } p)$   
 $\{t. \Gamma \vdash \langle \text{Call } p, \text{Normal } Z \rangle \Rightarrow \text{Normal } t\},$   
 $\{t. \Gamma \vdash \langle \text{Call } p, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\}$   
**by** (*auto*)  
**qed**  
**next**  
**case** (*DynCom c*)  
**have** *hyp*:  
 $\bigwedge s'. \forall Z. \Gamma, \Theta \vdash /_F \{s. s = Z \wedge \Gamma \vdash \langle c \ s', \text{Normal } s \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F))\}$   
 $c \ s'$   
 $\{t. \Gamma \vdash \langle c \ s', \text{Normal } Z \rangle \Rightarrow \text{Normal } t\}, \{t. \Gamma \vdash \langle c \ s', \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\}$   
**using** *DynCom* **by** *simp*  
**have** *hyp'*:  
 $\Gamma, \Theta \vdash /_F \{s. s = Z \wedge \Gamma \vdash \langle \text{DynCom } c, \text{Normal } s \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F))\}$   $c \ Z$   
 $\{t. \Gamma \vdash \langle \text{DynCom } c, \text{Normal } Z \rangle \Rightarrow \text{Normal } t\}, \{t. \Gamma \vdash \langle \text{DynCom } c, \text{Normal } Z \rangle \Rightarrow$   
 $\text{Abrupt } t\}$   
**by** (*rule ConseqMGT [OF hyp]*)  
*(fastforce simp add: final-notin-def intro: exec.intros)*  
**show**  $\Gamma, \Theta \vdash /_F \{s. s = Z \wedge \Gamma \vdash \langle \text{DynCom } c, \text{Normal } s \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F))\}$   
 $\text{DynCom } c$   
 $\{t. \Gamma \vdash \langle \text{DynCom } c, \text{Normal } Z \rangle \Rightarrow \text{Normal } t\},$   
 $\{t. \Gamma \vdash \langle \text{DynCom } c, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\}$   
**apply** (*rule hoarep.DynCom*)  
**apply** (*clarsimp*)  
**apply** (*rule hyp' [simplified]*)  
**done**  
**next**  
**case** (*Guard f g c*)  
**have** *hyp-c*:  $\forall Z. \Gamma, \Theta \vdash /_F \{s. s = Z \wedge \Gamma \vdash \langle c, \text{Normal } s \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F))\}$   $c$   
 $\{t. \Gamma \vdash \langle c, \text{Normal } Z \rangle \Rightarrow \text{Normal } t\},$   
 $\{t. \Gamma \vdash \langle c, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\}$   
**using** *Guard* **by** *iprover*  
**show** *?case*  
**proof** (*cases f \in F*)  
**case** *True*  
**from** *hyp-c*  
**have**  $\Gamma, \Theta \vdash /_F (g \cap \{s. s = Z \wedge$

```

       $\Gamma \langle \text{Guard } f \ g \ c, \text{Normal } s \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F))\}$ 
       $\begin{matrix} c \\ \{t. \Gamma \langle \text{Guard } f \ g \ c, \text{Normal } Z \rangle \Rightarrow \text{Normal } t\}, \\ \{t. \Gamma \langle \text{Guard } f \ g \ c, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\} \end{matrix}$ 
apply (rule ConseqMGT)
apply (insert True)
apply (auto simp add: final-notin-def intro: exec.intros)
done
from True this
show ?thesis
  by (rule conseqPre [OF Guarantee]) auto
next
case False
from hyp-c
have  $\Gamma, \Theta \vdash /_F$ 
   $(g \cap \{s. s = Z \wedge \Gamma \langle \text{Guard } f \ g \ c, \text{Normal } s \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F))\})$ 
   $\begin{matrix} c \\ \{t. \Gamma \langle \text{Guard } f \ g \ c, \text{Normal } Z \rangle \Rightarrow \text{Normal } t\}, \\ \{t. \Gamma \langle \text{Guard } f \ g \ c, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\} \end{matrix}$ 
apply (rule ConseqMGT)
apply clarify
apply (frule Guard-noFaultStuckD [OF - False])
apply (auto simp add: final-notin-def intro: exec.intros)
done
then show ?thesis
  apply (rule conseqPre [OF hoarep.Guard])
  apply clarify
  apply (frule Guard-noFaultStuckD [OF - False])
  apply auto
done
qed
next
case Throw
show  $\Gamma, \Theta \vdash /_F \{s. s = Z \wedge \Gamma \langle \text{Throw}, \text{Normal } s \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F))\}$ 
Throw
   $\begin{matrix} \{t. \Gamma \langle \text{Throw}, \text{Normal } Z \rangle \Rightarrow \text{Normal } t\}, \\ \{t. \Gamma \langle \text{Throw}, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\} \end{matrix}$ 
  by (rule conseqPre [OF hoarep.Throw]) (blast intro: exec.intros)
next
case (Catch  $c_1$   $c_2$ )
have  $\forall Z. \Gamma, \Theta \vdash /_F \{s. s = Z \wedge \Gamma \langle c_1, \text{Normal } s \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F))\}$ 
 $c_1$ 
   $\begin{matrix} \{t. \Gamma \langle c_1, \text{Normal } Z \rangle \Rightarrow \text{Normal } t\}, \\ \{t. \Gamma \langle c_1, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\} \end{matrix}$ 
  using Catch.hyps by iprover
hence  $\Gamma, \Theta \vdash /_F \{s. s = Z \wedge \Gamma \langle \text{Catch } c_1 \ c_2, \text{Normal } s \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F))\}$ 
 $c_1$ 
   $\begin{matrix} \{t. \Gamma \langle \text{Catch } c_1 \ c_2, \text{Normal } Z \rangle \Rightarrow \text{Normal } t\}, \\ \{t. \Gamma \langle c_1, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t \wedge \end{matrix}$ 

```



```

 $\forall p \in \text{dom } \Gamma. \forall Z. \Gamma, ?\Theta \vdash_{/F}$ 
  {s. s=Z  $\wedge$   $\Gamma \vdash \langle \text{Call } p, \text{Normal } s \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F))\}$ 
  (Call p)
  {t.  $\Gamma \vdash \langle \text{Call } p, \text{Normal } Z \rangle \Rightarrow \text{Normal } t$ },
  {t.  $\Gamma \vdash \langle \text{Call } p, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t$ }
  by (intro ballI allI, rule HoarePartialDef.Asm, auto)
  have  $\forall Z. \Gamma, ?\Theta \vdash_{/F}$  {s. s=Z  $\wedge$   $\Gamma \vdash \langle \text{the } (\Gamma \ p) , \text{Normal } s \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup$ 
Fault'(-F))}
      (the ( $\Gamma \ p$ ))
      {t.  $\Gamma \vdash \langle \text{the } (\Gamma \ p), \text{Normal } Z \rangle \Rightarrow \text{Normal } t$ },
      {t.  $\Gamma \vdash \langle \text{the } (\Gamma \ p), \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t$ }
      by (iprover intro: MGT-lemma [OF MGT-Calls])
  thus  $\Gamma, ?\Theta \vdash_{/F}$  (?Pre p Z) (the ( $\Gamma \ p$ )) (?Post p Z), (?Abr p Z)
  apply (rule ConseqMGT)
  apply (clarify, safe)
  proof -
  assume  $\Gamma \vdash \langle \text{Call } p, \text{Normal } Z \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F))$ 
  with defined show  $\Gamma \vdash \langle \text{the } (\Gamma \ p), \text{Normal } Z \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F))$ 
    by (fastforce simp add: final-notin-def
        intro: exec.intros)
  next
  fix t
  assume  $\Gamma \vdash \langle \text{the } (\Gamma \ p), \text{Normal } Z \rangle \Rightarrow \text{Normal } t$ 
  with defined
  show  $\Gamma \vdash \langle \text{Call } p, \text{Normal } Z \rangle \Rightarrow \text{Normal } t$ 
    by (auto intro: exec.Call)
  next
  fix t
  assume  $\Gamma \vdash \langle \text{the } (\Gamma \ p), \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t$ 
  with defined
  show  $\Gamma \vdash \langle \text{Call } p, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t$ 
    by (auto intro: exec.Call)
  qed
  qed
}
then show ?thesis
  apply -
  apply (intro ballI allI)
  apply (rule CallRec' [where Procs=dom  $\Gamma$  and
    P= $\lambda p \ Z. \{s. s=Z \wedge$ 
       $\Gamma \vdash \langle \text{Call } p, \text{Normal } s \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F))\}$  and
    Q= $\lambda p \ Z. \{t. \Gamma \vdash \langle \text{Call } p, \text{Normal } Z \rangle \Rightarrow \text{Normal } t\}$  and
    A= $\lambda p \ Z. \{t. \Gamma \vdash \langle \text{Call } p, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\}$ ])
  apply simp+
  done
qed

```



**theorem** *hoare-complete*:  $\Gamma \models_{/F} P \text{ c } Q, A \implies \Gamma, \{\} \vdash_{/F} P \text{ c } Q, A$   
 by (*iprover intro: MGT-implies-complete MGT-lemma [OF MGT-Calls]*)

**lemma** *hoare-complete'*:  
**assumes** *cvalid*:  $\forall n. \Gamma, \Theta \models n:_{/F} P \text{ c } Q, A$   
**shows**  $\Gamma, \Theta \vdash_{/F} P \text{ c } Q, A$   
**proof** (*cases*  $\Gamma \models_{/F} P \text{ c } Q, A$ )  
 case *True*  
 hence  $\Gamma, \{\} \vdash_{/F} P \text{ c } Q, A$   
 by (*rule hoare-complete*)  
 thus  $\Gamma, \Theta \vdash_{/F} P \text{ c } Q, A$   
 by (*rule hoare-augment-context simp*)  
**next**  
 case *False*  
**with** *cvalid*  
**show** *?thesis*  
 by (*rule ExFalso*)  
**qed**

**lemma** *hoare-strip- $\Gamma$* :  
**assumes** *deriv*:  $\Gamma, \{\} \vdash_{/F} P \text{ p } Q, A$   
**assumes** *F'*:  $F' \subseteq -F$   
**shows** *strip*  $F' \Gamma, \{\} \vdash_{/F} P \text{ p } Q, A$   
**proof** (*rule hoare-complete*)  
**from** *hoare-sound [OF deriv]* **have**  $\Gamma \models_{/F} P \text{ p } Q, A$   
 by (*simp add: cvalid-def*)  
**from** *this F'*  
**show** *strip*  $F' \Gamma \models_{/F} P \text{ p } Q, A$   
 by (*rule valid-to-valid-strip*)  
**qed**

## 5.3 And Now: Some Useful Rules

### 5.3.1 Consequence

**lemma** *LiberalConseq-sound*:  
**fixes** *F::'f set*  
**assumes** *cons*:  $\forall s \in P. \forall (t::('s, 'f) \text{ xstate}). \exists P' Q' A'. (\forall n. \Gamma, \Theta \models n:_{/F} P' \text{ c } Q', A')$   
 $\wedge$   
 $((s \in P' \longrightarrow t \in \text{Normal } \langle Q' \cup \text{Abrupt } \langle A' \rangle \rangle$   
 $\longrightarrow t \in \text{Normal } \langle Q \cup \text{Abrupt } \langle A \rangle \rangle)$   
**shows**  $\Gamma, \Theta \models n:_{/F} P \text{ c } Q, A$   
**proof** (*rule cvalidI*)  
**fix** *s t*  
**assume** *ctxt*:  $\forall (P, p, Q, A) \in \Theta. \Gamma \models n:_{/F} P (\text{Call } p) Q, A$   
**assume** *exec*:  $\Gamma \vdash \langle c, \text{Normal } s \rangle = n \Rightarrow t$   
**assume** *P*:  $s \in P$

**assume**  $t\text{-notin-}F: t \notin \text{Fault} \text{ ' } F$   
**show**  $t \in \text{Normal} \text{ ' } Q \cup \text{Abrupt} \text{ ' } A$   
**proof** –  
**from**  $P$  **cons obtain**  $P' Q' A'$  **where**  
 $\text{spec}: \forall n. \Gamma, \Theta \models n: /_F P' c Q', A'$  **and**  
 $\text{adapt}: (s \in P' \longrightarrow t \in \text{Normal} \text{ ' } Q' \cup \text{Abrupt} \text{ ' } A')$   
 $\longrightarrow t \in \text{Normal} \text{ ' } Q \cup \text{Abrupt} \text{ ' } A$   
**apply** –  
**apply** (*drule* (1) *bspec*)  
**apply** (*erule-tac*  $x=t$  **in** *allE*)  
**apply** (*elim* *exE* *conjE*)  
**apply** *iprover*  
**done**  
**from** *exec spec ctxt t-notin-F*  
**have**  $s \in P' \longrightarrow t \in \text{Normal} \text{ ' } Q' \cup \text{Abrupt} \text{ ' } A'$   
**by** (*simp add: cinvalid-def nvalid-def*)  
**with** *adapt* **show** *?thesis*  
**by** *simp*  
**qed**  
**qed**

**lemma** *LiberalConseq*:  
**fixes**  $F:: 'f \text{ set}$   
**assumes**  $\text{cons}: \forall s \in P. \forall (t::('s, 'f) \text{ xstate}). \exists P' Q' A'. \Gamma, \Theta \vdash /_F P' c Q', A' \wedge$   
 $((s \in P' \longrightarrow t \in \text{Normal} \text{ ' } Q' \cup \text{Abrupt} \text{ ' } A')$   
 $\longrightarrow t \in \text{Normal} \text{ ' } Q \cup \text{Abrupt} \text{ ' } A)$   
**shows**  $\Gamma, \Theta \vdash /_F P c Q, A$   
**apply** (*rule hoare-complete'*)  
**apply** (*rule allI*)  
**apply** (*rule LiberalConseq-sound*)  
**using** *cons*  
**apply** (*clarify*)  
**apply** (*drule* (1) *bspec*)  
**apply** (*erule-tac*  $x=t$  **in** *allE*)  
**apply** *clarify*  
**apply** (*rule-tac*  $x=P'$  **in** *exI*)  
**apply** (*rule-tac*  $x=Q'$  **in** *exI*)  
**apply** (*rule-tac*  $x=A'$  **in** *exI*)  
**apply** (*rule conjI*)  
**apply** (*blast intro: hoare-cinvalid*)  
**apply** *assumption*  
**done**

**lemma**  $\forall s \in P. \exists P' Q' A'. \Gamma, \Theta \vdash /_F P' c Q', A' \wedge s \in P' \wedge Q' \subseteq Q \wedge A' \subseteq A$   
 $\implies \Gamma, \Theta \vdash /_F P c Q, A$   
**apply** (*rule LiberalConseq*)  
**apply** (*rule ballI*)  
**apply** (*drule* (1) *bspec*)

**apply** *clarify*  
**apply** (*rule-tac*  $x=P'$  **in**  $exI$ )  
**apply** (*rule-tac*  $x=Q'$  **in**  $exI$ )  
**apply** (*rule-tac*  $x=A'$  **in**  $exI$ )  
**apply** *auto*  
**done**

**lemma**

**fixes**  $F:: 'f$  set

**assumes** *cons*:  $\forall s \in P. \exists P' Q' A'. \Gamma, \Theta \vdash_{/F} P' c Q', A' \wedge$

$(\forall (t::('s, 'f) \text{ xstate}). (s \in P' \longrightarrow t \in \text{Normal } ' Q' \cup \text{Abrupt } ' A')$   
 $\longrightarrow t \in \text{Normal } ' Q \cup \text{Abrupt } ' A)$

**shows**  $\Gamma, \Theta \vdash_{/F} P c Q, A$

**apply** (*rule* *Conseq*)  
**apply** (*rule* *ballI*)  
**apply** (*insert* *cons*)  
**apply** (*drule* (1) *bspec*)  
**apply** *clarify*  
**apply** (*rule-tac*  $x=P'$  **in**  $exI$ )  
**apply** (*rule-tac*  $x=Q'$  **in**  $exI$ )  
**apply** (*rule-tac*  $x=A'$  **in**  $exI$ )  
**apply** (*rule* *conjI*)  
**apply** *assumption*

**oops**

**lemma** *LiberalConseq'*:

**fixes**  $F:: 'f$  set

**assumes** *cons*:  $\forall s \in P. \exists P' Q' A'. \Gamma, \Theta \vdash_{/F} P' c Q', A' \wedge$

$(\forall (t::('s, 'f) \text{ xstate}). (s \in P' \longrightarrow t \in \text{Normal } ' Q' \cup \text{Abrupt } ' A')$   
 $\longrightarrow t \in \text{Normal } ' Q \cup \text{Abrupt } ' A)$

**shows**  $\Gamma, \Theta \vdash_{/F} P c Q, A$

**apply** (*rule* *LiberalConseq*)  
**apply** (*rule* *ballI*)  
**apply** (*rule* *allI*)  
**apply** (*insert* *cons*)  
**apply** (*drule* (1) *bspec*)  
**apply** *clarify*  
**apply** (*rule-tac*  $x=P'$  **in**  $exI$ )  
**apply** (*rule-tac*  $x=Q'$  **in**  $exI$ )  
**apply** (*rule-tac*  $x=A'$  **in**  $exI$ )  
**apply** *iprover*  
**done**

**lemma** *LiberalConseq''*:

**fixes**  $F:: 'f$  set

**assumes** *spec*:  $\forall Z. \Gamma, \Theta \vdash_{/F} (P' Z) c (Q' Z), (A' Z)$

**assumes** *cons*:  $\forall s (t::('s, 'f) \text{ xstate}).$

$$\begin{aligned}
& (\forall Z. s \in P' Z \longrightarrow t \in \text{Normal } ' Q' Z \cup \text{Abrupt } ' A' Z) \\
& \longrightarrow (s \in P \longrightarrow t \in \text{Normal } ' Q \cup \text{Abrupt } ' A)
\end{aligned}$$

**shows**  $\Gamma, \Theta \vdash_{/F} P \text{ c } Q, A$   
**apply** (rule LiberalConseq)  
**apply** (rule ballI)  
**apply** (rule allI)  
**apply** (insert cons)  
**apply** (erule-tac x=s in allE)  
**apply** (erule-tac x=t in allE)  
**apply** (case-tac t  $\in$  Normal ' Q  $\cup$  Abrupt ' A)  
**apply** (insert spec)  
**apply** iprover  
**apply** auto  
**done**

**primrec** *procs*:: ('s,'p,'f) com  $\Rightarrow$  'p set  
**where**  
*procs* Skip = {} |  
*procs* (Basic f) = {} |  
*procs* (Seq c<sub>1</sub> c<sub>2</sub>) = (procs c<sub>1</sub>  $\cup$  procs c<sub>2</sub>) |  
*procs* (Cond b c<sub>1</sub> c<sub>2</sub>) = (procs c<sub>1</sub>  $\cup$  procs c<sub>2</sub>) |  
*procs* (While b c) = procs c |  
*procs* (Call p) = {p} |  
*procs* (DynCom c) = ( $\bigcup$  s. procs (c s)) |  
*procs* (Guard f g c) = procs c |  
*procs* Throw = {} |  
*procs* (Catch c<sub>1</sub> c<sub>2</sub>) = (procs c<sub>1</sub>  $\cup$  procs c<sub>2</sub>)

**primrec** *noSpec*:: ('s,'p,'f) com  $\Rightarrow$  bool  
**where**  
*noSpec* Skip = True |  
*noSpec* (Basic f) = True |  
*noSpec* (Spec r) = False |  
*noSpec* (Seq c<sub>1</sub> c<sub>2</sub>) = (noSpec c<sub>1</sub>  $\wedge$  noSpec c<sub>2</sub>) |  
*noSpec* (Cond b c<sub>1</sub> c<sub>2</sub>) = (noSpec c<sub>1</sub>  $\wedge$  noSpec c<sub>2</sub>) |  
*noSpec* (While b c) = noSpec c |  
*noSpec* (Call p) = True |  
*noSpec* (DynCom c) = ( $\forall$  s. noSpec (c s)) |  
*noSpec* (Guard f g c) = noSpec c |  
*noSpec* Throw = True |  
*noSpec* (Catch c<sub>1</sub> c<sub>2</sub>) = (noSpec c<sub>1</sub>  $\wedge$  noSpec c<sub>2</sub>)

**lemma** *exec-noSpec-no-Stuck*:  
**assumes** *exec*:  $\Gamma \vdash \langle c, s \rangle \Rightarrow t$   
**assumes** *noSpec-c*: noSpec c  
**assumes** *noSpec- $\Gamma$* :  $\forall p \in \text{dom } \Gamma. \text{noSpec } (\text{the } (\Gamma \ p))$   
**assumes** *procs-subset*: procs c  $\subseteq$  dom  $\Gamma$   
**assumes** *procs-subset- $\Gamma$* :  $\forall p \in \text{dom } \Gamma. \text{procs } (\text{the } (\Gamma \ p)) \subseteq \text{dom } \Gamma$   
**assumes** *s-no-Stuck*: s  $\neq$  Stuck

**shows**  $t \neq \text{Stuck}$   
**using** *exec noSpec-c procs-subset s-no-Stuck* **proof** *induct*  
   **case** (*Call p bdy s t*) **with** *noSpec- $\Gamma$  procs-subset- $\Gamma$*  **show** *?case*  
     **by** (*auto dest!: bspec [of - - p]*)  
**next**  
   **case** (*DynCom c s t*) **then show** *?case*  
     **by** *auto blast*  
**qed** *auto*

**lemma** *execn-noSpec-no-Stuck*:  
**assumes** *exec:  $\Gamma \vdash \langle c, s \rangle = n \Rightarrow t$*   
**assumes** *noSpec-c: noSpec c*  
**assumes** *noSpec- $\Gamma$ :  $\forall p \in \text{dom } \Gamma. \text{noSpec } (\text{the } (\Gamma p))$*   
**assumes** *procs-subset: procs c  $\subseteq$  dom  $\Gamma$*   
**assumes** *procs-subset- $\Gamma$ :  $\forall p \in \text{dom } \Gamma. \text{procs } (\text{the } (\Gamma p)) \subseteq \text{dom } \Gamma$*   
**assumes** *s-no-Stuck: s  $\neq$  Stuck*  
**shows**  $t \neq \text{Stuck}$   
**using** *exec noSpec-c procs-subset s-no-Stuck* **proof** *induct*  
   **case** (*Call p bdy n s t*) **with** *noSpec- $\Gamma$  procs-subset- $\Gamma$*  **show** *?case*  
     **by** (*auto dest!: bspec [of - - p]*)  
**next**  
   **case** (*DynCom c s t*) **then show** *?case*  
     **by** *auto blast*  
**qed** *auto*

**lemma** *LiberalConseq-noguards-nothrows-sound*:  
**assumes** *spec:  $\forall Z. \forall n. \Gamma, \Theta \models n: /_F (P' Z) c (Q' Z), (A' Z)$*   
**assumes** *cons:  $\forall s t. (\forall Z. s \in P' Z \longrightarrow t \in Q' Z) \longrightarrow (s \in P \longrightarrow t \in Q)$*   
**assumes** *noguards-c: noguards c*  
**assumes** *noguards- $\Gamma$ :  $\forall p \in \text{dom } \Gamma. \text{noguards } (\text{the } (\Gamma p))$*   
**assumes** *nothrows-c: nothrows c*  
**assumes** *nothrows- $\Gamma$ :  $\forall p \in \text{dom } \Gamma. \text{nothrows } (\text{the } (\Gamma p))$*   
**assumes** *noSpec-c: noSpec c*  
**assumes** *noSpec- $\Gamma$ :  $\forall p \in \text{dom } \Gamma. \text{noSpec } (\text{the } (\Gamma p))$*   
**assumes** *procs-subset: procs c  $\subseteq$  dom  $\Gamma$*   
**assumes** *procs-subset- $\Gamma$ :  $\forall p \in \text{dom } \Gamma. \text{procs } (\text{the } (\Gamma p)) \subseteq \text{dom } \Gamma$*   
**shows**  $\Gamma, \Theta \models n: /_F P c Q, A$   
**proof** (*rule cinvalidI*)  
   **fix** *s t*  
   **assume** *ctxt:  $\forall (P, p, Q, A) \in \Theta. \Gamma \models n: /_F P (Call p) Q, A$*   
   **assume** *exec:  $\Gamma \vdash \langle c, \text{Normal } s \rangle = n \Rightarrow t$*   
   **assume** *P: s  $\in$  P*  
   **assume** *t-notin-F: t  $\notin$  Fault ' F*  
   **show** *t  $\in$  Normal ' Q  $\cup$  Abrupt ' A*  
   **proof** –  
     **from** *execn-noguards-no-Fault [OF exec noguards-c noguards- $\Gamma$ ]*  
     *execn-nothrows-no-Abrupt [OF exec nothrows-c nothrows- $\Gamma$ ]*  
     *execn-noSpec-no-Stuck [OF exec*

$noSpec-c$   $noSpec-\Gamma$   $procs-subset$   
 $procs-subset-\Gamma]$   
**obtain**  $t'$  **where**  $t: t=Normal\ t'$   
**by**  $(cases\ t)\ auto$   
**with**  $exec\ spec\ ctxt$   
**have**  $(\forall Z. s \in P' Z \longrightarrow t' \in Q' Z)$   
**by**  $(unfold\ cinvalid-def\ nvalid-def)\ blast$   
**with**  $cons\ P\ t$  **show**  $?thesis$   
**by**  $simp$   
**qed**  
**qed**

**lemma** *LiberalConseq-noguards-nothrows:*  
**assumes**  $spec: \forall Z. \Gamma, \Theta \vdash /_F (P' Z) c (Q' Z), (A' Z)$   
**assumes**  $cons: \forall s t. (\forall Z. s \in P' Z \longrightarrow t \in Q' Z)$   
 $\longrightarrow (s \in P \longrightarrow t \in Q)$   
**assumes**  $noguards-c: noguards\ c$   
**assumes**  $noguards-\Gamma: \forall p \in dom\ \Gamma. noguards\ (the\ (\Gamma\ p))$   
**assumes**  $nothrows-c: nothrows\ c$   
**assumes**  $nothrows-\Gamma: \forall p \in dom\ \Gamma. nothrows\ (the\ (\Gamma\ p))$   
**assumes**  $noSpec-c: noSpec\ c$   
**assumes**  $noSpec-\Gamma: \forall p \in dom\ \Gamma. noSpec\ (the\ (\Gamma\ p))$   
**assumes**  $procs-subset: procs\ c \subseteq dom\ \Gamma$   
**assumes**  $procs-subset-\Gamma: \forall p \in dom\ \Gamma. procs\ (the\ (\Gamma\ p)) \subseteq dom\ \Gamma$   
**shows**  $\Gamma, \Theta \vdash /_F P\ c\ Q, A$   
**apply**  $(rule\ hoare-complete')$   
**apply**  $(rule\ allI)$   
**apply**  $(rule\ LiberalConseq-noguards-nothrows-sound$   
 $[OF\ -\ cons\ noguards-c\ noguards-\Gamma\ nothrows-c\ nothrows-\Gamma$   
 $noSpec-c\ noSpec-\Gamma$   
 $procs-subset\ procs-subset-\Gamma])$   
**apply**  $(insert\ spec)$   
**apply**  $(intro\ allI)$   
**apply**  $(erule-tac\ x=Z\ in\ allE)$   
**by**  $(rule\ hoare-cinvalid)$

**lemma**  
**assumes**  $spec: \forall Z. \Gamma, \Theta \vdash /_F \{s. s=fst\ Z \wedge P\ s\ (snd\ Z)\} c \{t. Q\ (fst\ Z)\ (snd\ Z)$   
 $t\}, \{\}$   
**assumes**  $noguards-c: noguards\ c$   
**assumes**  $noguards-\Gamma: \forall p \in dom\ \Gamma. noguards\ (the\ (\Gamma\ p))$   
**assumes**  $nothrows-c: nothrows\ c$   
**assumes**  $nothrows-\Gamma: \forall p \in dom\ \Gamma. nothrows\ (the\ (\Gamma\ p))$   
**assumes**  $noSpec-c: noSpec\ c$   
**assumes**  $noSpec-\Gamma: \forall p \in dom\ \Gamma. noSpec\ (the\ (\Gamma\ p))$   
**assumes**  $procs-subset: procs\ c \subseteq dom\ \Gamma$   
**assumes**  $procs-subset-\Gamma: \forall p \in dom\ \Gamma. procs\ (the\ (\Gamma\ p)) \subseteq dom\ \Gamma$   
**shows**  $\forall \sigma. \Gamma, \Theta \vdash /_F \{s. s=\sigma\} c \{t. \forall l. P\ \sigma\ l \longrightarrow Q\ \sigma\ l\ t\}, \{\}$

**apply** (*rule allI*)  
**apply** (*rule LiberalConseq-noguards-nothrows*  
 $[OF\ spec - noguards-c\ noguards-\Gamma\ nothrows-c\ nothrows-\Gamma$   
 $noSpec-c\ noSpec-\Gamma$   
 $procs-subset\ procs-subset-\Gamma]$ )  
**apply auto**  
**done**

### 5.3.2 Modify Return

**lemma** *Proc-exnModifyReturn-sound*:

**assumes** *valid-call*:  $\forall n. \Gamma, \Theta \models_n: /_F P\ call\text{-}exn\ init\ p\ return'\ result\text{-}exn\ c\ Q, A$   
**assumes** *valid-modif*:  
 $\forall \sigma. \forall n. \Gamma, \Theta \models_n: /_{UNIV} \{\sigma\}\ Call\ p\ (Modif\ \sigma), (ModifAbr\ \sigma)$   
**assumes** *ret-modif*:  
 $\forall s\ t. t \in Modif\ (init\ s)$   
 $\longrightarrow return'\ s\ t = return\ s\ t$   
**assumes** *ret-modifAbr*:  $\forall s\ t. t \in ModifAbr\ (init\ s)$   
 $\longrightarrow result\text{-}exn\ (return'\ s\ t)\ t = result\text{-}exn\ (return\ s\ t)\ t$   
**shows**  $\Gamma, \Theta \models_n: /_F P\ (call\text{-}exn\ init\ p\ return\ result\text{-}exn\ c)\ Q, A$   
**proof** (*rule cinvalidI*)

**fix**  $s\ t$

**assume** *ctxt*:  $\forall (P, p, Q, A) \in \Theta. \Gamma \models_n: /_F P\ (Call\ p)\ Q, A$

**then have** *ctxt'*:  $\forall (P, p, Q, A) \in \Theta. \Gamma \models_n: /_{UNIV} P\ (Call\ p)\ Q, A$

**by** (*auto intro: nvalid-augment-Faults*)

**assume** *exec*:  $\Gamma \vdash \langle call\text{-}exn\ init\ p\ return\ result\text{-}exn\ c, Normal\ s \rangle =n \Rightarrow t$

**assume** *P*:  $s \in P$

**assume** *t-notin-F*:  $t \notin Fault\ 'F$

**from** *exec*

**show**  $t \in Normal\ 'Q \cup Abrupt\ 'A$

**proof** (*cases rule: execn-call-exn-Normal-elim*)

**fix** *bdy m t'*

**assume** *bdy*:  $\Gamma\ p = Some\ bdy$

**assume** *exec-body*:  $\Gamma \vdash \langle bdy, Normal\ (init\ s) \rangle =m \Rightarrow Normal\ t'$

**assume** *exec-c*:  $\Gamma \vdash \langle c\ s\ t', Normal\ (return\ s\ t') \rangle =Suc\ m \Rightarrow t$

**assume** *n*:  $n = Suc\ m$

**from** *exec-body n bdy*

**have**  $\Gamma \vdash \langle Call\ p, Normal\ (init\ s) \rangle =n \Rightarrow Normal\ t'$

**by** (*auto simp add: intro: execn.Call*)

**from** *cnvalidD [OF valid-modif [rule-format, of n init s] ctxt' this] P*

**have**  $t' \in Modif\ (init\ s)$

**by** *auto*

**with** *ret-modif* **have**  $Normal\ (return'\ s\ t') =$

$Normal\ (return\ s\ t')$

**by** *simp*

**with** *exec-body exec-c bdy n*

**have**  $\Gamma \vdash \langle call\text{-}exn\ init\ p\ return'\ result\text{-}exn\ c, Normal\ s \rangle =n \Rightarrow t$

**by** (*auto intro: execn-call-exn*)

**from** *cnvalidD [OF valid-call [rule-format] ctxt this] P t-notin-F*

```

show ?thesis
  by simp
next
  fix bdy m t'
  assume bdy:  $\Gamma p = \text{Some } bdy$ 
  assume exec-body:  $\Gamma \langle bdy, \text{Normal } (init\ s) \rangle = m \Rightarrow \text{Abrupt } t'$ 
  assume n:  $n = \text{Suc } m$ 
  assume t:  $t = \text{Abrupt } (\text{result-exn } (\text{return } s\ t')\ t')$ 
  also from exec-body n bdy
  have  $\Gamma \langle \text{Call } p, \text{Normal } (init\ s) \rangle = n \Rightarrow \text{Abrupt } t'$ 
    by (auto simp add: intro: execn.intros)
  from cinvalidD [OF valid-modif [rule-format, of n init s] ctxt' this] P
  have  $t' \in \text{ModifAbr } (init\ s)$ 
    by auto
  with ret-modifAbr have  $\text{Abrupt } (\text{result-exn } (\text{return } s\ t')\ t') = \text{Abrupt } (\text{result-exn } (\text{return}'\ s\ t')\ t')$ 
    by simp
  finally have  $t = \text{Abrupt } (\text{result-exn } (\text{return}'\ s\ t')\ t')$  .
  with exec-body bdy n
  have  $\Gamma \langle \text{call-exn } init\ p\ \text{return}'\ \text{result-exn } c, \text{Normal } s \rangle = n \Rightarrow t$ 
    by (auto intro: execn-call-exnAbrupt)
  from cinvalidD [OF valid-call [rule-format] ctxt this] P t-notin-F
  show ?thesis
    by simp
next
  fix bdy m f
  assume bdy:  $\Gamma p = \text{Some } bdy$ 
  assume  $\Gamma \langle bdy, \text{Normal } (init\ s) \rangle = m \Rightarrow \text{Fault } f\ n = \text{Suc } m$ 
     $t = \text{Fault } f$ 
  with bdy have  $\Gamma \langle \text{call-exn } init\ p\ \text{return}'\ \text{result-exn } c, \text{Normal } s \rangle = n \Rightarrow t$ 
    by (auto intro: execn-call-exnFault)
  from valid-call [rule-format] ctxt this P t-notin-F
  show ?thesis
    by (rule cinvalidD)
next
  fix bdy m
  assume bdy:  $\Gamma p = \text{Some } bdy$ 
  assume  $\Gamma \langle bdy, \text{Normal } (init\ s) \rangle = m \Rightarrow \text{Stuck } n = \text{Suc } m$ 
     $t = \text{Stuck}$ 
  with bdy have  $\Gamma \langle \text{call-exn } init\ p\ \text{return}'\ \text{result-exn } c, \text{Normal } s \rangle = n \Rightarrow t$ 
    by (auto intro: execn-call-exnStuck)
  from valid-call [rule-format] ctxt this P t-notin-F
  show ?thesis
    by (rule cinvalidD)
next
  fix m
  assume  $\Gamma p = \text{None}$ 
  and  $n = \text{Suc } m\ t = \text{Stuck}$ 
  then have  $\Gamma \langle \text{call-exn } init\ p\ \text{return}'\ \text{result-exn } c, \text{Normal } s \rangle = n \Rightarrow t$ 

```



by (auto intro: execn-call-exnUndefined)  
 from valid-call [rule-format] ctxt this P t-notin-F  
 show ?thesis  
 by (rule cvalidD)  
 qed  
 qed

**lemma ProcModifyReturn-sound:**  
 assumes valid-call:  $\forall n. \Gamma, \Theta \models_{/F} P \text{ call init } p \text{ return}' c \ Q, A$   
 assumes valid-modif:  
 $\forall \sigma. \forall n. \Gamma, \Theta \models_{/UNIV} \{\sigma\} \text{ Call } p \ (\text{Modif } \sigma), (\text{ModifAbr } \sigma)$   
 assumes ret-modif:  
 $\forall s \ t. t \in \text{Modif } (\text{init } s)$   
 $\longrightarrow \text{return}' s \ t = \text{return } s \ t$   
 assumes ret-modifAbr:  $\forall s \ t. t \in \text{ModifAbr } (\text{init } s)$   
 $\longrightarrow \text{return}' s \ t = \text{return } s \ t$   
 shows  $\Gamma, \Theta \models_{/F} P \ (\text{call init } p \ \text{return } c) \ Q, A$   
 using valid-call valid-modif ret-modif ret-modifAbr  
 unfolding call-call-exn  
 by (rule Proc-exnModifyReturn-sound)

**lemma Proc-exnModifyReturn:**  
 assumes spec:  $\Gamma, \Theta \vdash_{/F} P \ (\text{call-exn init } p \ \text{return}' \ \text{result-exn } c) \ Q, A$   
 assumes result-conform:  
 $\forall s \ t. t \in \text{Modif } (\text{init } s) \longrightarrow (\text{return}' s \ t) = (\text{return } s \ t)$   
 assumes return-conform:  
 $\forall s \ t. t \in \text{ModifAbr } (\text{init } s)$   
 $\longrightarrow (\text{result-exn } (\text{return}' s \ t) \ t) = (\text{result-exn } (\text{return } s \ t) \ t)$   
 assumes modifies-spec:  
 $\forall \sigma. \Gamma, \Theta \vdash_{/UNIV} \{\sigma\} \text{ Call } p \ (\text{Modif } \sigma), (\text{ModifAbr } \sigma)$   
 shows  $\Gamma, \Theta \vdash_{/F} P \ (\text{call-exn init } p \ \text{return} \ \text{result-exn } c) \ Q, A$   
 apply (rule hoare-complete')  
 apply (rule allI)  
 apply (rule Proc-exnModifyReturn-sound  
 [where Modif=Modif and ModifAbr=ModifAbr,  
 OF - - result-conform return-conform] )  
 using spec  
 apply (blast intro: hoare-cvalid)  
 using modifies-spec  
 apply (blast intro: hoare-cvalid)  
 done

**lemma ProcModifyReturn:**  
 assumes spec:  $\Gamma, \Theta \vdash_{/F} P \ (\text{call init } p \ \text{return}' c) \ Q, A$   
 assumes result-conform:  
 $\forall s \ t. t \in \text{Modif } (\text{init } s) \longrightarrow (\text{return}' s \ t) = (\text{return } s \ t)$   
 assumes return-conform:  
 $\forall s \ t. t \in \text{ModifAbr } (\text{init } s)$

$\longrightarrow (\text{return}' s t) = (\text{return } s t)$

**assumes** *modifies-spec*:  
 $\forall \sigma. \Gamma, \Theta \vdash_{/UNIV} \{\sigma\} \text{ Call } p (\text{Modif } \sigma), (\text{ModifAbr } \sigma)$

**shows**  $\Gamma, \Theta \vdash_{/F} P (\text{call init } p \text{ return } c) Q, A$

**using** *spec result-conform return-conform modifies-spec*

**unfolding** *call-call-exn*

**by** (*rule Proc-exnModifyReturn*)

**lemma** *Proc-exnModifyReturnSameFaults-sound*:  
**assumes** *valid-call*:  $\forall n. \Gamma, \Theta \models n:_{/F} P \text{ call-exn init } p \text{ return}' \text{ result-exn } c Q, A$   
**assumes** *valid-modif*:  
 $\forall \sigma. \forall n. \Gamma, \Theta \models n:_{/F} \{\sigma\} \text{ Call } p (\text{Modif } \sigma), (\text{ModifAbr } \sigma)$   
**assumes** *ret-modif*:  
 $\forall s t. t \in \text{Modif } (\text{init } s)$   
 $\longrightarrow \text{return}' s t = \text{return } s t$   
**assumes** *ret-modifAbr*:  $\forall s t. t \in \text{ModifAbr } (\text{init } s)$   
 $\longrightarrow \text{result-exn } (\text{return}' s t) t = \text{result-exn } (\text{return } s t) t$   
**shows**  $\Gamma, \Theta \models n:_{/F} P (\text{call-exn init } p \text{ return result-exn } c) Q, A$

**proof** (*rule cinvalidI*)  
**fix**  $s t$   
**assume** *ctxt*:  $\forall (P, p, Q, A) \in \Theta. \Gamma \models n:_{/F} P (\text{Call } p) Q, A$   
**assume** *exec*:  $\Gamma \vdash \langle \text{call-exn init } p \text{ return result-exn } c, \text{Normal } s \rangle = n \Rightarrow t$   
**assume**  $P: s \in P$   
**assume** *t-notin-F*:  $t \notin \text{Fault } ' F$   
**from** *exec*  
**show**  $t \in \text{Normal } ' Q \cup \text{Abrupt } ' A$   
**proof** (*cases rule: execn-call-exn-Normal-elim*)  
**fix**  $\text{bdy } m t'$   
**assume** *bdy*:  $\Gamma p = \text{Some } \text{bdy}$   
**assume** *exec-body*:  $\Gamma \vdash \langle \text{bdy}, \text{Normal } (\text{init } s) \rangle = m \Rightarrow \text{Normal } t'$   
**assume** *exec-c*:  $\Gamma \vdash \langle c s t', \text{Normal } (\text{return } s t') \rangle = \text{Suc } m \Rightarrow t$   
**assume**  $n: n = \text{Suc } m$   
**from** *exec-body*  $n \text{ bdy}$   
**have**  $\Gamma \vdash \langle \text{Call } p, \text{Normal } (\text{init } s) \rangle = n \Rightarrow \text{Normal } t'$   
**by** (*auto simp add: intro: execn.intros*)  
**from** *cnvalidD* [*OF valid-modif [rule-format, of n init s] ctxt this*]  $P$   
**have**  $t' \in \text{Modif } (\text{init } s)$   
**by** *auto*  
**with** *ret-modif* **have**  $\text{Normal } (\text{return}' s t') =$   
 $\text{Normal } (\text{return } s t')$   
**by** *simp*  
**with** *exec-body* *exec-c*  $\text{bdy } n$   
**have**  $\Gamma \vdash \langle \text{call-exn init } p \text{ return}' \text{ result-exn } c, \text{Normal } s \rangle = n \Rightarrow t$   
**by** (*auto intro: execn-call-exn*)  
**from** *cnvalidD* [*OF valid-call [rule-format] ctxt this*]  $P \text{ t-notin-F}$   
**show** *?thesis*  
**by** *simp*

**next**

```

fix bdy m t'
assume bdy:  $\Gamma p = \text{Some } bdy$ 
assume exec-body:  $\Gamma \langle bdy, \text{Normal } (init\ s) \rangle = m \Rightarrow \text{Abrupt } t'$ 
assume n:  $n = \text{Suc } m$ 
assume t:  $t = \text{Abrupt } (\text{result-exn } (\text{return } s\ t')\ t')$ 
also
from exec-body n bdy
have  $\Gamma \langle \text{Call } p, \text{Normal } (init\ s) \rangle = n \Rightarrow \text{Abrupt } t'$ 
  by (auto simp add: intro: execn.intros)
from cinvalidD [OF valid-modif [rule-format, of n init s] ctxt this] P
have  $t' \in \text{ModifAbr } (init\ s)$ 
  by auto
with ret-modifAbr have  $\text{Abrupt } (\text{result-exn } (\text{return } s\ t')\ t') = \text{Abrupt } (\text{result-exn } (\text{return}'\ s\ t')\ t')$ 
  by simp
finally have  $t = \text{Abrupt } (\text{result-exn } (\text{return}'\ s\ t')\ t')$  .
with exec-body bdy n
have  $\Gamma \langle \text{call-exn } init\ p\ \text{return}'\ \text{result-exn } c, \text{Normal } s \rangle = n \Rightarrow t$ 
  by (auto intro: execn-call-exnAbrupt)
from cinvalidD [OF valid-call [rule-format] ctxt this] P t-notin-F
show ?thesis
  by simp
next
fix bdy m f
assume bdy:  $\Gamma p = \text{Some } bdy$ 
assume  $\Gamma \langle bdy, \text{Normal } (init\ s) \rangle = m \Rightarrow \text{Fault } f\ n = \text{Suc } m$  and
   $t = \text{Fault } f$ 
with bdy have  $\Gamma \langle \text{call-exn } init\ p\ \text{return}'\ \text{result-exn } c, \text{Normal } s \rangle = n \Rightarrow t$ 
  by (auto intro: execn-call-exnFault)
from cinvalidD [OF valid-call [rule-format] ctxt this] P t-notin-F
show ?thesis
  by simp
next
fix bdy m
assume bdy:  $\Gamma p = \text{Some } bdy$ 
assume  $\Gamma \langle bdy, \text{Normal } (init\ s) \rangle = m \Rightarrow \text{Stuck } n = \text{Suc } m$ 
   $t = \text{Stuck}$ 
with bdy have  $\Gamma \langle \text{call-exn } init\ p\ \text{return}'\ \text{result-exn } c, \text{Normal } s \rangle = n \Rightarrow t$ 
  by (auto intro: execn-call-exnStuck)
from valid-call [rule-format] ctxt this P t-notin-F
show ?thesis
  by (rule cinvalidD)
next
fix m
assume  $\Gamma p = \text{None}$ 
and  $n = \text{Suc } m\ t = \text{Stuck}$ 
then have  $\Gamma \langle \text{call-exn } init\ p\ \text{return}'\ \text{result-exn } c, \text{Normal } s \rangle = n \Rightarrow t$ 
  by (auto intro: execn-call-exnUndefined)
from valid-call [rule-format] ctxt this P t-notin-F

```

**show** *?thesis*  
**by** (*rule cinvalidD*)  
**qed**  
**qed**

**lemma** *ProcModifyReturnSameFaults-sound*:  
**assumes** *valid-call*:  $\forall n. \Gamma, \Theta \models_n: /_F P \text{ call init } p \text{ return}' c \ Q, A$   
**assumes** *valid-modif*:  
 $\forall \sigma. \forall n. \Gamma, \Theta \models_n: /_F \{ \sigma \} \text{ Call } p \text{ (Modif } \sigma), (\text{ModifAbr } \sigma)$   
**assumes** *ret-modif*:  
 $\forall s \ t. t \in \text{Modif } (\text{init } s)$   
 $\longrightarrow \text{return}' s \ t = \text{return } s \ t$   
**assumes** *ret-modifAbr*:  $\forall s \ t. t \in \text{ModifAbr } (\text{init } s)$   
 $\longrightarrow \text{return}' s \ t = \text{return } s \ t$   
**shows**  $\Gamma, \Theta \models_n: /_F P \text{ (call init } p \text{ return } c) \ Q, A$   
**using** *valid-call valid-modif ret-modif ret-modifAbr*  
**unfolding** *call-call-exn*  
**by** (*rule Proc-exnModifyReturnSameFaults-sound*)

**lemma** *Proc-exnModifyReturnSameFaults*:  
**assumes** *spec*:  $\Gamma, \Theta \vdash /_F P \text{ (call-exn init } p \text{ return}' \text{ result-exn } c) \ Q, A$   
**assumes** *result-conform*:  
 $\forall s \ t. t \in \text{Modif } (\text{init } s) \longrightarrow (\text{return}' s \ t) = (\text{return } s \ t)$   
**assumes** *return-conform*:  
 $\forall s \ t. t \in \text{ModifAbr } (\text{init } s) \longrightarrow (\text{result-exn } (\text{return}' s \ t) \ t) = (\text{result-exn } (\text{return } s \ t) \ t)$   
**assumes** *modifies-spec*:  
 $\forall \sigma. \Gamma, \Theta \vdash /_F \{ \sigma \} \text{ Call } p \text{ (Modif } \sigma), (\text{ModifAbr } \sigma)$   
**shows**  $\Gamma, \Theta \vdash /_F P \text{ (call-exn init } p \text{ return } \text{result-exn } c) \ Q, A$   
**apply** (*rule hoare-complete'*)  
**apply** (*rule allI*)  
**apply** (*rule Proc-exnModifyReturnSameFaults-sound*  
 $[\text{where } \text{Modif} = \text{Modif} \ \text{and} \ \text{ModifAbr} = \text{ModifAbr},$   
 $\text{OF } - - \text{result-conform } \text{return-conform}]$ )  
**using** *spec*  
**apply** (*blast intro: hoare-cinvalid*)  
**using** *modifies-spec*  
**apply** (*blast intro: hoare-cinvalid*)  
**done**

**lemma** *ProcModifyReturnSameFaults*:  
**assumes** *spec*:  $\Gamma, \Theta \vdash /_F P \text{ (call init } p \text{ return}' c) \ Q, A$   
**assumes** *result-conform*:  
 $\forall s \ t. t \in \text{Modif } (\text{init } s) \longrightarrow (\text{return}' s \ t) = (\text{return } s \ t)$   
**assumes** *return-conform*:  
 $\forall s \ t. t \in \text{ModifAbr } (\text{init } s) \longrightarrow (\text{return}' s \ t) = (\text{return } s \ t)$

**assumes** *modifies-spec*:  
 $\forall \sigma. \Gamma, \Theta \vdash_{/F} \{\sigma\} \text{ Call } p \text{ (Modif } \sigma), (\text{ModifAbr } \sigma)$   
**shows**  $\Gamma, \Theta \vdash_{/F} P \text{ (call init } p \text{ return } c) Q, A$   
**using** *spec result-conform return-conform modifies-spec*  
**unfolding** *call-call-exn*  
**by** (*rule Proc-exnModifyReturnSameFaults*)

### 5.3.3 DynCall

**lemma** *dynProc-exnModifyReturn-sound*:  
**assumes** *valid-call*:  $\bigwedge n. \Gamma, \Theta \models n:_{/F} P \text{ dynCall-exn } f g \text{ init } p \text{ return}' \text{ result-exn } c$   
 $Q, A$   
**assumes** *valid-modif*:  
 $\forall s \in P. \forall \sigma. \forall n.$   
 $\Gamma, \Theta \models n:_{/UNIV} \{\sigma\} \text{ Call } (p s) \text{ (Modif } \sigma), (\text{ModifAbr } \sigma)$   
**assumes** *ret-modif*:  
 $\forall s t. t \in \text{Modif } (\text{init } s)$   
 $\longrightarrow \text{return}' s t = \text{return } s t$   
**assumes** *ret-modifAbr*:  $\forall s t. t \in \text{ModifAbr } (\text{init } s)$   
 $\longrightarrow \text{result-exn } (\text{return}' s t) t = \text{result-exn } (\text{return } s t) t$   
**shows**  $\Gamma, \Theta \models n:_{/F} P \text{ (dynCall-exn } f g \text{ init } p \text{ return result-exn } c) Q, A$   
**proof** (*rule cinvalidI*)  
**fix**  $s t$   
**assume** *ctxt*:  $\forall (P, p, Q, A) \in \Theta. \Gamma \models n:_{/F} P \text{ (Call } p) Q, A$   
**then have** *ctxt'*:  $\forall (P, p, Q, A) \in \Theta. \Gamma \models n:_{/UNIV} P \text{ (Call } p) Q, A$   
**by** (*auto intro: nvalid-augment-Faults*)  
**assume** *exec*:  $\Gamma \vdash \langle \text{dynCall-exn } f g \text{ init } p \text{ return result-exn } c, \text{Normal } s \rangle = n \Rightarrow t$   
**assume** *t-notin-F*:  $t \notin \text{Fault } ' F$   
**assume**  $P: s \in P$   
**with** *valid-modif*  
**have** *valid-modif'*:  $\forall \sigma. \forall n.$   
 $\Gamma, \Theta \models n:_{/UNIV} \{\sigma\} \text{ Call } (p s) \text{ (Modif } \sigma), (\text{ModifAbr } \sigma)$   
**by** *blast*  
**from** *exec*  
**have** *exec-call*:  $\Gamma \vdash \langle \text{maybe-guard } f g \text{ (call-exn init } (p s) \text{ return result-exn } c), \text{Normal } s \rangle = n \Rightarrow t$   
**by** (*cases rule: execn-dynCall-exn-Normal-elim*)  
**then show**  $t \in \text{Normal } ' Q \cup \text{Abrupt } ' A$   
**proof** (*cases rule: execn-maybe-guard-Normal-elim-cases*)  
**case** *noFault*  
**from** *noFault* **have** *guards-ok*:  $s \in g$  **by** *simp*  
**from** *noFault* **have**  $\Gamma \vdash \langle \text{call-exn init } (p s) \text{ return result-exn } c, \text{Normal } s \rangle = n \Rightarrow$   
 $t$  **by** *simp*  
**then show**  $t \in \text{Normal } ' Q \cup \text{Abrupt } ' A$   
**proof** (*cases rule: execn-call-exn-Normal-elim*)  
**fix** *bdy m t'*  
**assume** *bdy*:  $\Gamma (p s) = \text{Some } \text{bdy}$   
**assume** *exec-body*:  $\Gamma \vdash \langle \text{bdy}, \text{Normal } (\text{init } s) \rangle = m \Rightarrow \text{Normal } t'$

```

assume exec-c:  $\Gamma \vdash \langle c \ s \ t', Normal \ (return \ s \ t') \rangle = Suc \ m \Rightarrow t$ 
assume n:  $n = Suc \ m$ 
from exec-body n bdy
have  $\Gamma \vdash \langle Call \ (p \ s) \ , Normal \ (init \ s) \rangle = n \Rightarrow Normal \ t'$ 
  by (auto simp add: intro: execn.intros)
from cvalidD [OF valid-modif' [rule-format, of n init s] ctxt' this] P
have  $t' \in Modif \ (init \ s)$ 
  by auto
with ret-modif have  $Normal \ (return' \ s \ t') = Normal \ (return \ s \ t')$ 
  by simp
with exec-body exec-c bdy n
have  $\Gamma \vdash \langle call\text{-exn} \ init \ (p \ s) \ return' \ result\text{-exn} \ c, Normal \ s \rangle = n \Rightarrow t$ 
  by (auto intro: execn-call-exn)
from execn-maybe-guard-noFault [OF this guards-ok]
have  $\Gamma \vdash \langle dynCall\text{-exn} \ f \ g \ init \ p \ return' \ result\text{-exn} \ c, Normal \ s \rangle = n \Rightarrow t$ 
  by (simp add: dynCall-exn-def execn-maybe-guard-DynCom)
from cvalidD [OF valid-call ctxt this] P t-notin-F
show ?thesis
  by simp
next
fix bdy m t'
assume bdy:  $\Gamma \ (p \ s) = Some \ bdy$ 
assume exec-body:  $\Gamma \vdash \langle bdy, Normal \ (init \ s) \rangle = m \Rightarrow Abrupt \ t'$ 
assume n:  $n = Suc \ m$ 
assume t:  $t = Abrupt \ (result\text{-exn} \ (return \ s \ t') \ t')$ 
also from exec-body n bdy
have  $\Gamma \vdash \langle Call \ (p \ s) \ , Normal \ (init \ s) \rangle = n \Rightarrow Abrupt \ t'$ 
  by (auto simp add: intro: execn.intros)
from cvalidD [OF valid-modif' [rule-format, of n init s] ctxt' this] P
have  $t' \in ModifAbr \ (init \ s)$ 
  by auto
with ret-modifAbr have  $Abrupt \ (result\text{-exn} \ (return \ s \ t') \ t') = Abrupt \ (result\text{-exn}$ 
(return' \ s \ t') \ t')
  by simp
finally have  $t = Abrupt \ (result\text{-exn} \ (return' \ s \ t') \ t') .$ 
with exec-body bdy n
have  $\Gamma \vdash \langle call\text{-exn} \ init \ (p \ s) \ return' \ result\text{-exn} \ c, Normal \ s \rangle = n \Rightarrow t$ 
  by (auto intro: execn-call-exnAbrupt)
from execn-maybe-guard-noFault [OF this guards-ok]
have  $\Gamma \vdash \langle dynCall\text{-exn} \ f \ g \ init \ p \ return' \ result\text{-exn} \ c, Normal \ s \rangle = n \Rightarrow t$ 
  by (simp add: dynCall-exn-def execn-maybe-guard-DynCom)
from cvalidD [OF valid-call ctxt this] P t-notin-F
show ?thesis
  by simp
next
fix bdy m f'
assume bdy:  $\Gamma \ (p \ s) = Some \ bdy$ 
assume  $\Gamma \vdash \langle bdy, Normal \ (init \ s) \rangle = m \Rightarrow Fault \ f' \ n = Suc \ m$ 
   $t = Fault \ f'$ 

```

```

with bdy have  $\Gamma \vdash \langle \text{call-exn } \text{init } (p \ s) \ \text{return}' \ \text{result-exn } c, \text{Normal } s \rangle =n \Rightarrow t$ 
  by (auto intro: execn-call-exnFault)
from execn-maybe-guard-noFault [OF this guards-ok]
have  $\Gamma \vdash \langle \text{dynCall-exn } f \ g \ \text{init } p \ \text{return}' \ \text{result-exn } c, \text{Normal } s \rangle =n \Rightarrow t$ 
  by (simp add: dynCall-exn-def execn-maybe-guard-DynCom)
from valid-call ctxt this P t-notin-F
show ?thesis
  by (rule cinvalidD)
next
fix bdy m
assume bdy:  $\Gamma \ (p \ s) = \text{Some } bdy$ 
assume  $\Gamma \vdash \langle bdy, \text{Normal } (\text{init } s) \rangle =m \Rightarrow \text{Stuck } n = \text{Suc } m$ 
  t = Stuck
with bdy have  $\Gamma \vdash \langle \text{call-exn } \text{init } (p \ s) \ \text{return}' \ \text{result-exn } c, \text{Normal } s \rangle =n \Rightarrow t$ 
  by (auto intro: execn-call-exnStuck)
from execn-maybe-guard-noFault [OF this guards-ok]
have  $\Gamma \vdash \langle \text{dynCall-exn } f \ g \ \text{init } p \ \text{return}' \ \text{result-exn } c, \text{Normal } s \rangle =n \Rightarrow t$ 
  by (simp add: dynCall-exn-def execn-maybe-guard-DynCom)
from valid-call ctxt this P t-notin-F
show ?thesis
  by (rule cinvalidD)
next
fix m
assume  $\Gamma \ (p \ s) = \text{None}$ 
and n = Suc m t = Stuck
hence  $\Gamma \vdash \langle \text{call-exn } \text{init } (p \ s) \ \text{return}' \ \text{result-exn } c, \text{Normal } s \rangle =n \Rightarrow t$ 
  by (auto intro: execn-call-exnUndefined)
from execn-maybe-guard-noFault [OF this guards-ok]
have  $\Gamma \vdash \langle \text{dynCall-exn } f \ g \ \text{init } p \ \text{return}' \ \text{result-exn } c, \text{Normal } s \rangle =n \Rightarrow t$ 
  by (simp add: dynCall-exn-def execn-maybe-guard-DynCom)
from valid-call ctxt this P t-notin-F
show ?thesis
  by (rule cinvalidD)
qed
next
case (someFault)
then obtain guards-fail:  $s \notin g$ 
  and t:  $t = \text{Fault } f$  by simp
from execn-maybe-guard-Fault [OF guards-fail] t
have  $\Gamma \vdash \langle \text{dynCall-exn } f \ g \ \text{init } p \ \text{return}' \ \text{result-exn } c, \text{Normal } s \rangle =n \Rightarrow t$ 
  by (simp add: dynCall-exn-def execn-maybe-guard-DynCom)
from cinvalidD [OF valid-call ctxt this] P t-notin-F
show ?thesis by simp
qed
qed

lemma dynProcModifyReturn-sound:
assumes valid-call:  $\bigwedge n. \Gamma, \Theta \models n \ /_F P \ \text{dynCall } \text{init } p \ \text{return}' \ c \ Q, A$ 
assumes valid-modif:

```

$\forall s \in P. \forall \sigma. \forall n.$   
 $\Gamma, \Theta \models_n: /_{UNIV} \{\sigma\} \text{ Call } (p \ s) \ (\text{Modif } \sigma), (\text{ModifAbr } \sigma)$   
**assumes** *ret-modif*:  
 $\forall s \ t. t \in \text{Modif } (\text{init } s)$   
 $\longrightarrow \text{return}' \ s \ t = \text{return } s \ t$   
**assumes** *ret-modifAbr*:  $\forall s \ t. t \in \text{ModifAbr } (\text{init } s)$   
 $\longrightarrow \text{return}' \ s \ t = \text{return } s \ t$   
**shows**  $\Gamma, \Theta \models_n: /_F P \ (\text{dynCall } \text{init } p \ \text{return } c) \ Q, A$   
**using** *valid-call valid-modif ret-modif ret-modifAbr*  
**unfolding** *dynCall-dynCall-exn*  
**by** (*rule dynProc-exnModifyReturn-sound*)

**lemma** *dynProc-exnModifyReturn*:

**assumes** *dyn-call*:  $\Gamma, \Theta \vdash /_F P \ \text{dynCall-exn } f \ g \ \text{init } p \ \text{return}' \ \text{result-exn } c \ Q, A$

**assumes** *ret-modif*:

$\forall s \ t. t \in \text{Modif } (\text{init } s)$   
 $\longrightarrow \text{return}' \ s \ t = \text{return } s \ t$

**assumes** *ret-modifAbr*:  $\forall s \ t. t \in \text{ModifAbr } (\text{init } s)$

$\longrightarrow \text{result-exn } (\text{return}' \ s \ t) \ t = \text{result-exn } (\text{return } s \ t) \ t$

**assumes** *modif*:

$\forall s \in P. \forall \sigma.$

$\Gamma, \Theta \vdash /_{UNIV} \{\sigma\} \text{ Call } (p \ s) \ (\text{Modif } \sigma), (\text{ModifAbr } \sigma)$

**shows**  $\Gamma, \Theta \vdash /_F P \ (\text{dynCall-exn } f \ g \ \text{init } p \ \text{return } \text{result-exn } c) \ Q, A$

**apply** (*rule hoare-complete'*)

**apply** (*rule allI*)

**apply** (*rule dynProc-exnModifyReturn-sound* [**where** *Modif=Modif* **and** *ModifAbr=ModifAbr*,

*OF hoare-cnvalid* [*OF dyn-call*] - *ret-modif ret-modifAbr*])

**apply** (*intro ballI allI*)

**apply** (*rule hoare-cnvalid* [*OF modif* [*rule-format*]])

**apply** *assumption*

**done**

**lemma** *dynProcModifyReturn*:

**assumes** *dyn-call*:  $\Gamma, \Theta \vdash /_F P \ \text{dynCall } \text{init } p \ \text{return}' \ c \ Q, A$

**assumes** *ret-modif*:

$\forall s \ t. t \in \text{Modif } (\text{init } s)$   
 $\longrightarrow \text{return}' \ s \ t = \text{return } s \ t$

**assumes** *ret-modifAbr*:  $\forall s \ t. t \in \text{ModifAbr } (\text{init } s)$

$\longrightarrow \text{return}' \ s \ t = \text{return } s \ t$

**assumes** *modif*:

$\forall s \in P. \forall \sigma.$

$\Gamma, \Theta \vdash /_{UNIV} \{\sigma\} \text{ Call } (p \ s) \ (\text{Modif } \sigma), (\text{ModifAbr } \sigma)$

**shows**  $\Gamma, \Theta \vdash /_F P \ (\text{dynCall } \text{init } p \ \text{return } c) \ Q, A$

**using** *dyn-call ret-modif ret-modifAbr modif*

**unfolding** *dynCall-dynCall-exn*

**by** (*rule dynProc-exnModifyReturn*)



**lemma** *dynProc-exnModifyReturnSameFaults-sound*:  
**assumes** *valid-call*:  $\bigwedge n. \Gamma, \Theta \models_{/F} P \text{ dynCall-exn } f \ g \ \text{init } p \ \text{return}' \ \text{result-exn } c \ Q, A$   
**assumes** *valid-modif*:  
 $\forall s \in P. \forall \sigma. \forall n. \Gamma, \Theta \models_{/F} \{\sigma\} \text{ Call } (p \ s) \ (\text{Modif } \sigma), (\text{ModifAbr } \sigma)$   
**assumes** *ret-modif*:  
 $\forall s \ t. t \in \text{Modif } (\text{init } s) \longrightarrow \text{return}' \ s \ t = \text{return } s \ t$   
**assumes** *ret-modifAbr*:  $\forall s \ t. t \in \text{ModifAbr } (\text{init } s) \longrightarrow \text{result-exn } (\text{return}' \ s \ t) \ t = \text{result-exn } (\text{return } s \ t) \ t$   
**shows**  $\Gamma, \Theta \models_{/F} P \ (\text{dynCall-exn } f \ g \ \text{init } p \ \text{return} \ \text{result-exn } c) \ Q, A$   
**proof** (*rule cinvalidI*)  
**fix**  $s \ t$   
**assume** *ctxt*:  $\forall (P, p, Q, A) \in \Theta. \Gamma \models_{/F} P \ (\text{Call } p) \ Q, A$   
**assume** *exec*:  $\Gamma \vdash \langle \text{dynCall-exn } f \ g \ \text{init } p \ \text{return} \ \text{result-exn } c, \text{Normal } s \rangle =_n \Rightarrow t$   
**assume** *t-notin-F*:  $t \notin \text{Fault } F$   
**assume**  $P: s \in P$   
**with** *valid-modif*  
**have** *valid-modif'*:  $\forall \sigma. \forall n. \Gamma, \Theta \models_{/F} \{\sigma\} \text{ Call } (p \ s) \ (\text{Modif } \sigma), (\text{ModifAbr } \sigma)$   
**by** *blast*  
**from** *exec*  
**have** *exec-call*:  $\Gamma \vdash \langle \text{maybe-guard } f \ g \ (\text{call-exn } \text{init } (p \ s) \ \text{return} \ \text{result-exn } c), \text{Normal } s \rangle =_n \Rightarrow t$   
**by** (*cases rule: execn-dynCall-exn-Normal-elim*)  
**then show**  $t \in \text{Normal } Q \cup \text{Abrupt } A$   
**proof** (*cases rule: execn-maybe-guard-Normal-elim-cases*)  
**case** *noFault*  
**from** *noFault* **have** *guards-ok*:  $s \in g$  **by** *simp*  
**from** *noFault* **have**  $\Gamma \vdash \langle \text{call-exn } \text{init } (p \ s) \ \text{return} \ \text{result-exn } c, \text{Normal } s \rangle =_n \Rightarrow t$   
**by** *simp*  
**then show**  $t \in \text{Normal } Q \cup \text{Abrupt } A$   
**proof** (*cases rule: execn-call-exn-Normal-elim*)  
**fix**  $\text{bdy } m \ t'$   
**assume** *bdy*:  $\Gamma \ (p \ s) = \text{Some } \text{bdy}$   
**assume** *exec-body*:  $\Gamma \vdash \langle \text{bdy}, \text{Normal } (\text{init } s) \rangle =_m \Rightarrow \text{Normal } t'$   
**assume** *exec-c*:  $\Gamma \vdash \langle c \ s \ t', \text{Normal } (\text{return } s \ t') \rangle =_{\text{Suc } m} \Rightarrow t$   
**assume**  $n: n = \text{Suc } m$   
**from** *exec-body*  $n \ \text{bdy}$   
**have**  $\Gamma \vdash \langle \text{Call } (p \ s) \ , \text{Normal } (\text{init } s) \rangle =_n \Rightarrow \text{Normal } t'$   
**by** (*auto simp add: intro: execn.Call*)  
**from** *cinvalidD* [*OF* *valid-modif'* [*rule-format*, *of n init s*] *ctxt this*]  $P$   
**have**  $t' \in \text{Modif } (\text{init } s)$   
**by** *auto*  
**with** *ret-modif* **have**  $\text{Normal } (\text{return}' \ s \ t') = \text{Normal } (\text{return } s \ t')$   
**by** *simp*  
**with** *exec-body* *exec-c*  $\text{bdy } n$

```

have  $\Gamma \vdash \langle \text{call-exn } \text{init } (p \ s) \ \text{return}' \ \text{result-exn } c, \text{Normal } s \rangle = n \Rightarrow t$ 
  by (auto intro: execn-call-exn)
from execn-maybe-guard-noFault [OF this guards-ok]
have  $\Gamma \vdash \langle \text{dynCall-exn } f \ g \ \text{init } p \ \text{return}' \ \text{result-exn } c, \text{Normal } s \rangle = n \Rightarrow t$ 
  by (simp add: dynCall-exn-def execn-maybe-guard-DynCom)
from cvalidD [OF valid-call ctxt this] P t-notin-F
show ?thesis
  by simp
next
fix bdy m t'
assume bdy:  $\Gamma (p \ s) = \text{Some } \text{bdy}$ 
assume exec-body:  $\Gamma \vdash \langle \text{bdy}, \text{Normal } (\text{init } s) \rangle = m \Rightarrow \text{Abrupt } t'$ 
assume n:  $n = \text{Suc } m$ 
assume t:  $t = \text{Abrupt } (\text{result-exn } (\text{return } s \ t') \ t')$ 
also from exec-body n bdy
have  $\Gamma \vdash \langle \text{Call } (p \ s), \text{Normal } (\text{init } s) \rangle = n \Rightarrow \text{Abrupt } t'$ 
  by (auto simp add: intro: execn.intros)
from cvalidD [OF valid-modif' [rule-format, of n init s] ctxt this] P
have  $t' \in \text{ModifAbr } (\text{init } s)$ 
  by auto
with ret-modifAbr have  $\text{Abrupt } (\text{result-exn } (\text{return } s \ t') \ t') = \text{Abrupt } (\text{result-exn } (\text{return}' \ s \ t') \ t')$ 
  by simp
finally have  $t = \text{Abrupt } (\text{result-exn } (\text{return}' \ s \ t') \ t')$  .
with exec-body bdy n
have  $\Gamma \vdash \langle \text{call-exn } \text{init } (p \ s) \ \text{return}' \ \text{result-exn } c, \text{Normal } s \rangle = n \Rightarrow t$ 
  by (auto intro: execn-call-exnAbrupt)
from execn-maybe-guard-noFault [OF this guards-ok]
have  $\Gamma \vdash \langle \text{dynCall-exn } f \ g \ \text{init } p \ \text{return}' \ \text{result-exn } c, \text{Normal } s \rangle = n \Rightarrow t$ 
  by (simp add: dynCall-exn-def execn-maybe-guard-DynCom)
from cvalidD [OF valid-call ctxt this] P t-notin-F
show ?thesis
  by simp
next
fix bdy m f'
assume bdy:  $\Gamma (p \ s) = \text{Some } \text{bdy}$ 
assume  $\Gamma \vdash \langle \text{bdy}, \text{Normal } (\text{init } s) \rangle = m \Rightarrow \text{Fault } f' \ n = \text{Suc } m$  and
   $t: t = \text{Fault } f'$ 
with bdy have  $\Gamma \vdash \langle \text{call-exn } \text{init } (p \ s) \ \text{return}' \ \text{result-exn } c, \text{Normal } s \rangle = n \Rightarrow t$ 
  by (auto intro: execn-call-exnFault)
from execn-maybe-guard-noFault [OF this guards-ok]
have  $\Gamma \vdash \langle \text{dynCall-exn } f \ g \ \text{init } p \ \text{return}' \ \text{result-exn } c, \text{Normal } s \rangle = n \Rightarrow t$ 
  by (simp add: dynCall-exn-def execn-maybe-guard-DynCom)
from cvalidD [OF valid-call ctxt this P] t t-notin-F
show ?thesis
  by simp
next
fix bdy m
assume bdy:  $\Gamma (p \ s) = \text{Some } \text{bdy}$ 

```

```

assume  $\Gamma \vdash \langle bdy, Normal (init\ s) \rangle = n \Rightarrow Stuck\ n = Suc\ m$ 
   $t = Stuck$ 
with  $bdy$  have  $\Gamma \vdash \langle call\text{-}exn\ init\ (p\ s)\ return'\ result\text{-}exn\ c, Normal\ s \rangle = n \Rightarrow t$ 
  by (auto intro: execn-call-exnStuck)
from execn-maybe-guard-noFault [OF this guards-ok]
have  $\Gamma \vdash \langle dynCall\text{-}exn\ f\ g\ init\ p\ return'\ result\text{-}exn\ c, Normal\ s \rangle = n \Rightarrow t$ 
  by (simp add: dynCall-exn-def execn-maybe-guard-DynCom)
from valid-call ctxt this P t-notin-F
show ?thesis
  by (rule cinvalidD)
next
fix  $m$ 
assume  $\Gamma (p\ s) = None$ 
and  $n = Suc\ m\ t = Stuck$ 
hence  $\Gamma \vdash \langle call\text{-}exn\ init\ (p\ s)\ return'\ result\text{-}exn\ c, Normal\ s \rangle = n \Rightarrow t$ 
  by (auto intro: execn-call-exnUndefined)
from execn-maybe-guard-noFault [OF this guards-ok]
have  $\Gamma \vdash \langle dynCall\text{-}exn\ f\ g\ init\ p\ return'\ result\text{-}exn\ c, Normal\ s \rangle = n \Rightarrow t$ 
  by (simp add: dynCall-exn-def execn-maybe-guard-DynCom)
from valid-call ctxt this P t-notin-F
show ?thesis
  by (rule cinvalidD)
qed
next
case (someFault)
then obtain guards-fail:s  $s \notin g$ 
  and  $t: t = Fault\ f$  by simp
from execn-maybe-guard-Fault [OF guards-fail]  $t$ 
have  $\Gamma \vdash \langle dynCall\text{-}exn\ f\ g\ init\ p\ return'\ result\text{-}exn\ c, Normal\ s \rangle = n \Rightarrow t$ 
  by (simp add: dynCall-exn-def execn-guards-DynCom)
from cnvalidD [OF valid-call ctxt this]  $P\ t\text{-notin-F}$ 
show ?thesis by simp
qed
qed

```

**lemma** *dynProcModifyReturnSameFaults-sound*:

**assumes** *valid-call*:  $\bigwedge n. \Gamma, \Theta \models n: /_F P\ dynCall\ init\ p\ return'\ c\ Q, A$

**assumes** *valid-modif*:

$\forall s \in P. \forall \sigma. \forall n.$

$\Gamma, \Theta \models n: /_F \{\sigma\}\ Call\ (p\ s)\ (Modif\ \sigma), (ModifAbr\ \sigma)$

**assumes** *ret-modif*:

$\forall s\ t. t \in Modif\ (init\ s) \longrightarrow return'\ s\ t = return\ s\ t$

**assumes** *ret-modifAbr*:  $\forall s\ t. t \in ModifAbr\ (init\ s) \longrightarrow return'\ s\ t = return\ s\ t$

**shows**  $\Gamma, \Theta \models n: /_F P\ (dynCall\ init\ p\ return\ c)\ Q, A$

**using** *valid-call valid-modif ret-modif ret-modifAbr*

**unfolding** *dynCall-dynCall-exn*

**by** (*rule dynProc-exnModifyReturnSameFaults-sound*)

**lemma** *dynProc-exnModifyReturnSameFaults*:  
**assumes** *dyn-call*:  $\Gamma, \Theta \vdash_F P \text{ dynCall-exn } f \ g \ \text{init } p \ \text{return}' \ \text{result-exn } c \ Q, A$   
**assumes** *ret-modif*:  
 $\forall s \ t. t \in \text{Modif} \ (\text{init } s)$   
 $\longrightarrow \text{return}' \ s \ t = \text{return } s \ t$   
**assumes** *ret-modifAbr*:  $\forall s \ t. t \in \text{ModifAbr} \ (\text{init } s)$   
 $\longrightarrow \text{result-exn} \ (\text{return}' \ s \ t) \ t = \text{result-exn} \ (\text{return } s \ t) \ t$   
**assumes** *modif*:  
 $\forall s \in P. \forall \sigma. \Gamma, \Theta \vdash_F \{\sigma\} \ \text{Call} \ (p \ s) \ (\text{Modif } \sigma), (\text{ModifAbr } \sigma)$   
**shows**  $\Gamma, \Theta \vdash_F P \ (\text{dynCall-exn } f \ g \ \text{init } p \ \text{return} \ \text{result-exn } c) \ Q, A$   
**apply** (*rule hoare-complete'*)  
**apply** (*rule allI*)  
**apply** (*rule dynProc-exnModifyReturnSameFaults-sound*  
[**where** *Modif*=*Modif* **and** *ModifAbr*=*ModifAbr*,  
*OF hoare-cnvalid* [*OF dyn-call*] - *ret-modif ret-modifAbr*])  
**apply** (*intro ballI allI*)  
**apply** (*rule hoare-cnvalid* [*OF modif*] [*rule-format*])  
**apply** *assumption*  
**done**

**lemma** *dynProcModifyReturnSameFaults*:  
**assumes** *dyn-call*:  $\Gamma, \Theta \vdash_F P \ \text{dynCall} \ \text{init } p \ \text{return}' \ c \ Q, A$   
**assumes** *ret-modif*:  
 $\forall s \ t. t \in \text{Modif} \ (\text{init } s)$   
 $\longrightarrow \text{return}' \ s \ t = \text{return } s \ t$   
**assumes** *ret-modifAbr*:  $\forall s \ t. t \in \text{ModifAbr} \ (\text{init } s)$   
 $\longrightarrow \text{return}' \ s \ t = \text{return } s \ t$   
**assumes** *modif*:  
 $\forall s \in P. \forall \sigma. \Gamma, \Theta \vdash_F \{\sigma\} \ \text{Call} \ (p \ s) \ (\text{Modif } \sigma), (\text{ModifAbr } \sigma)$   
**shows**  $\Gamma, \Theta \vdash_F P \ (\text{dynCall} \ \text{init } p \ \text{return} \ c) \ Q, A$   
**using** *dyn-call ret-modif ret-modifAbr modif*  
**unfolding** *dynCall-dynCall-exn*  
**by** (*rule dynProc-exnModifyReturnSameFaults*)

### 5.3.4 Conjunction of Postcondition

**lemma** *PostConjI-sound*:  
**assumes** *valid-Q*:  $\forall n. \Gamma, \Theta \models_n \vdash_F P \ c \ Q, A$   
**assumes** *valid-R*:  $\forall n. \Gamma, \Theta \models_n \vdash_F P \ c \ R, B$   
**shows**  $\Gamma, \Theta \models_n \vdash_F P \ c \ (Q \cap R), (A \cap B)$   
**proof** (*rule cnvalidI*)  
**fix** *s t*  
**assume** *ctxt*:  $\forall (P, p, Q, A) \in \Theta. \Gamma \models_n \vdash_F P \ (\text{Call } p) \ Q, A$   
**assume** *exec*:  $\Gamma \vdash \langle c, \text{Normal } s \rangle =_n \Rightarrow t$   
**assume** *P*:  $s \in P$   
**assume** *t-notin-F*:  $t \notin \text{Fault } F$   
**from** *valid-Q* [*rule-format*] *ctxt exec P t-notin-F* **have**  $t \in \text{Normal } Q \cup \text{Abrupt } A$

by (rule cvalidD)  
**moreover**  
**from** valid-R [rule-format] ctxt exec P t-notin-F **have**  $t \in \text{Normal} \text{ ' } R \cup \text{Abrupt}$   
 $\text{' } B$   
 by (rule cvalidD)  
**ultimately show**  $t \in \text{Normal} \text{ ' } (Q \cap R) \cup \text{Abrupt} \text{ ' } (A \cap B)$   
 by blast  
**qed**

**lemma** PostConjI:  
**assumes** deriv-Q:  $\Gamma, \Theta \vdash_{/F} P \text{ c } Q, A$   
**assumes** deriv-R:  $\Gamma, \Theta \vdash_{/F} P \text{ c } R, B$   
**shows**  $\Gamma, \Theta \vdash_{/F} P \text{ c } (Q \cap R), (A \cap B)$   
**apply** (rule hoare-complete')  
**apply** (rule allI)  
**apply** (rule PostConjI-sound)  
**using** deriv-Q  
**apply** (blast intro: hoare-cvalid)  
**using** deriv-R  
**apply** (blast intro: hoare-cvalid)  
**done**

**lemma** Merge-PostConj-sound:  
**assumes** validF:  $\forall n. \Gamma, \Theta \models n:_{/F} P \text{ c } Q, A$   
**assumes** validG:  $\forall n. \Gamma, \Theta \models n:_{/G} P' \text{ c } R, X$   
**assumes** F-G:  $F \subseteq G$   
**assumes** P-P':  $P \subseteq P'$   
**shows**  $\Gamma, \Theta \models n:_{/F} P \text{ c } (Q \cap R), (A \cap X)$   
**proof** (rule cvalidI)  
**fix** s t  
**assume** ctxt:  $\forall (P, p, Q, A) \in \Theta. \Gamma \models n:_{/F} P \text{ (Call } p) \text{ } Q, A$   
**with** F-G **have** ctxt':  $\forall (P, p, Q, A) \in \Theta. \Gamma \models n:_{/G} P \text{ (Call } p) \text{ } Q, A$   
 by (auto intro: nvalid-augment-Faults)  
**assume** exec:  $\Gamma \vdash \langle c, \text{Normal } s \rangle = n \Rightarrow t$   
**assume** P:  $s \in P$   
**with** P-P' **have** P':  $s \in P'$   
 by auto  
**assume** t-noFault:  $t \notin \text{Fault} \text{ ' } F$   
**show**  $t \in \text{Normal} \text{ ' } (Q \cap R) \cup \text{Abrupt} \text{ ' } (A \cap X)$   
**proof** –  
**from** cvalidD [OF validF [rule-format] ctxt exec P t-noFault]  
**have** \*:  $t \in \text{Normal} \text{ ' } Q \cup \text{Abrupt} \text{ ' } A$ .  
**then have**  $t \notin \text{Fault} \text{ ' } G$   
 by auto  
**from** cvalidD [OF validG [rule-format] ctxt' exec P' this]  
**have**  $t \in \text{Normal} \text{ ' } R \cup \text{Abrupt} \text{ ' } X$  .  
**with** \* **show** ?thesis by auto  
**qed**

qed

**lemma** *Merge-PostConj*:

**assumes** *validF*:  $\Gamma, \Theta \vdash_{/F} P \text{ c } Q, A$

**assumes** *validG*:  $\Gamma, \Theta \vdash_{/G} P' \text{ c } R, X$

**assumes** *F-G*:  $F \subseteq G$

**assumes** *P-P'*:  $P \subseteq P'$

**shows**  $\Gamma, \Theta \vdash_{/F} P \text{ c } (Q \cap R), (A \cap X)$

**apply** (*rule hoare-complete'*)

**apply** (*rule allI*)

**apply** (*rule Merge-PostConj-sound* [*OF - - F-G P-P'*])

**using** *validF* **apply** (*blast intro:hoare-cnvalid*)

**using** *validG* **apply** (*blast intro:hoare-cnvalid*)

**done**

### 5.3.5 Weaken Context

**lemma** *WeakenContext-sound*:

**assumes** *valid-c*:  $\forall n. \Gamma, \Theta' \models_n /F P \text{ c } Q, A$

**assumes** *valid-ctxt*:  $\forall (P, p, Q, A) \in \Theta'. \Gamma, \Theta \models_n /F P (\text{Call } p) Q, A$

**shows**  $\Gamma, \Theta \models_n /F P \text{ c } Q, A$

**proof** (*rule cnvalidI*)

**fix** *s t*

**assume** *ctxt*:  $\forall (P, p, Q, A) \in \Theta. \Gamma \models_n /F P (\text{Call } p) Q, A$

**with** *valid-ctxt*

**have** *ctxt'*:  $\forall (P, p, Q, A) \in \Theta'. \Gamma \models_n /F P (\text{Call } p) Q, A$

**by** (*simp add: cnvalid-def*)

**assume** *exec*:  $\Gamma \vdash \langle c, \text{Normal } s \rangle = n \Rightarrow t$

**assume** *P*:  $s \in P$

**assume** *t-notin-F*:  $t \notin \text{Fault } F$

**from** *valid-c* [*rule-format*] *ctxt' exec P t-notin-F*

**show**  $t \in \text{Normal } Q \cup \text{Abrupt } A$

**by** (*rule cnvalidD*)

qed

**lemma** *WeakenContext*:

**assumes** *deriv-c*:  $\Gamma, \Theta \vdash_{/F} P \text{ c } Q, A$

**assumes** *deriv-ctxt*:  $\forall (P, p, Q, A) \in \Theta'. \Gamma, \Theta \vdash_{/F} P (\text{Call } p) Q, A$

**shows**  $\Gamma, \Theta \vdash_{/F} P \text{ c } Q, A$

**apply** (*rule hoare-complete'*)

**apply** (*rule allI*)

**apply** (*rule WeakenContext-sound*)

**using** *deriv-c*

**apply** (*blast intro: hoare-cnvalid*)

**using** *deriv-ctxt*

**apply** (*blast intro: hoare-cnvalid*)

**done**

### 5.3.6 Guards and Guarantees

**lemma** *SplitGuards-sound*:

**assumes** *valid-c1*:  $\forall n. \Gamma, \Theta \models n: /_F P \ c_1 \ Q, A$

**assumes** *valid-c2*:  $\forall n. \Gamma, \Theta \models n: /_F P \ c_2 \ UNIV, UNIV$

**assumes** *c*:  $(c_1 \cap_g c_2) = \text{Some } c$

**shows**  $\Gamma, \Theta \models n: /_F P \ c \ Q, A$

**proof** (*rule cinvalidI*)

**fix** *s t*

**assume** *ctxt*:  $\forall (P, p, Q, A) \in \Theta. \Gamma \models n: /_F P \ (\text{Call } p) \ Q, A$

**assume** *exec*:  $\Gamma \vdash \langle c, \text{Normal } s \rangle = n \Rightarrow t$

**assume** *P*:  $s \in P$

**assume** *t-notin-F*:  $t \notin \text{Fault } ' F$

**show**  $t \in \text{Normal } ' Q \cup \text{Abrupt } ' A$

**proof** (*cases t*)

**case** *Normal*

**with** *inter-guards-execn-noFault* [*OF c exec*]

**have**  $\Gamma \vdash \langle c_1, \text{Normal } s \rangle = n \Rightarrow t$  **by** *simp*

**from** *valid-c1* [*rule-format*] *ctxt* *this P t-notin-F*

**show** *?thesis*

**by** (*rule cinvalidD*)

**next**

**case** *Abrupt*

**with** *inter-guards-execn-noFault* [*OF c exec*]

**have**  $\Gamma \vdash \langle c_1, \text{Normal } s \rangle = n \Rightarrow t$  **by** *simp*

**from** *valid-c1* [*rule-format*] *ctxt* *this P t-notin-F*

**show** *?thesis*

**by** (*rule cinvalidD*)

**next**

**case** (*Fault f*)

**with** *exec inter-guards-execn-Fault* [*OF c*]

**have**  $\Gamma \vdash \langle c_1, \text{Normal } s \rangle = n \Rightarrow \text{Fault } f \vee \Gamma \vdash \langle c_2, \text{Normal } s \rangle = n \Rightarrow \text{Fault } f$

**by** *auto*

**then show** *?thesis*

**proof** (*cases rule: disjE [consumes 1]*)

**assume**  $\Gamma \vdash \langle c_1, \text{Normal } s \rangle = n \Rightarrow \text{Fault } f$

**from** *Fault cinvalidD* [*OF valid-c1*] [*rule-format*] *ctxt* *this P*] *t-notin-F*

**show** *?thesis*

**by** *blast*

**next**

**assume**  $\Gamma \vdash \langle c_2, \text{Normal } s \rangle = n \Rightarrow \text{Fault } f$

**from** *Fault cinvalidD* [*OF valid-c2*] [*rule-format*] *ctxt* *this P*] *t-notin-F*

**show** *?thesis*

**by** *blast*

**qed**

**next**

**case** *Stuck*

**with** *inter-guards-execn-noFault* [*OF c exec*]

**have**  $\Gamma \vdash \langle c_1, \text{Normal } s \rangle = n \Rightarrow t$  **by** *simp*

```

from valid-c1 [rule-format] ctxt this P t-notin-F
show ?thesis
  by (rule cinvalidD)
qed
qed

```

```

lemma SplitGuards:
  assumes c: (c1  $\cap_g$  c2) = Some c
  assumes deriv-c1:  $\Gamma, \Theta \vdash_{/F} P c_1 Q, A$ 
  assumes deriv-c2:  $\Gamma, \Theta \vdash_{/F} P c_2 UNIV, UNIV$ 
  shows  $\Gamma, \Theta \vdash_{/F} P c Q, A$ 
apply (rule hoare-complete')
apply (rule allI)
apply (rule SplitGuards-sound [OF - - c])
using deriv-c1
apply (blast intro: hoare-cinvalid)
using deriv-c2
apply (blast intro: hoare-cinvalid)
done

```

```

lemma CombineStrip-sound:
  assumes valid:  $\forall n. \Gamma, \Theta \models n:_{/F} P c Q, A$ 
  assumes valid-strip:  $\forall n. \Gamma, \Theta \models n:_{/\{\}} P (\text{strip-guards } (-F) c) UNIV, UNIV$ 
  shows  $\Gamma, \Theta \models n:_{/\{\}} P c Q, A$ 
proof (rule cinvalidI)
  fix s t
  assume ctxt:  $\forall (P, p, Q, A) \in \Theta. \Gamma \models n:_{/\{\}} P (\text{Call } p) Q, A$ 
  hence ctxt':  $\forall (P, p, Q, A) \in \Theta. \Gamma \models n:_{/F} P (\text{Call } p) Q, A$ 
  by (auto intro: nvalid-augment-Faults)
  assume exec:  $\Gamma \vdash \langle c, \text{Normal } s \rangle = n \Rightarrow t$ 
  assume P:  $s \in P$ 
  assume t-noFault:  $t \notin \text{Fault } \{\}$ 
  show  $t \in \text{Normal } Q \cup \text{Abrupt } A$ 
  proof (cases t)
    case (Normal t')
      from cnvalidD [OF valid [rule-format] ctxt' exec P] Normal
      show ?thesis
      by auto
    next
      case (Abrupt t')
        from cnvalidD [OF valid [rule-format] ctxt' exec P] Abrupt
        show ?thesis
        by auto
    next
      case (Fault f)
        show ?thesis
  proof (cases f  $\in F$ )
    case True

```



```

hence  $f \notin -F$  by simp
with exec Fault
have  $\Gamma \vdash \langle \text{strip-guards } (-F) \ c, \text{Normal } s \rangle =n \Rightarrow \text{Fault } f$ 
  by (auto intro: execn-to-execn-strip-guards-Fault)
from cnvalidD [OF valid-strip [rule-format] ctxt this P] Fault
have False
  by auto
thus ?thesis ..
next
  case False
  with cnvalidD [OF valid [rule-format] ctxt' exec P] Fault
  show ?thesis
    by auto
  qed
next
  case Stuck
  from cnvalidD [OF valid [rule-format] ctxt' exec P] Stuck
  show ?thesis
    by auto
  qed
qed

```

```

lemma CombineStrip:
  assumes deriv:  $\Gamma, \Theta \vdash_{/F} P \ c \ Q, A$ 
  assumes deriv-strip:  $\Gamma, \Theta \vdash_{/\{\}} P \ (\text{strip-guards } (-F) \ c) \ \text{UNIV}, \text{UNIV}$ 
  shows  $\Gamma, \Theta \vdash_{/\{\}} P \ c \ Q, A$ 
apply (rule hoare-complete')
apply (rule allI)
apply (rule CombineStrip-sound)
apply (iprover intro: hoare-cnvalid [OF deriv])
apply (iprover intro: hoare-cnvalid [OF deriv-strip])
done

```

```

lemma GuardsFlip-sound:
  assumes valid:  $\forall n. \Gamma, \Theta \models n:_{/F} P \ c \ Q, A$ 
  assumes validFlip:  $\forall n. \Gamma, \Theta \models n:_{/-F} P \ c \ \text{UNIV}, \text{UNIV}$ 
  shows  $\Gamma, \Theta \models n:_{/\{\}} P \ c \ Q, A$ 
proof (rule cnvalidI)
  fix s t
  assume ctxt:  $\forall (P, p, Q, A) \in \Theta. \Gamma \models n:_{/\{\}} P \ (\text{Call } p) \ Q, A$ 
  hence ctxt':  $\forall (P, p, Q, A) \in \Theta. \Gamma \models n:_{/F} P \ (\text{Call } p) \ Q, A$ 
    by (auto intro: nvalid-augment-Faults)
  from ctxt have ctxtFlip:  $\forall (P, p, Q, A) \in \Theta. \Gamma \models n:_{/-F} P \ (\text{Call } p) \ Q, A$ 
    by (auto intro: nvalid-augment-Faults)
  assume exec:  $\Gamma \vdash \langle c, \text{Normal } s \rangle =n \Rightarrow t$ 
  assume P:  $s \in P$ 
  assume t-noFault:  $t \notin \text{Fault } \{\}$ 
  show  $t \in \text{Normal } \{ Q \cup \text{Abrupt } \{ A \}$ 

```

```

proof (cases t)
  case (Normal t')
    from cinvalidD [OF valid [rule-format] ctxt' exec P] Normal
    show ?thesis
    by auto
  next
    case (Abrupt t')
      from cinvalidD [OF valid [rule-format] ctxt' exec P] Abrupt
      show ?thesis
      by auto
  next
    case (Fault f)
      show ?thesis
      proof (cases f ∈ F)
        case True
          hence f ∉ -F by simp
          with cinvalidD [OF validFlip [rule-format] ctxtFlip exec P] Fault
          have False
          by auto
          thus ?thesis ..
        next
          case False
            with cinvalidD [OF valid [rule-format] ctxt' exec P] Fault
            show ?thesis
            by auto
      qed
  next
    case Stuck
      from cinvalidD [OF valid [rule-format] ctxt' exec P] Stuck
      show ?thesis
      by auto
  qed
qed

```

```

lemma GuardsFlip:
  assumes deriv:  $\Gamma, \Theta \vdash_{/F} P \text{ c } Q, A$ 
  assumes derivFlip:  $\Gamma, \Theta \vdash_{/-F} P \text{ c } UNIV, UNIV$ 
  shows  $\Gamma, \Theta \vdash_{/\{\}} P \text{ c } Q, A$ 
apply (rule hoare-complete')
apply (rule allI)
apply (rule GuardsFlip-sound)
apply (iprover intro: hoare-cinvalid [OF deriv])
apply (iprover intro: hoare-cinvalid [OF derivFlip])
done

```

```

lemma MarkGuardsI-sound:
  assumes valid:  $\forall n. \Gamma, \Theta \models n: / \{\} P \text{ c } Q, A$ 
  shows  $\Gamma, \Theta \models n: / \{\} P \text{ mark-guards } f \text{ c } Q, A$ 

```

**proof** (*rule cinvalidI*)  
**fix**  $s\ t$   
**assume**  $ctxt: \forall (P, p, Q, A) \in \Theta. \Gamma \models n: / \{ \} P (Call\ p)\ Q, A$   
**assume**  $exec: \Gamma \vdash \langle mark\text{-}guards\ f\ c, Normal\ s \rangle = n \Rightarrow t$   
**from**  $execn\text{-}mark\text{-}guards\text{-}to\text{-}execn$  [*OF exec*] **obtain**  $t'$  **where**  
 $exec\text{-}c: \Gamma \vdash \langle c, Normal\ s \rangle = n \Rightarrow t'$  **and**  
 $t'\text{-}noFault: \neg isFault\ t' \longrightarrow t' = t$   
**by** *blast*  
**assume**  $P: s \in P$   
**assume**  $t\text{-}noFault: t \notin Fault\ ' \{ \}$   
**show**  $t \in Normal\ ' Q \cup Abrupt\ ' A$   
**proof** –  
**from**  $cnvalidD$  [*OF valid [rule-format] ctxt exec-c P*]  
**have**  $t' \in Normal\ ' Q \cup Abrupt\ ' A$   
**by** *blast*  
**with**  $t'\text{-}noFault$   
**show** *?thesis*  
**by** *auto*  
**qed**  
**qed**

**lemma** *MarkGuardsI*:  
**assumes**  $deriv: \Gamma, \Theta \vdash / \{ \} P\ c\ Q, A$   
**shows**  $\Gamma, \Theta \vdash / \{ \} P\ mark\text{-}guards\ f\ c\ Q, A$   
**apply** (*rule hoare-complete'*)  
**apply** (*rule allI*)  
**apply** (*rule MarkGuardsI-sound*)  
**apply** (*iprover intro: hoare-cinvalid [OF deriv]*)  
**done**

**lemma** *MarkGuardsD-sound*:  
**assumes**  $valid: \forall n. \Gamma, \Theta \models n: / \{ \} P\ mark\text{-}guards\ f\ c\ Q, A$   
**shows**  $\Gamma, \Theta \models n: / \{ \} P\ c\ Q, A$   
**proof** (*rule cinvalidI*)  
**fix**  $s\ t$   
**assume**  $ctxt: \forall (P, p, Q, A) \in \Theta. \Gamma \models n: / \{ \} P (Call\ p)\ Q, A$   
**assume**  $exec: \Gamma \vdash \langle c, Normal\ s \rangle = n \Rightarrow t$   
**assume**  $P: s \in P$   
**assume**  $t\text{-}noFault: t \notin Fault\ ' \{ \}$   
**show**  $t \in Normal\ ' Q \cup Abrupt\ ' A$   
**proof** (*cases isFault t*)  
**case** *True*  
**with**  $execn\text{-}to\text{-}execn\text{-}mark\text{-}guards\text{-}Fault$  [*OF exec* ]  
**obtain**  $f'$  **where**  $\Gamma \vdash \langle mark\text{-}guards\ f\ c, Normal\ s \rangle = n \Rightarrow Fault\ f'$   
**by** (*fastforce elim: isFaultE*)  
**from**  $cnvalidD$  [*OF valid [rule-format] ctxt this P*]  
**have** *False*  
**by** *auto*

```

    thus ?thesis ..
  next
    case False
    from execn-to-execn-mark-guards [OF exec False]
    obtain f' where  $\Gamma \vdash \langle \text{mark-guards } f \ c, \text{Normal } s \rangle = n \Rightarrow t$ 
      by auto
    from cvalidD [OF valid [rule-format] ctxt this P]
    show ?thesis
      by auto
  qed
qed

```

```

lemma MarkGuardsD:
  assumes deriv:  $\Gamma, \Theta \vdash / \{ \} P \text{ mark-guards } f \ c \ Q, A$ 
  shows  $\Gamma, \Theta \vdash / \{ \} P \ c \ Q, A$ 
  apply (rule hoare-complete')
  apply (rule allI)
  apply (rule MarkGuardsD-sound)
  apply (iprover intro: hoare-cvalid [OF deriv])
done

```

```

lemma MergeGuardsI-sound:
  assumes valid:  $\forall n. \Gamma, \Theta \models n: /_F P \ c \ Q, A$ 
  shows  $\Gamma, \Theta \models n: /_F P \ \text{merge-guards } c \ Q, A$ 
  proof (rule cvalidI)
    fix s t
    assume ctxt:  $\forall (P, p, Q, A) \in \Theta. \Gamma \models n: /_F P \ (\text{Call } p) \ Q, A$ 
    assume exec-merge:  $\Gamma \vdash \langle \text{merge-guards } c, \text{Normal } s \rangle = n \Rightarrow t$ 
    from execn-merge-guards-to-execn [OF exec-merge]
    have exec:  $\Gamma \vdash \langle c, \text{Normal } s \rangle = n \Rightarrow t$  .
    assume P:  $s \in P$ 
    assume t-notin-F:  $t \notin \text{Fault } 'F$ 
    from cvalidD [OF valid [rule-format] ctxt exec P t-notin-F]
    show  $t \in \text{Normal } 'Q \cup \text{Abrupt } 'A$ .
  qed

```

```

lemma MergeGuardsI:
  assumes deriv:  $\Gamma, \Theta \vdash /_F P \ c \ Q, A$ 
  shows  $\Gamma, \Theta \vdash /_F P \ \text{merge-guards } c \ Q, A$ 
  apply (rule hoare-complete')
  apply (rule allI)
  apply (rule MergeGuardsI-sound)
  apply (iprover intro: hoare-cvalid [OF deriv])
done

```

```

lemma MergeGuardsD-sound:
  assumes valid:  $\forall n. \Gamma, \Theta \models n: /_F P \ \text{merge-guards } c \ Q, A$ 
  shows  $\Gamma, \Theta \models n: /_F P \ c \ Q, A$ 

```

**proof** (*rule cvalidI*)  
**fix**  $s\ t$   
**assume**  $ctxt: \forall (P, p, Q, A) \in \Theta. \Gamma \models_n /_F P (Call\ p)\ Q, A$   
**assume**  $exec: \Gamma \vdash \langle c, Normal\ s \rangle = n \Rightarrow t$   
**from** *execn-to-execn-merge-guards* [*OF exec*]  
**have**  $exec\ merge: \Gamma \vdash \langle merge\ guards\ c, Normal\ s \rangle = n \Rightarrow t.$   
**assume**  $P: s \in P$   
**assume**  $t\ notin\ F: t \notin Fault\ 'F$   
**from** *cvalidD* [*OF valid* [*rule-format*] *ctxt exec-merge P t-notin-F*]  
**show**  $t \in Normal\ 'Q \cup Abrupt\ 'A.$   
**qed**

**lemma** *MergeGuardsD*:  
**assumes**  $deriv: \Gamma, \Theta \vdash /_F P\ merge\ guards\ c\ Q, A$   
**shows**  $\Gamma, \Theta \vdash /_F P\ c\ Q, A$   
**apply** (*rule hoare-complete'*)  
**apply** (*rule allI*)  
**apply** (*rule MergeGuardsD-sound*)  
**apply** (*iprover intro: hoare-cvalid* [*OF deriv*])  
**done**

**lemma** *SubsetGuards-sound*:  
**assumes**  $c\text{-}c': c \subseteq_g c'$   
**assumes**  $valid: \forall n. \Gamma, \Theta \models_n /_{\{\}} P\ c'\ Q, A$   
**shows**  $\Gamma, \Theta \models_n /_{\{\}} P\ c\ Q, A$   
**proof** (*rule cvalidI*)  
**fix**  $s\ t$   
**assume**  $ctxt: \forall (P, p, Q, A) \in \Theta. \Gamma \models_n /_{\{\}} P (Call\ p)\ Q, A$   
**assume**  $exec: \Gamma \vdash \langle c, Normal\ s \rangle = n \Rightarrow t$   
**from** *execn-to-execn-subseteq-guards* [*OF c-c' exec*] **obtain**  $t'$  **where**  
 $exec\ c': \Gamma \vdash \langle c', Normal\ s \rangle = n \Rightarrow t'$  **and**  
 $t'\ noFault: \neg isFault\ t' \longrightarrow t' = t$   
**by** *blast*  
**assume**  $P: s \in P$   
**assume**  $t\ noFault: t \notin Fault\ '\{\}$   
**from** *cvalidD* [*OF valid* [*rule-format*] *ctxt exec-c' P*]  $t'\ noFault\ t\ noFault$   
**show**  $t \in Normal\ 'Q \cup Abrupt\ 'A$   
**by** *auto*  
**qed**

**lemma** *SubsetGuards*:  
**assumes**  $c\text{-}c': c \subseteq_g c'$   
**assumes**  $deriv: \Gamma, \Theta \vdash /_{\{\}} P\ c'\ Q, A$   
**shows**  $\Gamma, \Theta \vdash /_{\{\}} P\ c\ Q, A$   
**apply** (*rule hoare-complete'*)  
**apply** (*rule allI*)  
**apply** (*rule SubsetGuards-sound* [*OF c-c'*])

**apply** (*iprover intro: hoare-cnvalid [OF deriv]*)  
**done**

**lemma** *NormalizeD-sound*:  
**assumes** *valid*:  $\forall n. \Gamma, \Theta \models n: /_F P \text{ (normalize } c) Q, A$   
**shows**  $\Gamma, \Theta \models n: /_F P c Q, A$   
**proof** (*rule cnvalidI*)  
**fix** *s t*  
**assume** *ctxt*:  $\forall (P, p, Q, A) \in \Theta. \Gamma \models n: /_F P \text{ (Call } p) Q, A$   
**assume** *exec*:  $\Gamma \vdash \langle c, \text{Normal } s \rangle = n \Rightarrow t$   
**hence** *exec-norm*:  $\Gamma \vdash \langle \text{normalize } c, \text{Normal } s \rangle = n \Rightarrow t$   
**by** (*rule execn-to-execn-normalize*)  
**assume** *P*:  $s \in P$   
**assume** *noFault*:  $t \notin \text{Fault } ' F$   
**from** *cnvalidD [OF valid [rule-format] ctxt exec-norm P noFault]*  
**show**  $t \in \text{Normal } ' Q \cup \text{Abrupt } ' A.$   
**qed**

**lemma** *NormalizeD*:  
**assumes** *deriv*:  $\Gamma, \Theta \vdash /_F P \text{ (normalize } c) Q, A$   
**shows**  $\Gamma, \Theta \vdash /_F P c Q, A$   
**apply** (*rule hoare-complete'*)  
**apply** (*rule allI*)  
**apply** (*rule NormalizeD-sound*)  
**apply** (*iprover intro: hoare-cnvalid [OF deriv]*)  
**done**

**lemma** *NormalizeI-sound*:  
**assumes** *valid*:  $\forall n. \Gamma, \Theta \models n: /_F P c Q, A$   
**shows**  $\Gamma, \Theta \models n: /_F P \text{ (normalize } c) Q, A$   
**proof** (*rule cnvalidI*)  
**fix** *s t*  
**assume** *ctxt*:  $\forall (P, p, Q, A) \in \Theta. \Gamma \models n: /_F P \text{ (Call } p) Q, A$   
**assume**  $\Gamma \vdash \langle \text{normalize } c, \text{Normal } s \rangle = n \Rightarrow t$   
**hence** *exec*:  $\Gamma \vdash \langle c, \text{Normal } s \rangle = n \Rightarrow t$   
**by** (*rule execn-normalize-to-execn*)  
**assume** *P*:  $s \in P$   
**assume** *noFault*:  $t \notin \text{Fault } ' F$   
**from** *cnvalidD [OF valid [rule-format] ctxt exec P noFault]*  
**show**  $t \in \text{Normal } ' Q \cup \text{Abrupt } ' A.$   
**qed**

**lemma** *NormalizeI*:  
**assumes** *deriv*:  $\Gamma, \Theta \vdash /_F P c Q, A$   
**shows**  $\Gamma, \Theta \vdash /_F P \text{ (normalize } c) Q, A$   
**apply** (*rule hoare-complete'*)  
**apply** (*rule allI*)  
**apply** (*rule NormalizeI-sound*)

**apply** (*iprover intro: hoare-cnvalid [OF deriv]*)  
**done**

### 5.3.7 Restricting the Procedure Environment

**lemma** *nvalid-restrict-to-nvalid*:

**assumes** *valid-c*:  $\Gamma \Vdash_M n: /_F P \ c \ Q, A$

**shows**  $\Gamma \Vdash n: /_F P \ c \ Q, A$

**proof** (*rule nvalidI*)

**fix**  $s \ t$

**assume** *exec*:  $\Gamma \vdash \langle c, Normal \ s \rangle = n \Rightarrow t$

**assume** *P*:  $s \in P$

**assume** *t-notin-F*:  $t \notin Fault \ ' \ F$

**show**  $t \in Normal \ ' \ Q \cup \ Abrupt \ ' \ A$

**proof** –

**from** *execn-to-execn-restrict* [*OF exec*]

**obtain**  $t'$  **where**

*exec-res*:  $\Gamma \Vdash_M \langle c, Normal \ s \rangle = n \Rightarrow t'$  **and**

*t-Fault*:  $\forall f. t = Fault \ f \longrightarrow t' \in \{Fault \ f, Stuck\}$  **and**

*t'-notStuck*:  $t' \neq Stuck \longrightarrow t' = t$

**by** *blast*

**from** *t-Fault t-notin-F t'-notStuck* **have**  $t' \notin Fault \ ' \ F$

**by** (*cases t'*) *auto*

**with** *valid-c exec-res P*

**have**  $t' \in Normal \ ' \ Q \cup \ Abrupt \ ' \ A$

**by** (*auto simp add: nvalid-def*)

**with** *t'-notStuck*

**show** *?thesis*

**by** *auto*

**qed**

**qed**

**lemma** *valid-restrict-to-valid*:

**assumes** *valid-c*:  $\Gamma \Vdash_M /_F P \ c \ Q, A$

**shows**  $\Gamma \Vdash /_F P \ c \ Q, A$

**proof** (*rule validI*)

**fix**  $s \ t$

**assume** *exec*:  $\Gamma \vdash \langle c, Normal \ s \rangle \Rightarrow t$

**assume** *P*:  $s \in P$

**assume** *t-notin-F*:  $t \notin Fault \ ' \ F$

**show**  $t \in Normal \ ' \ Q \cup \ Abrupt \ ' \ A$

**proof** –

**from** *exec-to-exec-restrict* [*OF exec*]

**obtain**  $t'$  **where**

*exec-res*:  $\Gamma \Vdash_M \langle c, Normal \ s \rangle \Rightarrow t'$  **and**

*t-Fault*:  $\forall f. t = Fault \ f \longrightarrow t' \in \{Fault \ f, Stuck\}$  **and**

*t'-notStuck*:  $t' \neq Stuck \longrightarrow t' = t$

**by** *blast*

**from** *t-Fault t-notin-F t'-notStuck* **have**  $t' \notin Fault \ ' \ F$

```

    by (cases t') auto
  with valid-c exec-res P
  have t' ∈ Normal ' Q ∪ Abrupt ' A
    by (auto simp add: valid-def)
  with t'-notStuck
  show ?thesis
    by auto
qed

```

```

lemma augment-procs:
  assumes deriv-c:  $\Gamma \mid M, \{\} \vdash /_F P \text{ c } Q, A$ 
  shows  $\Gamma, \{\} \vdash /_F P \text{ c } Q, A$ 
    apply (rule hoare-complete)
    apply (rule valid-restrict-to-valid)
    apply (insert hoare-sound [OF deriv-c])
    by (simp add: cvalid-def)

```

```

lemma augment-Faults:
  assumes deriv-c:  $\Gamma, \{\} \vdash /_F P \text{ c } Q, A$ 
  assumes F:  $F \subseteq F'$ 
  shows  $\Gamma, \{\} \vdash /_{F'} P \text{ c } Q, A$ 
    apply (rule hoare-complete)
    apply (rule valid-augment-Faults [OF - F])
    apply (insert hoare-sound [OF deriv-c])
    by (simp add: cvalid-def)

```

end

## 6 Derived Hoare Rules for Partial Correctness

```

theory HoarePartial imports HoarePartialProps begin

```

```

lemma conseq-no-aux:
   $\llbracket \Gamma, \Theta \vdash /_F P' \text{ c } Q', A';$ 
   $\forall s. s \in P \longrightarrow (s \in P' \wedge (Q' \subseteq Q) \wedge (A' \subseteq A)) \rrbracket$ 
 $\implies$ 
   $\Gamma, \Theta \vdash /_F P \text{ c } Q, A$ 
  by (rule conseq [where P'= $\lambda Z. P'$  and Q'= $\lambda Z. Q'$  and A'= $\lambda Z. A'$ ]) auto

```

```

lemma conseq-exploit-pre:
   $\llbracket \forall s \in P. \Gamma, \Theta \vdash /_F (\{s\} \cap P) \text{ c } Q, A \rrbracket$ 
 $\implies$ 
   $\Gamma, \Theta \vdash /_F P \text{ c } Q, A$ 
  apply (rule Conseq)
  apply clarify

```



**apply** (*rule-tac*  $x=\{s\} \cap P$  **in**  $exI$ )  
**apply** (*rule-tac*  $x=Q$  **in**  $exI$ )  
**apply** (*rule-tac*  $x=A$  **in**  $exI$ )  
**by** *simp*

**lemma** *conseq*:  $\llbracket \forall Z. \Gamma, \Theta \vdash_{/F} (P' Z) c (Q' Z), (A' Z);$   
 $\forall s. s \in P \longrightarrow (\exists Z. s \in P' Z \wedge (Q' Z \subseteq Q) \wedge (A' Z \subseteq A)) \rrbracket$   
 $\implies$   
 $\Gamma, \Theta \vdash_{/F} P c Q, A$   
**by** (*rule Conseq'*) *blast*

**lemma** *Lem*:  $\llbracket \forall Z. \Gamma, \Theta \vdash_{/F} (P' Z) c (Q' Z), (A' Z);$   
 $P \subseteq \{s. \exists Z. s \in P' Z \wedge (Q' Z \subseteq Q) \wedge (A' Z \subseteq A)\} \rrbracket$   
 $\implies$   
 $\Gamma, \Theta \vdash_{/F} P (lem\ x\ c)\ Q, A$   
**apply** (*unfold lem-def*)  
**apply** (*erule conseq*)  
**apply** *blast*  
**done**

**lemma** *LemAnno*:  
**assumes** *conseq*:  $P \subseteq \{s. \exists Z. s \in P' Z \wedge$   
 $(\forall t. t \in Q' Z \longrightarrow t \in Q) \wedge (\forall t. t \in A' Z \longrightarrow t \in A)\}$   
**assumes** *lem*:  $\forall Z. \Gamma, \Theta \vdash_{/F} (P' Z) c (Q' Z), (A' Z)$   
**shows**  $\Gamma, \Theta \vdash_{/F} P (lem\ x\ c)\ Q, A$   
**apply** (*rule Lem [OF lem]*)  
**using** *conseq*  
**by** *blast*

**lemma** *LemAnnoNoAbrupt*:  
**assumes** *conseq*:  $P \subseteq \{s. \exists Z. s \in P' Z \wedge (\forall t. t \in Q' Z \longrightarrow t \in Q)\}$   
**assumes** *lem*:  $\forall Z. \Gamma, \Theta \vdash_{/F} (P' Z) c (Q' Z), \{\}$   
**shows**  $\Gamma, \Theta \vdash_{/F} P (lem\ x\ c)\ Q, \{\}$   
**apply** (*rule Lem [OF lem]*)  
**using** *conseq*  
**by** *blast*

**lemma** *TrivPost*:  $\forall Z. \Gamma, \Theta \vdash_{/F} (P' Z) c (Q' Z), (A' Z)$   
 $\implies$   
 $\forall Z. \Gamma, \Theta \vdash_{/F} (P' Z) c UNIV, UNIV$   
**apply** (*rule allI*)  
**apply** (*erule conseq*)  
**apply** *auto*  
**done**

**lemma** *TrivPostNoAbr*:  $\forall Z. \Gamma, \Theta \vdash_{/F} (P' Z) c (Q' Z), \{\}$   
 $\implies$

$\forall Z. \Gamma, \Theta \vdash /_F (P' Z) c \text{ UNIV}, \{\}$

**apply** (rule allI)  
**apply** (erule conseq)  
**apply** auto  
**done**

**lemma** conseq-under-new-pre:  $\llbracket \Gamma, \Theta \vdash /_F P' c Q', A';$   
 $\forall s \in P. s \in P' \wedge Q' \subseteq Q \wedge A' \subseteq A \rrbracket$   
 $\implies \Gamma, \Theta \vdash /_F P c Q, A$   
**apply** (rule conseq)  
**apply** (rule allI)  
**apply** assumption  
**apply** auto  
**done**

**lemma** conseq-Kleymann:  $\llbracket \forall Z. \Gamma, \Theta \vdash /_F (P' Z) c (Q' Z), (A' Z);$   
 $\forall s \in P. (\exists Z. s \in P' Z \wedge (Q' Z \subseteq Q) \wedge (A' Z \subseteq A)) \rrbracket$   
 $\implies$   
 $\Gamma, \Theta \vdash /_F P c Q, A$   
**by** (rule Conseq') blast

**lemma** DynComConseq:  
**assumes**  $P \subseteq \{s. \exists P' Q' A'. \Gamma, \Theta \vdash /_F P' (c s) Q', A' \wedge P \subseteq P' \wedge Q' \subseteq Q \wedge A' \subseteq A\}$   
**shows**  $\Gamma, \Theta \vdash /_F P \text{ DynCom } c Q, A$   
**using** assms  
**apply** –  
**apply** (rule DynCom)  
**apply** clarsimp  
**apply** (rule Conseq)  
**apply** clarsimp  
**apply** blast  
**done**

**lemma** SpecAnno:  
**assumes** consequence:  $P \subseteq \{s. (\exists Z. s \in P' Z \wedge (Q' Z \subseteq Q) \wedge (A' Z \subseteq A))\}$   
**assumes** spec:  $\forall Z. \Gamma, \Theta \vdash /_F (P' Z) (c Z) (Q' Z), (A' Z)$   
**assumes** bdy-constant:  $\forall Z. c Z = c \text{ undefined}$   
**shows**  $\Gamma, \Theta \vdash /_F P (\text{specAnno } P' c Q' A') Q, A$   
**proof** –  
**from** spec bdy-constant  
**have**  $\forall Z. \Gamma, \Theta \vdash /_F ((P' Z)) (c \text{ undefined}) (Q' Z), (A' Z)$   
**apply** –  
**apply** (rule allI)  
**apply** (erule-tac  $x=Z$  in allE)  
**apply** (erule-tac  $x=Z$  in allE)  
**apply** simp  
**done**

**with** *consequence show* *?thesis*  
**apply** (*simp add: specAnno-def*)  
**apply** (*erule conseq*)  
**apply** *blast*  
**done**  
**qed**

**lemma** *SpecAnno'*:

$\llbracket P \subseteq \{s. \exists Z. s \in P' Z \wedge$   
 $(\forall t. t \in Q' Z \longrightarrow t \in Q) \wedge (\forall t. t \in A' Z \longrightarrow t \in A)\};$   
 $\forall Z. \Gamma, \Theta \vdash_{/F} (P' Z) (c Z) (Q' Z), (A' Z);$   
 $\forall Z. c Z = c \text{ undefined}$   
 $\rrbracket \Longrightarrow$   
 $\Gamma, \Theta \vdash_{/F} P (\text{specAnno } P' c Q' A') Q, A$   
**apply** (*simp only: subset-iff [THEN sym]*)  
**apply** (*erule (1) SpecAnno*)  
**apply** *assumption*  
**done**

**lemma** *SpecAnnoNoAbrupt*:

$\llbracket P \subseteq \{s. \exists Z. s \in P' Z \wedge$   
 $(\forall t. t \in Q' Z \longrightarrow t \in Q)\};$   
 $\forall Z. \Gamma, \Theta \vdash_{/F} (P' Z) (c Z) (Q' Z), \{\};$   
 $\forall Z. c Z = c \text{ undefined}$   
 $\rrbracket \Longrightarrow$   
 $\Gamma, \Theta \vdash_{/F} P (\text{specAnno } P' c Q' (\lambda s. \{\})) Q, A$   
**apply** (*rule SpecAnno'*)  
**apply** *auto*  
**done**

**lemma** *Skip*:  $P \subseteq Q \Longrightarrow \Gamma, \Theta \vdash_{/F} P \text{ Skip } Q, A$   
**by** (*rule hoarep.Skip [THEN conseqPre], simp*)

**lemma** *Basic*:  $P \subseteq \{s. (f s) \in Q\} \Longrightarrow \Gamma, \Theta \vdash_{/F} P (\text{Basic } f) Q, A$   
**by** (*rule hoarep.Basic [THEN conseqPre]*)

**lemma** *BasicCond*:

$\llbracket P \subseteq \{s. (b s \longrightarrow f s \in Q) \wedge (\neg b s \longrightarrow g s \in Q)\} \rrbracket \Longrightarrow$   
 $\Gamma, \Theta \vdash_{/F} P \text{ Basic } (\lambda s. \text{if } b s \text{ then } f s \text{ else } g s) Q, A$   
**apply** (*rule Basic*)  
**apply** *auto*  
**done**

**lemma** *Spec*:  $P \subseteq \{s. (\forall t. (s, t) \in r \longrightarrow t \in Q) \wedge (\exists t. (s, t) \in r)\}$   
 $\Longrightarrow \Gamma, \Theta \vdash_{/F} P (\text{Spec } r) Q, A$   
**by** (*rule hoarep.Spec [THEN conseqPre]*)

**lemma SpecIf:**  
 $\llbracket P \subseteq \{s. (b\ s \longrightarrow f\ s \in Q) \wedge (\neg\ b\ s \longrightarrow g\ s \in Q \wedge h\ s \in Q)\} \rrbracket \Longrightarrow$   
 $\Gamma, \Theta \vdash /_F P\ Spec\ (if\text{-rel}\ b\ f\ g\ h)\ Q, A$   
**apply** (rule Spec)  
**apply** (auto simp add: if-rel-def)  
**done**

**lemma Seq [trans, intro?]:**  
 $\llbracket \Gamma, \Theta \vdash /_F P\ c_1\ R, A; \Gamma, \Theta \vdash /_F R\ c_2\ Q, A \rrbracket \Longrightarrow \Gamma, \Theta \vdash /_F P\ (Seq\ c_1\ c_2)\ Q, A$   
**by** (rule hoarep.Seq)

**lemma SeqSame:**  
 $\llbracket \Gamma, \Theta \vdash /_F P\ c_1\ Q, A; \Gamma, \Theta \vdash /_F Q\ c_2\ Q, A \rrbracket \Longrightarrow \Gamma, \Theta \vdash /_F P\ (Seq\ c_1\ c_2)\ Q, A$   
**by** (rule hoarep.Seq)

**lemma SeqSwap:**  
 $\llbracket \Gamma, \Theta \vdash /_F R\ c_2\ Q, A; \Gamma, \Theta \vdash /_F P\ c_1\ R, A \rrbracket \Longrightarrow \Gamma, \Theta \vdash /_F P\ (Seq\ c_1\ c_2)\ Q, A$   
**by** (rule Seq)

**lemma BSeq:**  
 $\llbracket \Gamma, \Theta \vdash /_F P\ c_1\ R, A; \Gamma, \Theta \vdash /_F R\ c_2\ Q, A \rrbracket \Longrightarrow \Gamma, \Theta \vdash /_F P\ (bseq\ c_1\ c_2)\ Q, A$   
**by** (unfold bseq-def) (rule Seq)

**lemma BSeqSame:**  
 $\llbracket \Gamma, \Theta \vdash /_F P\ c_1\ Q, A; \Gamma, \Theta \vdash /_F Q\ c_2\ Q, A \rrbracket \Longrightarrow \Gamma, \Theta \vdash /_F P\ (bseq\ c_1\ c_2)\ Q, A$   
**by** (rule BSeq)

**lemma Cond:**  
**assumes** wp:  $P \subseteq \{s. (s \in b \longrightarrow s \in P_1) \wedge (s \notin b \longrightarrow s \in P_2)\}$   
**assumes** deriv-c1:  $\Gamma, \Theta \vdash /_F P_1\ c_1\ Q, A$   
**assumes** deriv-c2:  $\Gamma, \Theta \vdash /_F P_2\ c_2\ Q, A$   
**shows**  $\Gamma, \Theta \vdash /_F P\ (Cond\ b\ c_1\ c_2)\ Q, A$   
**proof** (rule hoarep.Cond [THEN conseqPre])  
**from** deriv-c1  
**show**  $\Gamma, \Theta \vdash /_F (\{s. (s \in b \longrightarrow s \in P_1) \wedge (s \notin b \longrightarrow s \in P_2)\} \cap b)\ c_1\ Q, A$   
**by** (rule conseqPre) blast  
**next**  
**from** deriv-c2  
**show**  $\Gamma, \Theta \vdash /_F (\{s. (s \in b \longrightarrow s \in P_1) \wedge (s \notin b \longrightarrow s \in P_2)\} \cap -\ b)\ c_2\ Q, A$   
**by** (rule conseqPre) blast  
**next**  
**show**  $P \subseteq \{s. (s \in b \longrightarrow s \in P_1) \wedge (s \notin b \longrightarrow s \in P_2)\}$  **by** (rule wp)  
**qed**

**lemma** *CondSwap*:

$\llbracket \Gamma, \Theta \vdash /_F P1 \ c1 \ Q, A; \Gamma, \Theta \vdash /_F P2 \ c2 \ Q, A; P \subseteq \{s. (s \in b \longrightarrow s \in P1) \wedge (s \notin b \longrightarrow s \in P2)\} \rrbracket$

$\implies$

$\Gamma, \Theta \vdash /_F P \ (Cond \ b \ c1 \ c2) \ Q, A$

**by** (*rule Cond*)

**lemma** *Cond'*:

$\llbracket P \subseteq \{s. (b \subseteq P1) \wedge (- \ b \subseteq P2)\}; \Gamma, \Theta \vdash /_F P1 \ c1 \ Q, A; \Gamma, \Theta \vdash /_F P2 \ c2 \ Q, A \rrbracket$

$\implies$

$\Gamma, \Theta \vdash /_F P \ (Cond \ b \ c1 \ c2) \ Q, A$

**by** (*rule CondSwap*) *blast+*

**lemma** *CondInv*:

**assumes** *wp*:  $P \subseteq Q$

**assumes** *inv*:  $Q \subseteq \{s. (s \in b \longrightarrow s \in P1) \wedge (s \notin b \longrightarrow s \in P2)\}$

**assumes** *deriv-c1*:  $\Gamma, \Theta \vdash /_F P1 \ c1 \ Q, A$

**assumes** *deriv-c2*:  $\Gamma, \Theta \vdash /_F P2 \ c2 \ Q, A$

**shows**  $\Gamma, \Theta \vdash /_F P \ (Cond \ b \ c1 \ c2) \ Q, A$

**proof** –

**from** *wp inv*

**have**  $P \subseteq \{s. (s \in b \longrightarrow s \in P1) \wedge (s \notin b \longrightarrow s \in P2)\}$

**by** *blast*

**from** *Cond* [*OF this deriv-c1 deriv-c2*]

**show** *?thesis* .

**qed**

**lemma** *CondInv'*:

**assumes** *wp*:  $P \subseteq I$

**assumes** *inv*:  $I \subseteq \{s. (s \in b \longrightarrow s \in P1) \wedge (s \notin b \longrightarrow s \in P2)\}$

**assumes** *wp'*:  $I \subseteq Q$

**assumes** *deriv-c1*:  $\Gamma, \Theta \vdash /_F P1 \ c1 \ I, A$

**assumes** *deriv-c2*:  $\Gamma, \Theta \vdash /_F P2 \ c2 \ I, A$

**shows**  $\Gamma, \Theta \vdash /_F P \ (Cond \ b \ c1 \ c2) \ Q, A$

**proof** –

**from** *CondInv* [*OF wp inv deriv-c1 deriv-c2*]

**have**  $\Gamma, \Theta \vdash /_F P \ (Cond \ b \ c1 \ c2) \ I, A$ .

**from** *conseqPost* [*OF this wp' subset-refl*]

**show** *?thesis* .

**qed**

**lemma** *switchNil*:

$P \subseteq Q \implies \Gamma, \Theta \vdash /_F P \ (switch \ v \ []) \ Q, A$

**by** (*simp add: Skip*)

**lemma** *switchCons*:

$\llbracket P \subseteq \{s. (v \ s \in V \longrightarrow s \in P1) \wedge (v \ s \notin V \longrightarrow s \in P2)\}; \rrbracket$

$$\begin{aligned} & \Gamma, \Theta \vdash_{/F} P_1 \ c \ Q, A; \\ & \Gamma, \Theta \vdash_{/F} P_2 \ (\text{switch } v \ \text{vs}) \ Q, A \\ \implies & \Gamma, \Theta \vdash_{/F} P \ (\text{switch } v \ ((V, c) \# \text{vs})) \ Q, A \\ & \text{by } (\text{simp add: Cond}) \end{aligned}$$

**lemma** *Guard*:

$$\begin{aligned} & \llbracket P \subseteq g \cap R; \Gamma, \Theta \vdash_{/F} R \ c \ Q, A \rrbracket \\ \implies & \Gamma, \Theta \vdash_{/F} P \ (\text{Guard } f \ g \ c) \ Q, A \\ \text{apply} & \ (\text{rule } \text{Guard} \ [\text{THEN } \text{conseqPre}, \ \text{of } \text{---} \ R]) \\ \text{apply} & \ (\text{erule } \text{conseqPre}) \\ \text{apply} & \ \text{auto} \\ \text{done} & \end{aligned}$$

**lemma** *GuardSwap*:

$$\begin{aligned} & \llbracket \Gamma, \Theta \vdash_{/F} R \ c \ Q, A; P \subseteq g \cap R \rrbracket \\ \implies & \Gamma, \Theta \vdash_{/F} P \ (\text{Guard } f \ g \ c) \ Q, A \\ \text{by} & \ (\text{rule } \text{Guard}) \end{aligned}$$

**lemma** *Guarantee*:

$$\begin{aligned} & \llbracket P \subseteq \{s. s \in g \longrightarrow s \in R\}; \Gamma, \Theta \vdash_{/F} R \ c \ Q, A; f \in F \rrbracket \\ \implies & \Gamma, \Theta \vdash_{/F} P \ (\text{Guard } f \ g \ c) \ Q, A \\ \text{apply} & \ (\text{rule } \text{Guarantee} \ [\text{THEN } \text{conseqPre}, \ \text{of } \text{-----} \ \{s. s \in g \longrightarrow s \in R\}]) \\ \text{apply} & \ \text{assumption} \\ \text{apply} & \ (\text{erule } \text{conseqPre}) \\ \text{apply} & \ \text{auto} \\ \text{done} & \end{aligned}$$

**lemma** *GuaranteeSwap*:

$$\begin{aligned} & \llbracket \Gamma, \Theta \vdash_{/F} R \ c \ Q, A; P \subseteq \{s. s \in g \longrightarrow s \in R\}; f \in F \rrbracket \\ \implies & \Gamma, \Theta \vdash_{/F} P \ (\text{Guard } f \ g \ c) \ Q, A \\ \text{by} & \ (\text{rule } \text{Guarantee}) \end{aligned}$$

**lemma** *GuardStrip*:

$$\begin{aligned} & \llbracket P \subseteq R; \Gamma, \Theta \vdash_{/F} R \ c \ Q, A; f \in F \rrbracket \\ \implies & \Gamma, \Theta \vdash_{/F} P \ (\text{Guard } f \ g \ c) \ Q, A \\ \text{apply} & \ (\text{rule } \text{Guarantee} \ [\text{THEN } \text{conseqPre}]) \\ \text{apply} & \ \text{auto} \\ \text{done} & \end{aligned}$$

**lemma** *GuardStripSame*:

$$\begin{aligned} & \llbracket \Gamma, \Theta \vdash_{/F} P \ c \ Q, A; f \in F \rrbracket \\ \implies & \Gamma, \Theta \vdash_{/F} P \ (\text{Guard } f \ g \ c) \ Q, A \\ \text{by} & \ (\text{rule } \text{GuardStrip} \ [\text{OF } \text{subset-refl}]) \end{aligned}$$

**lemma** *GuardStripSwap*:

$$\llbracket \Gamma, \Theta \vdash_{/F} R \ c \ Q, A; P \subseteq R; f \in F \rrbracket$$

$\implies \Gamma, \Theta \vdash /_F P \text{ (Guard } f g c) Q, A$   
**by** (rule *GuardStrip*)

**lemma** *GuaranteeStrip*:  
 $\llbracket P \subseteq R; \Gamma, \Theta \vdash /_F R c Q, A; f \in F \rrbracket$   
 $\implies \Gamma, \Theta \vdash /_F P \text{ (guaranteeStrip } f g c) Q, A$   
**by** (unfold *guaranteeStrip-def*) (rule *GuardStrip*)

**lemma** *GuaranteeStripSwap*:  
 $\llbracket \Gamma, \Theta \vdash /_F R c Q, A; P \subseteq R; f \in F \rrbracket$   
 $\implies \Gamma, \Theta \vdash /_F P \text{ (guaranteeStrip } f g c) Q, A$   
**by** (unfold *guaranteeStrip-def*) (rule *GuardStrip*)

**lemma** *GuaranteeAsGuard*:  
 $\llbracket P \subseteq g \cap R; \Gamma, \Theta \vdash /_F R c Q, A \rrbracket$   
 $\implies \Gamma, \Theta \vdash /_F P \text{ (guaranteeStrip } f g c) Q, A$   
**by** (unfold *guaranteeStrip-def*) (rule *Guard*)

**lemma** *GuaranteeAsGuardSwap*:  
 $\llbracket \Gamma, \Theta \vdash /_F R c Q, A; P \subseteq g \cap R \rrbracket$   
 $\implies \Gamma, \Theta \vdash /_F P \text{ (guaranteeStrip } f g c) Q, A$   
**by** (rule *GuaranteeAsGuard*)

**lemma** *GuardsNil*:  
 $\Gamma, \Theta \vdash /_F P c Q, A \implies$   
 $\Gamma, \Theta \vdash /_F P \text{ (guards } [] c) Q, A$   
**by** *simp*

**lemma** *GuardsCons*:  
 $\Gamma, \Theta \vdash /_F P \text{ Guard } f g \text{ (guards } gs c) Q, A \implies$   
 $\Gamma, \Theta \vdash /_F P \text{ (guards } ((f,g)\#gs) c) Q, A$   
**by** *simp*

**lemma** *GuardsConsGuaranteeStrip*:  
 $\Gamma, \Theta \vdash /_F P \text{ guaranteeStrip } f g \text{ (guards } gs c) Q, A \implies$   
 $\Gamma, \Theta \vdash /_F P \text{ (guards (guaranteeStripPair } f g \#gs) c) Q, A$   
**by** (*simp add: guaranteeStripPair-def guaranteeStrip-def*)

**lemma** *While*:  
**assumes** *P-I*:  $P \subseteq I$   
**assumes** *deriv-body*:  $\Gamma, \Theta \vdash /_F (I \cap b) c I, A$   
**assumes** *I-Q*:  $I \cap \neg b \subseteq Q$   
**shows**  $\Gamma, \Theta \vdash /_F P \text{ (whileAnno } b I V c) Q, A$   
**proof** –  
**from** *deriv-body P-I I-Q*

```

show ?thesis
  apply (simp add: whileAnno-def)
  apply (erule conseqPrePost [OF HoarePartialDef.While])
  apply simp-all
  done
qed

```

$J$  will be instantiated by tactic with  $gs' \cap I$  for those guards that are not stripped.

```

lemma WhileAnnoG:
   $\Gamma, \Theta \vdash_F P$  (guards  $gs$ 
    (whileAnno  $b J V$  (Seq  $c$  (guards  $gs$  Skip))))  $Q, A$ 
   $\implies$ 
   $\Gamma, \Theta \vdash_F P$  (whileAnnoG  $gs b I V c$ )  $Q, A$ 
  by (simp add: whileAnnoG-def whileAnno-def while-def)

```

This form stems from *strip-guards*  $F$  (whileAnnoG  $gs b I V c$ )

```

lemma WhileNoGuard':
  assumes  $P-I$ :  $P \subseteq I$ 
  assumes deriv-body:  $\Gamma, \Theta \vdash_F (I \cap b) c I, A$ 
  assumes  $I-Q$ :  $I \cap \neg b \subseteq Q$ 
  shows  $\Gamma, \Theta \vdash_F P$  (whileAnno  $b I V$  (Seq  $c$  Skip))  $Q, A$ 
  apply (rule While [OF  $P-I - I-Q$ ])
  apply (rule Seq)
  apply (rule deriv-body)
  apply (rule hoarep.Skip)
  done

```

```

lemma WhileAnnoFix:
  assumes consequence:  $P \subseteq \{s. (\exists Z. s \in I Z \wedge (I Z \cap \neg b \subseteq Q))\}$ 
  assumes bdy:  $\forall Z. \Gamma, \Theta \vdash_F (I Z \cap b) (c Z) (I Z), A$ 
  assumes bdy-constant:  $\forall Z. c Z = c$  undefined
  shows  $\Gamma, \Theta \vdash_F P$  (whileAnnoFix  $b I V c$ )  $Q, A$ 
  proof -
    from bdy bdy-constant
    have bdy':  $\forall Z. \Gamma, \Theta \vdash_F (I Z \cap b) (c$  undefined)  $(I Z), A$ 
      apply -
      apply (rule allI)
      apply (erule-tac  $x=Z$  in allE)
      apply (erule-tac  $x=Z$  in allE)
      apply simp
      done
    have  $\forall Z. \Gamma, \Theta \vdash_F (I Z) ($ whileAnnoFix  $b I V c) (I Z \cap \neg b), A$ 
      apply rule
      apply (unfold whileAnnoFix-def)
      apply (rule hoarep.While)
      apply (rule bdy' [rule-format])
      done

```



**then**  
**show** *?thesis*  
**apply** (*rule conseq*)  
**using** *consequence*  
**by** *blast*  
**qed**

**lemma** *WhileAnnoFix'*:  
**assumes** *consequence*:  $P \subseteq \{s. (\exists Z. s \in I Z \wedge (\forall t. t \in I Z \cap -b \longrightarrow t \in Q))\}$   
**assumes** *bdy*:  $\forall Z. \Gamma, \Theta \vdash /_F (I Z \cap b) (c Z) (I Z), A$   
**assumes** *bdy-constant*:  $\forall Z. c Z = c \text{ undefined}$   
**shows**  $\Gamma, \Theta \vdash /_F P (\text{whileAnnoFix } b \ I \ V \ c) \ Q, A$   
**apply** (*rule WhileAnnoFix* [*OF - bdy bdy-constant*])  
**using** *consequence* **by** *blast*

**lemma** *WhileAnnoGFix*:  
**assumes** *whileAnnoFix*:  
 $\Gamma, \Theta \vdash /_F P (\text{guards } gs$   
 $\quad (\text{whileAnnoFix } b \ J \ V \ (\lambda Z. (\text{Seq } (c Z) (\text{guards } gs \ \text{Skip})))) \ Q, A$   
**shows**  $\Gamma, \Theta \vdash /_F P (\text{whileAnnoGFix } gs \ b \ I \ V \ c) \ Q, A$   
**using** *whileAnnoFix*  
**by** (*simp add: whileAnnoGFix-def whileAnnoFix-def while-def*)

**lemma** *Bind*:  
**assumes** *adapt*:  $P \subseteq \{s. s \in P' s\}$   
**assumes** *c*:  $\forall s. \Gamma, \Theta \vdash /_F (P' s) (c (e s)) \ Q, A$   
**shows**  $\Gamma, \Theta \vdash /_F P (\text{bind } e \ c) \ Q, A$   
**apply** (*rule conseq* [**where**  $P' = \lambda Z. \{s. s = Z \wedge s \in P' Z\}$  **and**  $Q' = \lambda Z. Q$  **and**  $A' = \lambda Z. A$ ])  
**apply** (*rule allI*)  
**apply** (*unfold bind-def*)  
**apply** (*rule DynCom*)  
**apply** (*rule ballI*)  
**apply** *simp*  
**apply** (*rule conseqPre*)  
**apply** (*rule c* [*rule-format*])  
**apply** *blast*  
**using** *adapt*  
**apply** *blast*  
**done**

**lemma** *Block-exn*:  
**assumes** *adapt*:  $P \subseteq \{s. \text{init } s \in P' s\}$   
**assumes** *bdy*:  $\forall s. \Gamma, \Theta \vdash /_F (P' s) \text{ bdy } \{t. \text{return } s \ t \in R \ s \ t\}, \{t. \text{result-exn } (\text{return } s \ t) \ t \in A\}$   
**assumes** *c*:  $\forall s \ t. \Gamma, \Theta \vdash /_F (R \ s \ t) (c \ s \ t) \ Q, A$   
**shows**  $\Gamma, \Theta \vdash /_F P (\text{block-exn } \text{init } \text{bdy } \text{return } \text{result-exn } \ c) \ Q, A$

```

apply (rule conseq [where  $P'=\lambda Z. \{s. s=Z \wedge \text{init } s \in P' Z\}$  and  $Q'=\lambda Z. Q$  and
 $A'=\lambda Z. A$ ])
prefer 2
using adapt
apply blast
apply (rule allI)
apply (unfold block-exn-def)
apply (rule DynCom)
apply (rule ballI)
apply clarsimp
apply (rule-tac  $R=\{t. \text{return } Z t \in R Z t\}$  in SeqSwap)
apply (rule-tac  $P'=\lambda Z'. \{t. t=Z' \wedge \text{return } Z t \in R Z t\}$  and
 $Q'=\lambda Z'. Q$  and  $A'=\lambda Z'. A$  in conseq)
prefer 2 apply simp
apply (rule allI)
apply (rule DynCom)
apply (clarsimp)
apply (rule SeqSwap)
apply (rule c [rule-format])
apply (rule Basic)
apply clarsimp
apply (rule-tac  $R=\{t. \text{result-exn } (\text{return } Z t) t \in A\}$  in Catch)
apply (rule-tac  $R=\{i. i \in P' Z\}$  in Seq)
apply (rule Basic)
apply clarsimp
apply simp
apply (rule bdy [rule-format])
apply (rule SeqSwap)
apply (rule Throw)
apply (rule Basic)
apply simp
done

```

**lemma** *Block*:

```

assumes adapt:  $P \subseteq \{s. \text{init } s \in P' s\}$ 
assumes bdy:  $\forall s. \Gamma, \Theta \vdash_F (P' s) \text{ bdy } \{t. \text{return } s t \in R s t\}, \{t. \text{return } s t \in A\}$ 
assumes c:  $\forall s t. \Gamma, \Theta \vdash_F (R s t) (c s t) Q, A$ 
shows  $\Gamma, \Theta \vdash_F P (\text{block init bdy return } c) Q, A$ 
  unfolding block-def
  by (rule Block-exn [OF adapt bdy c])

```

**lemma** *BlockSwap*:

```

assumes c:  $\forall s t. \Gamma, \Theta \vdash_F (R s t) (c s t) Q, A$ 
assumes bdy:  $\forall s. \Gamma, \Theta \vdash_F (P' s) \text{ bdy } \{t. \text{return } s t \in R s t\}, \{t. \text{return } s t \in A\}$ 
assumes adapt:  $P \subseteq \{s. \text{init } s \in P' s\}$ 
shows  $\Gamma, \Theta \vdash_F P (\text{block init bdy return } c) Q, A$ 
using adapt bdy c

```

by (rule Block)

**lemma** *Block-exnSpec*:

**assumes** *adapt*:  $P \subseteq \{s. \exists Z. \text{init } s \in P' Z \wedge$   
 $(\forall t. t \in Q' Z \longrightarrow \text{return } s t \in R s t) \wedge$   
 $(\forall t. t \in A' Z \longrightarrow (\text{result-exn } (\text{return } s t) t) \in A)\}$

**assumes** *c*:  $\forall s t. \Gamma, \Theta \vdash_F (R s t) (c s t) Q, A$

**assumes** *bdy*:  $\forall Z. \Gamma, \Theta \vdash_F (P' Z) \text{ bdy } (Q' Z), (A' Z)$

**shows**  $\Gamma, \Theta \vdash_F P (\text{block-exn init bdy return result-exn } c) Q, A$

**apply** (rule *conseq* [where  $P' = \lambda Z. \{s. \text{init } s \in P' Z \wedge$   
 $(\forall t. t \in Q' Z \longrightarrow \text{return } s t \in R s t) \wedge$   
 $(\forall t. t \in A' Z \longrightarrow (\text{result-exn } (\text{return } s t) t) \in A)\}$  **and**  
 $Q' = \lambda Z. Q$  **and**  
 $A' = \lambda Z. A$ ])

**prefer** 2

**using** *adapt*

**apply** *blast*

**apply** (rule *allI*)

**apply** (*unfold block-exn-def*)

**apply** (rule *DynCom*)

**apply** (rule *ballI*)

**apply** *clarsimp*

**apply** (rule-tac  $R = \{t. \text{return } s t \in R s t\}$  **in** *SeqSwap*)

**apply** (rule-tac  $P' = \lambda Z'. \{t. t = Z' \wedge \text{return } s t \in R s t\}$  **and**  
 $Q' = \lambda Z'. Q$  **and**  $A' = \lambda Z'. A$  **in** *conseq*)

**prefer** 2 **apply** *simp*

**apply** (rule *allI*)

**apply** (rule *DynCom*)

**apply** (*clarsimp*)

**apply** (rule *SeqSwap*)

**apply** (rule *c* [rule-format])

**apply** (rule *Basic*)

**apply** *clarsimp*

**apply** (rule-tac  $R = \{t. (\text{result-exn } (\text{return } s t) t) \in A\}$  **in** *Catch*)

**apply** (rule-tac  $R = \{i. i \in P' Z\}$  **in** *Seq*)

**apply** (rule *Basic*)

**apply** *clarsimp*

**apply** *simp*

**apply** (rule *conseq* [OF *bdy*])

**apply** *clarsimp*

**apply** *blast*

**apply** (rule *SeqSwap*)

**apply** (rule *Throw*)

**apply** (rule *Basic*)

**apply** *simp*

**done**

**lemma** *BlockSpec*:

**assumes** *adapt*:  $P \subseteq \{s. \exists Z. \text{init } s \in P' Z \wedge$   
 $(\forall t. t \in Q' Z \longrightarrow \text{return } s t \in R s t) \wedge$   
 $(\forall t. t \in A' Z \longrightarrow \text{return } s t \in A)\}$   
**assumes** *c*:  $\forall s t. \Gamma, \Theta \vdash_F (R s t) (c s t) Q, A$   
**assumes** *bdy*:  $\forall Z. \Gamma, \Theta \vdash_F (P' Z) \text{ bdy } (Q' Z), (A' Z)$   
**shows**  $\Gamma, \Theta \vdash_F P (\text{block init bdy return } c) Q, A$   
**unfolding** *block-def*  
**by** (*rule Block-ernSpec* [*OF adapt c bdy*])

**lemma** *Throw*:  $P \subseteq A \implies \Gamma, \Theta \vdash_F P \text{ Throw } Q, A$   
**by** (*rule hoarep.Throw* [*THEN conseqPre*])

**lemmas** *Catch* = *hoarep.Catch*

**lemma** *CatchSwap*:  $\llbracket \Gamma, \Theta \vdash_F R c_2 Q, A; \Gamma, \Theta \vdash_F P c_1 Q, R \rrbracket \implies \Gamma, \Theta \vdash_F P \text{ Catch}$   
 $c_1 c_2 Q, A$   
**by** (*rule hoarep.Catch*)

**lemma** *CatchSame*:  $\llbracket \Gamma, \Theta \vdash_F P c_1 Q, A; \Gamma, \Theta \vdash_F A c_2 Q, A \rrbracket \implies \Gamma, \Theta \vdash_F P \text{ Catch}$   
 $c_1 c_2 Q, A$   
**by** (*rule hoarep.Catch*)

**lemma** *raise*:  $P \subseteq \{s. f s \in A\} \implies \Gamma, \Theta \vdash_F P \text{ raise } f Q, A$   
**apply** (*simp add: raise-def*)  
**apply** (*rule Seq*)  
**apply** (*rule Basic*)  
**apply** (*assumption*)  
**apply** (*rule Throw*)  
**apply** (*rule subset-refl*)  
**done**

**lemma** *condCatch*:  $\llbracket \Gamma, \Theta \vdash_F P c_1 Q, ((b \cap R) \cup (-b \cap A)); \Gamma, \Theta \vdash_F R c_2 Q, A \rrbracket$   
 $\implies \Gamma, \Theta \vdash_F P \text{ condCatch } c_1 b c_2 Q, A$   
**apply** (*simp add: condCatch-def*)  
**apply** (*rule Catch*)  
**apply** *assumption*  
**apply** (*rule CondSwap*)  
**apply** (*assumption*)  
**apply** (*rule hoarep.Throw*)  
**apply** *blast*  
**done**

**lemma** *condCatchSwap*:  $\llbracket \Gamma, \Theta \vdash_F R c_2 Q, A; \Gamma, \Theta \vdash_F P c_1 Q, ((b \cap R) \cup (-b \cap A)) \rrbracket$   
 $\implies \Gamma, \Theta \vdash_F P \text{ condCatch } c_1 b c_2 Q, A$   
**by** (*rule condCatch*)

**lemma** *condCatchSame*:

**assumes**  $c1: \Gamma, \Theta \vdash_F P \ c_1 \ Q, A$   
**assumes**  $c2: \Gamma, \Theta \vdash_F A \ c_2 \ Q, A$   
**shows**  $\Gamma, \Theta \vdash_F P \ condCatch \ c_1 \ b \ c_2 \ Q, A$   
**proof** –  
**have**  $eq: ((b \cap A) \cup (-b \cap A)) = A$  **by** *blast*  
**show** *?thesis*  
**apply** (*rule condCatch [OF - c2]*)  
**apply** (*simp add: eq*)  
**apply** (*rule c1*)  
**done**  
**qed**

**lemma** *ProcSpec*:  
**assumes**  $adapt: P \subseteq \{s. \exists Z. \text{init } s \in P' \ Z \wedge$   
 $(\forall t. t \in Q' \ Z \longrightarrow \text{return } s \ t \in R \ s \ t) \wedge$   
 $(\forall t. t \in A' \ Z \longrightarrow \text{return } s \ t \in A)\}$   
**assumes**  $c: \forall s \ t. \Gamma, \Theta \vdash_F (R \ s \ t) \ (c \ s \ t) \ Q, A$   
**assumes**  $p: \forall Z. \Gamma, \Theta \vdash_F (P' \ Z) \ Call \ p \ (Q' \ Z), (A' \ Z)$   
**shows**  $\Gamma, \Theta \vdash_F P \ (\text{call } \text{init } p \ \text{return } c) \ Q, A$   
**using** *adapt c p*  
**apply** (*unfold call-def*)  
**by** (*rule BlockSpec*)

**lemma** *Proc-ernSpec*:  
**assumes**  $adapt: P \subseteq \{s. \exists Z. \text{init } s \in P' \ Z \wedge$   
 $(\forall t. t \in Q' \ Z \longrightarrow \text{return } s \ t \in R \ s \ t) \wedge$   
 $(\forall t. t \in A' \ Z \longrightarrow \text{result-ern } (\text{return } s \ t) \ t \in A)\}$   
**assumes**  $c: \forall s \ t. \Gamma, \Theta \vdash_F (R \ s \ t) \ (c \ s \ t) \ Q, A$   
**assumes**  $p: \forall Z. \Gamma, \Theta \vdash_F (P' \ Z) \ Call \ p \ (Q' \ Z), (A' \ Z)$   
**shows**  $\Gamma, \Theta \vdash_F P \ (\text{call-ern } \text{init } p \ \text{return } \text{result-ern } c) \ Q, A$   
**using** *adapt c p*  
**apply** (*unfold call-ern-def*)  
**by** (*rule Block-ernSpec*)

**lemma** *ProcSpec'*:  
**assumes**  $adapt: P \subseteq \{s. \exists Z. \text{init } s \in P' \ Z \wedge$   
 $(\forall t \in Q' \ Z. \text{return } s \ t \in R \ s \ t) \wedge$   
 $(\forall t \in A' \ Z. \text{return } s \ t \in A)\}$   
**assumes**  $c: \forall s \ t. \Gamma, \Theta \vdash_F (R \ s \ t) \ (c \ s \ t) \ Q, A$   
**assumes**  $p: \forall Z. \Gamma, \Theta \vdash_F (P' \ Z) \ Call \ p \ (Q' \ Z), (A' \ Z)$   
**shows**  $\Gamma, \Theta \vdash_F P \ (\text{call } \text{init } p \ \text{return } c) \ Q, A$   
**apply** (*rule ProcSpec [OF - c p]*)  
**apply** (*insert adapt*)  
**apply** *clarsimp*  
**apply** (*drule (1) subsetD*)  
**apply** (*clarsimp*)  
**apply** (*rule-tac x=Z in exI*)

**apply** *blast*  
**done**

**lemma** *Proc-exnSpecNoAbrupt*:

**assumes** *adapt*:  $P \subseteq \{s. \exists Z. \text{init } s \in P' Z \wedge$   
 $(\forall t. t \in Q' Z \longrightarrow \text{return } s t \in R s t)\}$

**assumes** *c*:  $\forall s t. \Gamma, \Theta \vdash_{/F} (R s t) (c s t) Q, A$

**assumes** *p*:  $\forall Z. \Gamma, \Theta \vdash_{/F} (P' Z) \text{ Call } p (Q' Z), \{\}$

**shows**  $\Gamma, \Theta \vdash_{/F} P (\text{call-exn init } p \text{ return result-exn } c) Q, A$

**apply** (*rule Proc-exnSpec [OF - c p]*)

**using** *adapt*

**apply** *simp*

**done**

**lemma** *ProcSpecNoAbrupt*:

**assumes** *adapt*:  $P \subseteq \{s. \exists Z. \text{init } s \in P' Z \wedge$   
 $(\forall t. t \in Q' Z \longrightarrow \text{return } s t \in R s t)\}$

**assumes** *c*:  $\forall s t. \Gamma, \Theta \vdash_{/F} (R s t) (c s t) Q, A$

**assumes** *p*:  $\forall Z. \Gamma, \Theta \vdash_{/F} (P' Z) \text{ Call } p (Q' Z), \{\}$

**shows**  $\Gamma, \Theta \vdash_{/F} P (\text{call init } p \text{ return } c) Q, A$

**apply** (*rule ProcSpec [OF - c p]*)

**using** *adapt*

**apply** *simp*

**done**

**lemma** *FCall*:

$\Gamma, \Theta \vdash_{/F} P (\text{call init } p \text{ return } (\lambda s t. c (\text{result } t))) Q, A$

$\implies \Gamma, \Theta \vdash_{/F} P (\text{fcall init } p \text{ return result } c) Q, A$

**by** (*simp add: fcall-def*)

**lemma** *ProcRec*:

**assumes** *deriv-bodies*:

$\forall p \in \text{Procs.}$

$\forall Z. \Gamma, \Theta \cup (\bigcup p \in \text{Procs. } \bigcup Z. \{(P p Z, p, Q p Z, A p Z)\})$

$\vdash_{/F} (P p Z) (\text{the } (\Gamma p)) (Q p Z), (A p Z)$

**assumes** *Procs-defined*:  $\text{Procs} \subseteq \text{dom } \Gamma$

**shows**  $\forall p \in \text{Procs. } \forall Z. \Gamma, \Theta \vdash_{/F} (P p Z) \text{ Call } p (Q p Z), (A p Z)$

**by** (*intro strip*)

(*rule CallRec'*)

[*OF - Procs-defined deriv-bodies*],

*simp-all*)

**lemma** *ProcRec'*:

**assumes** *ctxt*:  $\Theta' = \Theta \cup (\bigcup p \in \text{Procs. } \bigcup Z. \{(P p Z, p, Q p Z, A p Z)\})$

**assumes** *deriv-bodies*:

$\forall p \in \text{Procs. } \forall Z. \Gamma, \Theta' \vdash_{/F} (P p Z) (\text{the } (\Gamma p)) (Q p Z), (A p Z)$

**assumes** *Procs-defined*:  $Procs \subseteq dom \Gamma$   
**shows**  $\forall p \in Procs. \forall Z. \Gamma, \Theta \vdash_{/F} (P \ p \ Z) \text{ Call } p \ (Q \ p \ Z), (A \ p \ Z)$   
**using** *ctxt deriv-bodies*  
**apply** *simp*  
**apply** (*erule ProcRec [OF - Procs-defined]*)  
**done**

**lemma** *ProcRecList*:

**assumes** *deriv-bodies*:  
 $\forall p \in set \ Procs.$   
 $\forall Z. \Gamma, \Theta \cup (\bigcup p \in set \ Procs. \bigcup Z. \{(P \ p \ Z, p, Q \ p \ Z, A \ p \ Z)\})$   
 $\vdash_{/F} (P \ p \ Z) \ (the \ (\Gamma \ p)) \ (Q \ p \ Z), (A \ p \ Z)$   
**assumes** *dist: distinct Procs*  
**assumes** *Procs-defined*:  $set \ Procs \subseteq dom \Gamma$   
**shows**  $\forall p \in set \ Procs. \forall Z. \Gamma, \Theta \vdash_{/F} (P \ p \ Z) \text{ Call } p \ (Q \ p \ Z), (A \ p \ Z)$   
**using** *deriv-bodies Procs-defined*  
**by** (*rule ProcRec*)

**lemma** *ProcRecSpecs*:

$\llbracket \forall (P, p, Q, A) \in Specs. \Gamma, \Theta \cup Specs \vdash_{/F} P \ (the \ (\Gamma \ p)) \ Q, A;$   
 $\forall (P, p, Q, A) \in Specs. p \in dom \Gamma \rrbracket$   
 $\implies \forall (P, p, Q, A) \in Specs. \Gamma, \Theta \vdash_{/F} P \ (Call \ p) \ Q, A$

**apply** (*auto intro: CallRec*)

**done**

**lemma** *ProcRec1*:

**assumes** *deriv-body*:  
 $\forall Z. \Gamma, \Theta \cup (\bigcup Z. \{(P \ Z, p, Q \ Z, A \ Z)\}) \vdash_{/F} (P \ Z) \ (the \ (\Gamma \ p)) \ (Q \ Z), (A \ Z)$   
**assumes** *p-defined*:  $p \in dom \Gamma$   
**shows**  $\forall Z. \Gamma, \Theta \vdash_{/F} (P \ Z) \text{ Call } p \ (Q \ Z), (A \ Z)$

**proof** –

**from** *deriv-body p-defined*

**have**  $\forall p \in \{p\}. \forall Z. \Gamma, \Theta \vdash_{/F} (P \ Z) \text{ Call } p \ (Q \ Z), (A \ Z)$

**by** – (*rule ProcRec [where A= $\lambda p. A$  and P= $\lambda p. P$  and Q= $\lambda p. Q$ ],  
simp-all)*

**thus** *?thesis*

**by** *simp*

**qed**

**lemma** *ProcNoRec1*:

**assumes** *deriv-body*:  
 $\forall Z. \Gamma, \Theta \vdash_{/F} (P \ Z) \ (the \ (\Gamma \ p)) \ (Q \ Z), (A \ Z)$   
**assumes** *p-def*:  $p \in dom \Gamma$   
**shows**  $\forall Z. \Gamma, \Theta \vdash_{/F} (P \ Z) \text{ Call } p \ (Q \ Z), (A \ Z)$

**proof** –

**from** *deriv-body*

**have**  $\forall Z. \Gamma, \Theta \cup (\bigcup Z. \{(P\ Z, p, Q\ Z, A\ Z)\})$   
 $\vdash_{/F} (P\ Z) (the\ (\Gamma\ p)) (Q\ Z), (A\ Z)$   
**by** (*blast intro: hoare-augment-context*)  
**from** *this p-def*  
**show** *?thesis*  
**by** (*rule ProcRec1*)  
**qed**

**lemma ProcBody:**  
**assumes** *WP: P  $\subseteq$  P'*  
**assumes** *deriv-body:  $\Gamma, \Theta \vdash_{/F} P'$  body Q, A*  
**assumes** *body:  $\Gamma\ p = Some\ body$*   
**shows**  $\Gamma, \Theta \vdash_{/F} P\ Call\ p\ Q, A$   
**apply** (*rule conseqPre [OF - WP]*)  
**apply** (*rule ProcNoRec1 [rule-format, where P= $\lambda$ Z. P' and Q= $\lambda$ Z. Q and A= $\lambda$ Z. A]*)  
**apply** (*insert body*)  
**apply** *simp*  
**apply** (*rule hoare-augment-context [OF deriv-body]*)  
**apply** *blast*  
**apply** *fastforce*  
**done**

**lemma CallBody:**  
**assumes** *adapt: P  $\subseteq$  {s. init s  $\in$  P' s}*  
**assumes** *bdy:  $\forall s. \Gamma, \Theta \vdash_{/F} (P'\ s)\ body\ \{t. return\ s\ t \in R\ s\ t\}, \{t. return\ s\ t \in A\}$*   
**assumes** *c:  $\forall s\ t. \Gamma, \Theta \vdash_{/F} (R\ s\ t)\ (c\ s\ t)\ Q, A$*   
**assumes** *body:  $\Gamma\ p = Some\ body$*   
**shows**  $\Gamma, \Theta \vdash_{/F} P\ (call\ init\ p\ return\ c)\ Q, A$   
**apply** (*unfold call-def*)  
**apply** (*rule Block [OF adapt - c]*)  
**apply** (*rule allI*)  
**apply** (*rule ProcBody [where  $\Gamma = \Gamma$ , OF - bdy [rule-format] body]*)  
**apply** *simp*  
**done**

**lemma Call-exnBody:**  
**assumes** *adapt: P  $\subseteq$  {s. init s  $\in$  P' s}*  
**assumes** *bdy:  $\forall s. \Gamma, \Theta \vdash_{/F} (P'\ s)\ body\ \{t. return\ s\ t \in R\ s\ t\}, \{t. result-exn\ (return\ s\ t)\ t \in A\}$*   
**assumes** *c:  $\forall s\ t. \Gamma, \Theta \vdash_{/F} (R\ s\ t)\ (c\ s\ t)\ Q, A$*   
**assumes** *body:  $\Gamma\ p = Some\ body$*   
**shows**  $\Gamma, \Theta \vdash_{/F} P\ (call-exn\ init\ p\ return\ result-exn\ c)\ Q, A$   
**apply** (*unfold call-exn-def*)  
**apply** (*rule Block-exn [OF adapt - c]*)  
**apply** (*rule allI*)  
**apply** (*rule ProcBody [where  $\Gamma = \Gamma$ , OF - bdy [rule-format] body]*)  
**apply** *simp*



done

**lemmas** *ProcModifyReturn* = *HoarePartialProps.ProcModifyReturn*

**lemmas** *ProcModifyReturnSameFaults* = *HoarePartialProps.ProcModifyReturnSameFaults*

**lemmas** *Proc-exnModifyReturn* = *HoarePartialProps.Proc-exnModifyReturn*

**lemmas** *Proc-exnModifyReturnSameFaults* = *HoarePartialProps.Proc-exnModifyReturnSameFaults*

**lemma** *Proc-exnModifyReturnNoAbr*:

**assumes** *spec*:  $\Gamma, \Theta \vdash_F P$  (*call-exn init p return' result-exn c*)  $Q, A$

**assumes** *result-conform*:

$\forall s t. t \in \text{Modif } (\text{init } s) \longrightarrow (\text{return}' s t) = (\text{return } s t)$

**assumes** *modifies-spec*:

$\forall \sigma. \Gamma, \Theta \vdash_{UNIV} \{\sigma\}$  *Call p (Modif  $\sigma$ ), {}*

**shows**  $\Gamma, \Theta \vdash_F P$  (*call-exn init p return result-exn c*)  $Q, A$

**by** (*rule Proc-exnModifyReturn [OF spec result-conform - modifies-spec]*) *simp*

**lemma** *ProcModifyReturnNoAbr*:

**assumes** *spec*:  $\Gamma, \Theta \vdash_F P$  (*call init p return' c*)  $Q, A$

**assumes** *result-conform*:

$\forall s t. t \in \text{Modif } (\text{init } s) \longrightarrow (\text{return}' s t) = (\text{return } s t)$

**assumes** *modifies-spec*:

$\forall \sigma. \Gamma, \Theta \vdash_{UNIV} \{\sigma\}$  *Call p (Modif  $\sigma$ ), {}*

**shows**  $\Gamma, \Theta \vdash_F P$  (*call init p return c*)  $Q, A$

**by** (*rule ProcModifyReturn [OF spec result-conform - modifies-spec]*) *simp*

**lemma** *Proc-exnModifyReturnNoAbrSameFaults*:

**assumes** *spec*:  $\Gamma, \Theta \vdash_F P$  (*call-exn init p return' result-exn c*)  $Q, A$

**assumes** *result-conform*:

$\forall s t. t \in \text{Modif } (\text{init } s) \longrightarrow (\text{return}' s t) = (\text{return } s t)$

**assumes** *modifies-spec*:

$\forall \sigma. \Gamma, \Theta \vdash_F \{\sigma\}$  *Call p (Modif  $\sigma$ ), {}*

**shows**  $\Gamma, \Theta \vdash_F P$  (*call-exn init p return result-exn c*)  $Q, A$

**by** (*rule Proc-exnModifyReturnSameFaults [OF spec result-conform - modifies-spec]*)  
*simp*

**lemma** *ProcModifyReturnNoAbrSameFaults*:

**assumes** *spec*:  $\Gamma, \Theta \vdash_F P$  (*call init p return' c*)  $Q, A$

**assumes** *result-conform*:

$\forall s t. t \in \text{Modif } (\text{init } s) \longrightarrow (\text{return}' s t) = (\text{return } s t)$

**assumes** *modifies-spec*:

$\forall \sigma. \Gamma, \Theta \vdash_F \{\sigma\}$  *Call p (Modif  $\sigma$ ), {}*

**shows**  $\Gamma, \Theta \vdash_F P$  (*call init p return c*)  $Q, A$

**by** (*rule ProcModifyReturnSameFaults [OF spec result-conform - modifies-spec]*)  
*simp*

**lemma** *DynProc-exn*:

**assumes** *adapt*:  $P \subseteq \{s. \exists Z. \text{init } s \in P' s Z \wedge$

$(\forall t. t \in Q' s Z \longrightarrow \text{return } s t \in R s t) \wedge$   
 $(\forall t. t \in A' s Z \longrightarrow \text{result-ern } (\text{return } s t) t \in A)$

**assumes**  $c: \forall s t. \Gamma, \Theta \vdash_{/F} (R s t) (c s t) Q, A$   
**assumes**  $p: \forall s \in P. \forall Z. \Gamma, \Theta \vdash_{/F} (P' s Z) \text{ Call } (p s) (Q' s Z), (A' s Z)$   
**shows**  $\Gamma, \Theta \vdash_{/F} P \text{ dynCall-ern } f \text{ UNIV init } p \text{ return result-ern } c Q, A$   
**apply** (*rule conseq* [**where**  $P' = \lambda Z. \{s. s = Z \wedge s \in P\}$   
**and**  $Q' = \lambda Z. Q$  **and**  $A' = \lambda Z. A$ ])  
**prefer** 2  
**using** *adapt*  
**apply** *blast*  
**apply** (*rule allI*)  
**apply** (*unfold dynCall-ern-def call-ern-def maybe-guard-UNIV block-ern-def guards.simps*)  
**apply** (*rule DynCom*)  
**apply** *clarsimp*  
**apply** (*rule DynCom*)  
**apply** *clarsimp*  
**apply** (*frule in-mono* [*rule-format*, *OF adapt*])  
**apply** *clarsimp*  
**apply** (*rename-tac Z'*)  
**apply** (*rule-tac R = Q' Z Z' in Seq*)  
**apply** (*rule CatchSwap*)  
**apply** (*rule SeqSwap*)  
**apply** (*rule Throw*)  
**apply** (*rule subset-refl*)  
**apply** (*rule Basic*)  
**apply** (*rule subset-refl*)  
**apply** (*rule-tac R = {i. i \in P' Z Z'} in Seq*)  
**apply** (*rule Basic*)  
**apply** *clarsimp*  
**apply** *simp*  
**apply** (*rule-tac Q' = Q' Z Z' and A' = A' Z Z' in conseqPost*)  
**using**  $p$   
**apply** *clarsimp*  
**apply** *simp*  
**apply** *clarsimp*  
**apply** (*rule-tac P' = \lambda Z''. \{t. t = Z'' \wedge \text{return } Z t \in R Z t\}* **and**  
 $Q' = \lambda Z''. Q$  **and**  $A' = \lambda Z''. A$  **in conseq**)  
**prefer** 2 **apply** *simp*  
**apply** (*rule allI*)  
**apply** (*rule DynCom*)  
**apply** *clarsimp*  
**apply** (*rule SeqSwap*)  
**apply** (*rule c* [*rule-format*])  
**apply** (*rule Basic*)  
**apply** *clarsimp*  
**done**

**lemma** *DynProc-ern-guards-cons*:

**assumes**  $p: \Gamma, \Theta \vdash_{/F} P \text{ dynCall-ern } f \text{ UNIV init } p \text{ return result-ern } c Q, A$

**shows**  $\Gamma, \Theta \vdash_F (g \cap P) \text{ dynCall-exn } f g \text{ init } p \text{ return result-exn } c Q, A$   
**using**  $p$  **apply** (*clarsimp simp add: dynCall-exn-def maybe-guard-def*)  
**apply** (*rule Guard*)  
**apply** (*rule subset-refl*)  
**apply** *assumption*  
**done**

**lemma** *DynProc*:

**assumes** *adapt*:  $P \subseteq \{s. \exists Z. \text{init } s \in P' s Z \wedge$   
 $(\forall t. t \in Q' s Z \longrightarrow \text{return } s t \in R s t) \wedge$   
 $(\forall t. t \in A' s Z \longrightarrow \text{return } s t \in A)\}$   
**assumes**  $c: \forall s t. \Gamma, \Theta \vdash_F (R s t) (c s t) Q, A$   
**assumes**  $p: \forall s \in P. \forall Z. \Gamma, \Theta \vdash_F (P' s Z) \text{ Call } (p s) (Q' s Z), (A' s Z)$   
**shows**  $\Gamma, \Theta \vdash_F P \text{ dynCall init } p \text{ return } c Q, A$   
**using** *adapt c p unfolding dynCall-dynCall-exn by (rule DynProc-exn)*

**lemma** *DynProc-exn'*:

**assumes** *adapt*:  $P \subseteq \{s. \exists Z. \text{init } s \in P' s Z \wedge$   
 $(\forall t \in Q' s Z. \text{return } s t \in R s t) \wedge$   
 $(\forall t \in A' s Z. \text{result-exn } (\text{return } s t) t \in A)\}$   
**assumes**  $c: \forall s t. \Gamma, \Theta \vdash_F (R s t) (c s t) Q, A$   
**assumes**  $p: \forall s \in P. \forall Z. \Gamma, \Theta \vdash_F (P' s Z) \text{ Call } (p s) (Q' s Z), (A' s Z)$   
**shows**  $\Gamma, \Theta \vdash_F P \text{ dynCall-exn } f \text{ UNIV init } p \text{ return result-exn } c Q, A$

**proof** –

**from** *adapt* **have**  $P \subseteq \{s. \exists Z. \text{init } s \in P' s Z \wedge$   
 $(\forall t. t \in Q' s Z \longrightarrow \text{return } s t \in R s t) \wedge$   
 $(\forall t. t \in A' s Z \longrightarrow \text{result-exn } (\text{return } s t) t \in A)\}$

**by** *blast*

**from** *this c p* **show** *?thesis*

**by** (*rule DynProc-exn*)

**qed**

**lemma** *DynProc'*:

**assumes** *adapt*:  $P \subseteq \{s. \exists Z. \text{init } s \in P' s Z \wedge$   
 $(\forall t \in Q' s Z. \text{return } s t \in R s t) \wedge$   
 $(\forall t \in A' s Z. \text{return } s t \in A)\}$   
**assumes**  $c: \forall s t. \Gamma, \Theta \vdash_F (R s t) (c s t) Q, A$   
**assumes**  $p: \forall s \in P. \forall Z. \Gamma, \Theta \vdash_F (P' s Z) \text{ Call } (p s) (Q' s Z), (A' s Z)$   
**shows**  $\Gamma, \Theta \vdash_F P \text{ dynCall init } p \text{ return } c Q, A$   
**using** *adapt c p unfolding dynCall-dynCall-exn by (rule DynProc-exn')*

**lemma** *DynProc-exnStaticSpec*:

**assumes** *adapt*:  $P \subseteq \{s. s \in S \wedge (\exists Z. \text{init } s \in P' Z \wedge$   
 $(\forall \tau. \tau \in Q' Z \longrightarrow \text{return } s \tau \in R s \tau) \wedge$   
 $(\forall \tau. \tau \in A' Z \longrightarrow \text{result-exn } (\text{return } s \tau) \tau \in A)\}$   
**assumes**  $c: \forall s t. \Gamma, \Theta \vdash_F (R s t) (c s t) Q, A$   
**assumes** *spec*:  $\forall s \in S. \forall Z. \Gamma, \Theta \vdash_F (P' Z) \text{ Call } (p s) (Q' Z), (A' Z)$

**shows**  $\Gamma, \Theta \vdash_F P$  (*dynCall-exn f UNIV init p return result-exn c*)  $Q, A$   
**proof** –  
**from** *adapt* **have**  $P \subseteq S$   
**by** *blast*  
**have**  $\Gamma, \Theta \vdash_F (P \cap S)$  (*dynCall-exn f UNIV init p return result-exn c*)  $Q, A$   
**apply** (*rule DynProc-exn* [**where**  $P' = \lambda s Z. P' Z$  **and**  $Q' = \lambda s Z. Q' Z$   
**and**  $A' = \lambda s Z. A' Z, OF - c$ ])  
**apply** *clarsimp*  
**apply** (*frule in-mono* [*rule-format, OF adapt*])  
**apply** *clarsimp*  
**using** *spec*  
**apply** *clarsimp*  
**done**  
**thus** *?thesis*  
**by** (*rule conseqPre*) (*insert P-S, blast*)  
**qed**

**lemma** *DynProcStaticSpec*:  
**assumes** *adapt*:  $P \subseteq \{s. s \in S \wedge (\exists Z. \text{init } s \in P' Z \wedge$   
 $(\forall \tau. \tau \in Q' Z \longrightarrow \text{return } s \tau \in R s \tau) \wedge$   
 $(\forall \tau. \tau \in A' Z \longrightarrow \text{return } s \tau \in A))\}$   
**assumes** *c*:  $\forall s t. \Gamma, \Theta \vdash_F (R s t) (c s t) Q, A$   
**assumes** *spec*:  $\forall s \in S. \forall Z. \Gamma, \Theta \vdash_F (P' Z) \text{Call } (p s) (Q' Z), (A' Z)$   
**shows**  $\Gamma, \Theta \vdash_F P$  (*dynCall init p return c*)  $Q, A$   
**using** *adapt c spec unfolding dynCall-dynCall-exn* **by** (*rule DynProc-exnStaticSpec*)

**lemma** *DynProc-exnProcPar*:  
**assumes** *adapt*:  $P \subseteq \{s. p s = q \wedge (\exists Z. \text{init } s \in P' Z \wedge$   
 $(\forall \tau. \tau \in Q' Z \longrightarrow \text{return } s \tau \in R s \tau) \wedge$   
 $(\forall \tau. \tau \in A' Z \longrightarrow \text{result-exn } (\text{return } s \tau) \tau \in A))\}$   
**assumes** *c*:  $\forall s t. \Gamma, \Theta \vdash_F (R s t) (c s t) Q, A$   
**assumes** *spec*:  $\forall Z. \Gamma, \Theta \vdash_F (P' Z) \text{Call } q (Q' Z), (A' Z)$   
**shows**  $\Gamma, \Theta \vdash_F P$  (*dynCall-exn f UNIV init p return result-exn c*)  $Q, A$   
**apply** (*rule DynProc-exnStaticSpec* [**where**  $S = \{s. p s = q\}$ , *simplified, OF adapt*  
*c*])  
**using** *spec*  
**apply** *simp*  
**done**

**lemma** *DynProcProcPar*:  
**assumes** *adapt*:  $P \subseteq \{s. p s = q \wedge (\exists Z. \text{init } s \in P' Z \wedge$   
 $(\forall \tau. \tau \in Q' Z \longrightarrow \text{return } s \tau \in R s \tau) \wedge$   
 $(\forall \tau. \tau \in A' Z \longrightarrow \text{return } s \tau \in A))\}$   
**assumes** *c*:  $\forall s t. \Gamma, \Theta \vdash_F (R s t) (c s t) Q, A$   
**assumes** *spec*:  $\forall Z. \Gamma, \Theta \vdash_F (P' Z) \text{Call } q (Q' Z), (A' Z)$   
**shows**  $\Gamma, \Theta \vdash_F P$  (*dynCall init p return c*)  $Q, A$   
**apply** (*rule DynProcStaticSpec* [**where**  $S = \{s. p s = q\}$ , *simplified, OF adapt c*])

```

using spec
apply simp
done

```

**lemma** *DynProc-exnProcParNoAbrupt*:

```

assumes adapt:  $P \subseteq \{s. p\ s = q \wedge (\exists Z. \text{init } s \in P' Z \wedge$ 
   $(\forall \tau. \tau \in Q' Z \longrightarrow \text{return } s\ \tau \in R\ s\ \tau))\}$ 
assumes c:  $\forall s\ t. \Gamma, \Theta \vdash_{/F} (R\ s\ t) (c\ s\ t) Q, A$ 
assumes spec:  $\forall Z. \Gamma, \Theta \vdash_{/F} (P' Z) \text{ Call } q (Q' Z), \{\}$ 
shows  $\Gamma, \Theta \vdash_{/F} P (\text{dynCall-exn } f\ \text{UNIV } \text{init } p\ \text{return } \text{result-exn } c) Q, A$ 

```

**proof** –

```

have  $P \subseteq \{s. p\ s = q \wedge (\exists Z. \text{init } s \in P' Z \wedge$ 
   $(\forall t. t \in Q' Z \longrightarrow \text{return } s\ t \in R\ s\ t) \wedge$ 
   $(\forall t. t \in \{\} \longrightarrow \text{result-exn } (\text{return } s\ t) t \in A))\}$ 
  (is  $P \subseteq ?P'$ )

```

**proof**

**fix** *s*

**assume** *P*:  $s \in P$

**with** *adapt* **obtain** *Z* **where**

*Pre*:  $p\ s = q \wedge \text{init } s \in P' Z$  **and**

*adapt-Norm*:  $\forall \tau. \tau \in Q' Z \longrightarrow \text{return } s\ \tau \in R\ s\ \tau$

**by** *blast*

**from** *adapt-Norm*

**have**  $\forall t. t \in Q' Z \longrightarrow \text{return } s\ t \in R\ s\ t$

**by** *auto*

**then**

**show**  $s \in ?P'$

**using** *Pre* **by** *blast*

**qed**

**note** *P* = *this*

**show** *?thesis*

**apply** –

**apply** (*rule DynProc-exnStaticSpec* [**where**  $S = \{s. p\ s = q\}$ , *simplified*, *OF P*

*c*])

**apply** (*insert spec*)

**apply** *auto*

**done**

**qed**

**lemma** *DynProcProcParNoAbrupt*:

```

assumes adapt:  $P \subseteq \{s. p\ s = q \wedge (\exists Z. \text{init } s \in P' Z \wedge$ 
   $(\forall \tau. \tau \in Q' Z \longrightarrow \text{return } s\ \tau \in R\ s\ \tau))\}$ 

```

```

assumes c:  $\forall s\ t. \Gamma, \Theta \vdash_{/F} (R\ s\ t) (c\ s\ t) Q, A$ 

```

```

assumes spec:  $\forall Z. \Gamma, \Theta \vdash_{/F} (P' Z) \text{ Call } q (Q' Z), \{\}$ 

```

```

shows  $\Gamma, \Theta \vdash_{/F} P (\text{dynCall } \text{init } p\ \text{return } c) Q, A$ 

```

**using** *adapt c spec* **unfolding** *dynCall-dynCall-exn* **by** (*rule DynProc-exnProcParNoAbrupt*)

**lemma** *DynProc-exnModifyReturnNoAbr*:

**assumes** *to-prove*:  $\Gamma, \Theta \vdash_F P \text{ (dynCall-exn } f \ g \ \text{init } p \ \text{return}' \ \text{result-exn } c) \ Q, A$   
**assumes** *ret-nrm-modif*:  $\forall s \ t. t \in (\text{Modif } (\text{init } s))$   
 $\longrightarrow \text{return}' \ s \ t = \text{return } s \ t$   
**assumes** *modif-clause*:  
 $\forall s \in P. \forall \sigma. \Gamma, \Theta \vdash_{UNIV} \{\sigma\} \text{ Call } (p \ s) \ (\text{Modif } \sigma), \{\}$   
**shows**  $\Gamma, \Theta \vdash_F P \text{ (dynCall-exn } f \ g \ \text{init } p \ \text{return } \text{result-exn } c) \ Q, A$   
**proof** –  
**from** *ret-nrm-modif*  
**have**  $\forall s \ t. t \in (\text{Modif } (\text{init } s))$   
 $\longrightarrow \text{return}' \ s \ t = \text{return } s \ t$   
**by** *iprover*  
**then**  
**have** *ret-nrm-modif'*:  $\forall s \ t. t \in (\text{Modif } (\text{init } s))$   
 $\longrightarrow \text{return}' \ s \ t = \text{return } s \ t$   
**by** *simp*  
**have** *ret-abr-modif'*:  $\forall s \ t. t \in \{\}$   
 $\longrightarrow \text{result-exn } (\text{return}' \ s \ t) \ t = \text{result-exn } (\text{return } s \ t) \ t$   
**by** *simp*  
**from** *to-prove* *ret-nrm-modif'* *ret-abr-modif'* *modif-clause* **show** *?thesis*  
**by** (*rule* *dynProc-exnModifyReturn*)  
**qed**

**lemma** *DynProcModifyReturnNoAbr*:

**assumes** *to-prove*:  $\Gamma, \Theta \vdash_F P \text{ (dynCall } \text{init } p \ \text{return}' \ c) \ Q, A$   
**assumes** *ret-nrm-modif*:  $\forall s \ t. t \in (\text{Modif } (\text{init } s))$   
 $\longrightarrow \text{return}' \ s \ t = \text{return } s \ t$   
**assumes** *modif-clause*:  
 $\forall s \in P. \forall \sigma. \Gamma, \Theta \vdash_{UNIV} \{\sigma\} \text{ Call } (p \ s) \ (\text{Modif } \sigma), \{\}$   
**shows**  $\Gamma, \Theta \vdash_F P \text{ (dynCall } \text{init } p \ \text{return } c) \ Q, A$   
**using** *to-prove* *ret-nrm-modif* *modif-clause* **unfolding** *dynCall-dynCall-exn*  
**by** (*rule* *DynProc-exnModifyReturnNoAbr*)

**lemma** *ProcDyn-exnModifyReturnNoAbrSameFaults*:

**assumes** *to-prove*:  $\Gamma, \Theta \vdash_F P \text{ (dynCall-exn } f \ g \ \text{init } p \ \text{return}' \ \text{result-exn } c) \ Q, A$   
**assumes** *ret-nrm-modif*:  $\forall s \ t. t \in (\text{Modif } (\text{init } s))$   
 $\longrightarrow \text{return}' \ s \ t = \text{return } s \ t$   
**assumes** *modif-clause*:  
 $\forall s \in P. \forall \sigma. \Gamma, \Theta \vdash_F \{\sigma\} \text{ (Call } (p \ s)) \ (\text{Modif } \sigma), \{\}$   
**shows**  $\Gamma, \Theta \vdash_F P \text{ (dynCall-exn } f \ g \ \text{init } p \ \text{return } \text{result-exn } c) \ Q, A$   
**proof** –  
**from** *ret-nrm-modif*  
**have**  $\forall s \ t. t \in (\text{Modif } (\text{init } s))$   
 $\longrightarrow \text{return}' \ s \ t = \text{return } s \ t$   
**by** *iprover*  
**then**  
**have** *ret-nrm-modif'*:  $\forall s \ t. t \in (\text{Modif } (\text{init } s))$   
 $\longrightarrow \text{return}' \ s \ t = \text{return } s \ t$   
**by** *simp*

**have**  $ret\text{-}abr\text{-}modif'$ :  $\forall s t. t \in \{\}$   
 $\longrightarrow result\text{-}exn (return' s t) t = result\text{-}exn (return s t) t$   
**by** *simp*  
**from** *to-prove*  $ret\text{-}nrm\text{-}modif'$   $ret\text{-}abr\text{-}modif'$  *modif-clause* **show** *?thesis*  
**by** (rule *dynProc-exnModifyReturnSameFaults*)  
**qed**

**lemma** *ProcDynModifyReturnNoAbrSameFaults*:  
**assumes** *to-prove*:  $\Gamma, \Theta \vdash /_F P (dynCall\ init\ p\ return' c) Q, A$   
**assumes**  $ret\text{-}nrm\text{-}modif$ :  $\forall s t. t \in (Modif (init\ s))$   
 $\longrightarrow return' s t = return s t$   
**assumes** *modif-clause*:  
 $\forall s \in P. \forall \sigma. \Gamma, \Theta \vdash /_F \{\sigma\} (Call (p\ s)) (Modif\ \sigma), \{\}$   
**shows**  $\Gamma, \Theta \vdash /_F P (dynCall\ init\ p\ return\ c) Q, A$   
**using** *to-prove*  $ret\text{-}nrm\text{-}modif$  *modif-clause* **unfolding** *dynCall-dynCall-exn*  
**by** (rule *ProcDyn-exnModifyReturnNoAbrSameFaults*)

**lemma** *Proc-exnProcParModifyReturn*:  
**assumes**  $q: P \subseteq \{s. p\ s = q\} \cap P'$   
— *DynProcProcPar* introduces the same constraint as first conjunction in  $P'$ , so the vcg can simplify it.  
**assumes** *to-prove*:  $\Gamma, \Theta \vdash /_F P' (dynCall\text{-}exn\ f\ g\ init\ p\ return' result\text{-}exn\ c) Q, A$   
**assumes**  $ret\text{-}nrm\text{-}modif$ :  $\forall s t. t \in (Modif (init\ s))$   
 $\longrightarrow return' s t = return s t$   
**assumes**  $ret\text{-}abr\text{-}modif$ :  $\forall s t. t \in (ModifAbr (init\ s))$   
 $\longrightarrow result\text{-}exn (return' s t) t = result\text{-}exn (return s t) t$   
**assumes** *modif-clause*:  
 $\forall \sigma. \Gamma, \Theta \vdash /_{UNIV} \{\sigma\} (Call\ q) (Modif\ \sigma), (ModifAbr\ \sigma)$   
**shows**  $\Gamma, \Theta \vdash /_F P (dynCall\text{-}exn\ f\ g\ init\ p\ return\ result\text{-}exn\ c) Q, A$

**proof** —  
**from** *to-prove* **have**  $\Gamma, \Theta \vdash /_F (\{s. p\ s = q\} \cap P')$  (*dynCall-exn f g init p return' result-exn c*)  $Q, A$   
**by** (rule *conseqPre*) *blast*  
**from** *this*  $ret\text{-}nrm\text{-}modif$   
 $ret\text{-}abr\text{-}modif$   
**have**  $\Gamma, \Theta \vdash /_F (\{s. p\ s = q\} \cap P')$  (*dynCall-exn f g init p return result-exn c*)  $Q, A$   
**by** (rule *dynProc-exnModifyReturn*) (*insert modif-clause, auto*)  
**from** *this*  $q$  **show** *?thesis*  
**by** (rule *conseqPre*)  
**qed**

**lemma** *ProcProcParModifyReturn*:  
**assumes**  $q: P \subseteq \{s. p\ s = q\} \cap P'$   
— *DynProcProcPar* introduces the same constraint as first conjunction in  $P'$ , so the vcg can simplify it.  
**assumes** *to-prove*:  $\Gamma, \Theta \vdash /_F P' (dynCall\ init\ p\ return' c) Q, A$   
**assumes**  $ret\text{-}nrm\text{-}modif$ :  $\forall s t. t \in (Modif (init\ s))$

$\longrightarrow \text{return}' s t = \text{return} s t$

**assumes** *ret-abr-modif*:  $\forall s t. t \in (\text{ModifAbr } (\text{init } s))$   
 $\longrightarrow \text{return}' s t = \text{return} s t$

**assumes** *modif-clause*:  
 $\forall \sigma. \Gamma, \Theta \vdash_{UNIV} \{\sigma\} (\text{Call } q) (\text{Modif } \sigma), (\text{ModifAbr } \sigma)$   
**shows**  $\Gamma, \Theta \vdash_F P (\text{dynCall } \text{init } p \text{ return } c) Q, A$

**using** *q to-prove ret-nrm-modif ret-abr-modif modif-clause unfolding dynCall-dynCall-exn*  
**by** (*rule Proc-exnProcParModifyReturn*)

**lemma** *Proc-exnProcParModifyReturnSameFaults*:

**assumes** *q*:  $P \subseteq \{s. p s = q\} \cap P'$

— *DynProcProcPar* introduces the same constraint as first conjunction in  $P'$ , so the vcg can simplify it.

**assumes** *to-prove*:  $\Gamma, \Theta \vdash_F P' (\text{dynCall-exn } f g \text{ init } p \text{ return}' \text{ result-exn } c) Q, A$

**assumes** *ret-nrm-modif*:  $\forall s t. t \in (\text{Modif } (\text{init } s))$

$\longrightarrow \text{return}' s t = \text{return} s t$

**assumes** *ret-abr-modif*:  $\forall s t. t \in (\text{ModifAbr } (\text{init } s))$

$\longrightarrow \text{result-exn } (\text{return}' s t) t = \text{result-exn } (\text{return} s t) t$

**assumes** *modif-clause*:

$\forall \sigma. \Gamma, \Theta \vdash_F \{\sigma\} \text{Call } q (\text{Modif } \sigma), (\text{ModifAbr } \sigma)$

**shows**  $\Gamma, \Theta \vdash_F P (\text{dynCall-exn } f g \text{ init } p \text{ return } \text{result-exn } c) Q, A$

**proof** —

**from** *to-prove*

**have**  $\Gamma, \Theta \vdash_F (\{s. p s = q\} \cap P') (\text{dynCall-exn } f g \text{ init } p \text{ return}' \text{ result-exn } c) Q, A$

**by** (*rule conseqPre*) *blast*

**from** *this ret-nrm-modif*

*ret-abr-modif*

**have**  $\Gamma, \Theta \vdash_F (\{s. p s = q\} \cap P') (\text{dynCall-exn } f g \text{ init } p \text{ return } \text{result-exn } c) Q, A$

**by** (*rule dynProc-exnModifyReturnSameFaults*) (*insert modif-clause, auto*)

**from** *this q show ?thesis*

**by** (*rule conseqPre*)

**qed**

**lemma** *ProcProcParModifyReturnSameFaults*:

**assumes** *q*:  $P \subseteq \{s. p s = q\} \cap P'$

— *DynProcProcPar* introduces the same constraint as first conjunction in  $P'$ , so the vcg can simplify it.

**assumes** *to-prove*:  $\Gamma, \Theta \vdash_F P' (\text{dynCall } \text{init } p \text{ return}' c) Q, A$

**assumes** *ret-nrm-modif*:  $\forall s t. t \in (\text{Modif } (\text{init } s))$

$\longrightarrow \text{return}' s t = \text{return} s t$

**assumes** *ret-abr-modif*:  $\forall s t. t \in (\text{ModifAbr } (\text{init } s))$

$\longrightarrow \text{return}' s t = \text{return} s t$

**assumes** *modif-clause*:

$\forall \sigma. \Gamma, \Theta \vdash_F \{\sigma\} \text{Call } q (\text{Modif } \sigma), (\text{ModifAbr } \sigma)$

**shows**  $\Gamma, \Theta \vdash_F P (\text{dynCall } \text{init } p \text{ return } c) Q, A$

**using** *q to-prove ret-nrm-modif ret-abr-modif modif-clause unfolding dynCall-dynCall-exn*  
**by** (*rule Proc-exnProcParModifyReturnSameFaults*)



**lemma** *Proc-exnProcParModifyReturnNoAbr*:

**assumes**  $q: P \subseteq \{s. p \ s = q\} \cap P'$

— *DynProcProcParNoAbrupt* introduces the same constraint as first conjunction in  $P'$ , so the vcg can simplify it.

**assumes** *to-prove*:  $\Gamma, \Theta \vdash_F P' (dynCall\text{-}exn \ f \ g \ init \ p \ return' \ result\text{-}exn \ c) \ Q, A$

**assumes** *ret-nrm-modif*:  $\forall s \ t. t \in (Modif \ (init \ s))$   
 $\longrightarrow return' \ s \ t = return \ s \ t$

**assumes** *modif-clause*:

$\forall \sigma. \Gamma, \Theta \vdash_{UNIV} \{\sigma\} (Call \ q) (Modif \ \sigma), \{\}$

**shows**  $\Gamma, \Theta \vdash_F P (dynCall\text{-}exn \ f \ g \ init \ p \ return \ result\text{-}exn \ c) \ Q, A$

**proof** —

**from** *to-prove* **have**  $\Gamma, \Theta \vdash_F (\{s. p \ s = q\} \cap P') (dynCall\text{-}exn \ f \ g \ init \ p \ return' \ result\text{-}exn \ c) \ Q, A$

**by** (*rule conseqPre*) *blast*

**from** *this ret-nrm-modif*

**have**  $\Gamma, \Theta \vdash_F (\{s. p \ s = q\} \cap P') (dynCall\text{-}exn \ f \ g \ init \ p \ return \ result\text{-}exn \ c) \ Q, A$

**by** (*rule DynProc-exnModifyReturnNoAbr*) (*insert modif-clause, auto*)

**from** *this q show ?thesis*

**by** (*rule conseqPre*)

**qed**

**lemma** *ProcProcParModifyReturnNoAbr*:

**assumes**  $q: P \subseteq \{s. p \ s = q\} \cap P'$

— *DynProcProcParNoAbrupt* introduces the same constraint as first conjunction in  $P'$ , so the vcg can simplify it.

**assumes** *to-prove*:  $\Gamma, \Theta \vdash_F P' (dynCall \ init \ p \ return' \ c) \ Q, A$

**assumes** *ret-nrm-modif*:  $\forall s \ t. t \in (Modif \ (init \ s))$   
 $\longrightarrow return' \ s \ t = return \ s \ t$

**assumes** *modif-clause*:

$\forall \sigma. \Gamma, \Theta \vdash_{UNIV} \{\sigma\} (Call \ q) (Modif \ \sigma), \{\}$

**shows**  $\Gamma, \Theta \vdash_F P (dynCall \ init \ p \ return \ c) \ Q, A$

**using**  $q$  *to-prove ret-nrm-modif modif-clause unfolding dynCall-dynCall-exn*  
**by** (*rule Proc-exnProcParModifyReturnNoAbr*)

**lemma** *Proc-exnProcParModifyReturnNoAbrSameFaults*:

**assumes**  $q: P \subseteq \{s. p \ s = q\} \cap P'$

— *DynProcProcParNoAbrupt* introduces the same constraint as first conjunction in  $P'$ , so the vcg can simplify it.

**assumes** *to-prove*:  $\Gamma, \Theta \vdash_F P' (dynCall\text{-}exn \ f \ g \ init \ p \ return' \ result\text{-}exn \ c) \ Q, A$

**assumes** *ret-nrm-modif*:  $\forall s \ t. t \in (Modif \ (init \ s))$   
 $\longrightarrow return' \ s \ t = return \ s \ t$

**assumes** *modif-clause*:

$\forall \sigma. \Gamma, \Theta \vdash_F \{\sigma\} (Call \ q) (Modif \ \sigma), \{\}$

**shows**  $\Gamma, \Theta \vdash_F P (dynCall\text{-}exn \ f \ g \ init \ p \ return \ result\text{-}exn \ c) \ Q, A$

**proof** —

**from** *to-prove* **have**

$\Gamma, \Theta \vdash /_F (\{s. p \ s = q\} \cap P')$  (*dynCall-exn f g init p return' result-exn c*)  $Q, A$   
**by** (*rule conseqPre*) *blast*  
**from** *this ret-nrm-modif*  
 $\Gamma, \Theta \vdash /_F (\{s. p \ s = q\} \cap P')$  (*dynCall-exn f g init p return result-exn c*)  $Q, A$   
**by** (*rule ProcDyn-exnModifyReturnNoAbrSameFaults*) (*insert modif-clause, auto*)  
**from** *this q show ?thesis*  
**by** (*rule conseqPre*)  
**qed**

**lemma** *ProcProcParModifyReturnNoAbrSameFaults*:

**assumes**  $q: P \subseteq \{s. p \ s = q\} \cap P'$   
— *DynProcProcParNoAbrupt* introduces the same constraint as first conjunction in  $P'$ , so the vcg can simplify it.  
**assumes** *to-prove*:  $\Gamma, \Theta \vdash /_F P'$  (*dynCall init p return' c*)  $Q, A$   
**assumes** *ret-nrm-modif*:  $\forall s \ t. t \in (\text{Modif } (\text{init } s))$   
 $\longrightarrow \text{return}' \ s \ t = \text{return} \ s \ t$   
**assumes** *modif-clause*:  
 $\forall \sigma. \Gamma, \Theta \vdash /_F \{\sigma\} (\text{Call } q) (\text{Modif } \sigma), \{\}$   
**shows**  $\Gamma, \Theta \vdash /_F P$  (*dynCall init p return c*)  $Q, A$   
**using**  $q$  *to-prove ret-nrm-modif modif-clause unfolding dynCall-dynCall-exn*  
*by (rule Proc-exnProcParModifyReturnNoAbrSameFaults)*

**lemma** *MergeGuards-iff*:  $\Gamma, \Theta \vdash /_F P$  *merge-guards c*  $Q, A = \Gamma, \Theta \vdash /_F P \ c \ Q, A$   
**by** (*auto intro: MergeGuardsI MergeGuardsD*)

**lemma** *CombineStrip'*:

**assumes** *deriv*:  $\Gamma, \Theta \vdash /_F P \ c' \ Q, A$   
**assumes** *deriv-strip-triv*:  $\Gamma, \{\} \vdash /_{\{\}} P \ c'' \ \text{UNIV}, \text{UNIV}$   
**assumes**  $c''$ :  $c'' = \text{mark-guards False } (\text{strip-guards } (-F) \ c')$   
**assumes**  $c$ :  $\text{merge-guards } c = \text{merge-guards } (\text{mark-guards False } \ c')$   
**shows**  $\Gamma, \Theta \vdash /_{\{\}} P \ c \ Q, A$

**proof** —

**from** *deriv-strip-triv* **have** *deriv-strip*:  $\Gamma, \Theta \vdash /_{\{\}} P \ c'' \ \text{UNIV}, \text{UNIV}$   
**by** (*auto intro: hoare-augment-context*)  
**from** *deriv-strip* [*simplified c''*]  
**have**  $\Gamma, \Theta \vdash /_{\{\}} P$  (*strip-guards (- F) c'*)  $\text{UNIV}, \text{UNIV}$   
**by** (*rule MarkGuardsD*)  
**with** *deriv*  
**have**  $\Gamma, \Theta \vdash /_{\{\}} P \ c' \ Q, A$   
**by** (*rule CombineStrip*)  
**hence**  $\Gamma, \Theta \vdash /_{\{\}} P$  *mark-guards False c' Q, A*  
**by** (*rule MarkGuardsI*)  
**hence**  $\Gamma, \Theta \vdash /_{\{\}} P$  *merge-guards (mark-guards False c') Q, A*  
**by** (*rule MergeGuardsI*)  
**hence**  $\Gamma, \Theta \vdash /_{\{\}} P$  *merge-guards c Q, A*  
**by** (*simp add: c*)

**thus** *?thesis*  
**by** (rule *MergeGuardsD*)  
**qed**

**lemma** *CombineStrip''*:  
**assumes** *deriv*:  $\Gamma, \Theta \vdash_{\{\text{True}\}} P \ c' \ Q, A$   
**assumes** *deriv-strip-triv*:  $\Gamma, \{\} \vdash_{\{\}} P \ c'' \ \text{UNIV}, \text{UNIV}$   
**assumes** *c''*:  $c'' = \text{mark-guards } \text{False} \ (\text{strip-guards } (\{\text{False}\}) \ c')$   
**assumes** *c*:  $\text{merge-guards } c = \text{merge-guards } (\text{mark-guards } \text{False} \ c')$   
**shows**  $\Gamma, \Theta \vdash_{\{\}} P \ c \ Q, A$   
**apply** (rule *CombineStrip'* [*OF deriv deriv-strip-triv - c*])  
**apply** (*insert c''*)  
**apply** (*subgoal-tac - \{\text{True}\} = \{\text{False}\}*)  
**apply** *auto*  
**done**

**lemma** *AsmUN*:  
 $(\bigcup Z. \{(P \ Z, p, Q \ Z, A \ Z)\}) \subseteq \Theta$   
 $\implies$   
 $\forall Z. \Gamma, \Theta \vdash_{/F} (P \ Z) \ (\text{Call } p) \ (Q \ Z), (A \ Z)$   
**by** (*blast intro: hoarep.Asm*)

**lemma** *augment-context'*:  
 $\llbracket \Theta \subseteq \Theta'; \forall Z. \Gamma, \Theta \vdash_{/F} (P \ Z) \ p \ (Q \ Z), (A \ Z) \rrbracket$   
 $\implies \forall Z. \Gamma, \Theta \vdash_{/F} (P \ Z) \ p \ (Q \ Z), (A \ Z)$   
**by** (*iprover intro: hoare-augment-context*)

**lemma** *hoarep-strip*:  
 $\llbracket \forall Z. \Gamma, \{\} \vdash_{/F} (P \ Z) \ p \ (Q \ Z), (A \ Z); F' \subseteq -F \rrbracket \implies$   
 $\forall Z. \text{strip } F' \ \Gamma, \{\} \vdash_{/F} (P \ Z) \ p \ (Q \ Z), (A \ Z)$   
**by** (*iprover intro: hoare-strip- $\Gamma$* )

**lemma** *augment-emptyFaults*:  
 $\llbracket \forall Z. \Gamma, \{\} \vdash_{\{\}} (P \ Z) \ p \ (Q \ Z), (A \ Z) \rrbracket \implies$   
 $\forall Z. \Gamma, \{\} \vdash_{/F} (P \ Z) \ p \ (Q \ Z), (A \ Z)$   
**by** (*blast intro: augment-Faults*)

**lemma** *augment-FaultsUNIV*:  
 $\llbracket \forall Z. \Gamma, \{\} \vdash_{/F} (P \ Z) \ p \ (Q \ Z), (A \ Z) \rrbracket \implies$   
 $\forall Z. \Gamma, \{\} \vdash_{/UNIV} (P \ Z) \ p \ (Q \ Z), (A \ Z)$   
**by** (*blast intro: augment-Faults*)

**lemma** *PostConjI* [*trans*]:  
 $\llbracket \Gamma, \Theta \vdash_{/F} P \ c \ Q, A; \Gamma, \Theta \vdash_{/F} P \ c \ R, B \rrbracket \implies \Gamma, \Theta \vdash_{/F} P \ c \ (Q \ \wedge \ R), (A \ \wedge \ B)$   
**by** (rule *PostConjI*)

**lemma** *PostConjI'* :  

$$\llbracket \Gamma, \Theta \vdash /_F P \ c \ Q, A; \Gamma, \Theta \vdash /_F P \ c \ Q, A \implies \Gamma, \Theta \vdash /_F P \ c \ R, B \rrbracket$$

$$\implies \Gamma, \Theta \vdash /_F P \ c \ (Q \cap R), (A \cap B)$$
  
**by** (*rule PostConjI*) *iprover+*

**lemma** *PostConjE* [*consumes I*]:  
**assumes** *conj*:  $\Gamma, \Theta \vdash /_F P \ c \ (Q \cap R), (A \cap B)$   
**assumes** *E*:  $\llbracket \Gamma, \Theta \vdash /_F P \ c \ Q, A; \Gamma, \Theta \vdash /_F P \ c \ R, B \rrbracket \implies S$   
**shows** *S*

**proof** –  
**from** *conj* **have**  $\Gamma, \Theta \vdash /_F P \ c \ Q, A$  **by** (*rule conseqPost*) *blast+*  
**moreover**  
**from** *conj* **have**  $\Gamma, \Theta \vdash /_F P \ c \ R, B$  **by** (*rule conseqPost*) *blast+*  
**ultimately show** *S*  
**by** (*rule E*)

**qed**

## 6.1 Rules for Single-Step Proof

We are now ready to introduce a set of Hoare rules to be used in single-step structured proofs in Isabelle/Isar.

Assertions of Hoare Logic may be manipulated in calculational proofs, with the inclusion expressed in terms of sets or predicates. Reversed order is supported as well.

**lemma** *annotateI* [*trans*]:  

$$\llbracket \Gamma, \Theta \vdash /_F P \ \text{anno } Q, A; \ c = \text{anno} \rrbracket \implies \Gamma, \Theta \vdash /_F P \ c \ Q, A$$
  
**by** *simp*

**lemma** *annotate-normI*:  
**assumes** *deriv-anno*:  $\Gamma, \Theta \vdash /_F P \ \text{anno } Q, A$   
**assumes** *norm-eq*: *normalize c = normalize anno*  
**shows**  $\Gamma, \Theta \vdash /_F P \ c \ Q, A$

**proof** –  
**from** *NormalizeI* [*OF deriv-anno*] *norm-eq*  
**have**  $\Gamma, \Theta \vdash /_F P \ \text{normalize } c \ Q, A$   
**by** *simp*  
**from** *NormalizeD* [*OF this*]  
**show** *?thesis* .

**qed**

**lemma** *annotateWhile*:  

$$\llbracket \Gamma, \Theta \vdash /_F P \ (\text{whileAnnoG } gs \ b \ I \ V \ c) \ Q, A \rrbracket \implies \Gamma, \Theta \vdash /_F P \ (\text{while } gs \ b \ c) \ Q, A$$
  
**by** (*simp add: whileAnnoG-def*)

**lemma** *reannotateWhile*:

$\llbracket \Gamma, \Theta \vdash /_F P \text{ (whileAnnoG gs b I V c) } Q, A \rrbracket \Longrightarrow \Gamma, \Theta \vdash /_F P \text{ (whileAnnoG gs b J V c) } Q, A$   
**by** (*simp add: whileAnnoG-def*)

**lemma** *reannotateWhileNoGuard*:  
 $\llbracket \Gamma, \Theta \vdash /_F P \text{ (whileAnno b I V c) } Q, A \rrbracket \Longrightarrow \Gamma, \Theta \vdash /_F P \text{ (whileAnno b J V c) } Q, A$   
**by** (*simp add: whileAnno-def*)

**lemma** [*trans*]:  $P' \subseteq P \Longrightarrow \Gamma, \Theta \vdash /_F P c Q, A \Longrightarrow \Gamma, \Theta \vdash /_F P' c Q, A$   
**by** (*rule conseqPre*)

**lemma** [*trans*]:  $Q \subseteq Q' \Longrightarrow \Gamma, \Theta \vdash /_F P c Q, A \Longrightarrow \Gamma, \Theta \vdash /_F P c Q', A$   
**by** (*rule conseqPost*) *blast+*

**lemma** [*trans*]:  
 $\Gamma, \Theta \vdash /_F \{s. P s\} c Q, A \Longrightarrow (\bigwedge s. P' s \longrightarrow P s) \Longrightarrow \Gamma, \Theta \vdash /_F \{s. P' s\} c Q, A$   
**by** (*rule conseqPre*) *auto*

**lemma** [*trans*]:  
 $(\bigwedge s. P' s \longrightarrow P s) \Longrightarrow \Gamma, \Theta \vdash /_F \{s. P s\} c Q, A \Longrightarrow \Gamma, \Theta \vdash /_F \{s. P' s\} c Q, A$   
**by** (*rule conseqPre*) *auto*

**lemma** [*trans*]:  
 $\Gamma, \Theta \vdash /_F P c \{s. Q s\}, A \Longrightarrow (\bigwedge s. Q s \longrightarrow Q' s) \Longrightarrow \Gamma, \Theta \vdash /_F P c \{s. Q' s\}, A$   
**by** (*rule conseqPost*) *auto*

**lemma** [*trans*]:  
 $(\bigwedge s. Q s \longrightarrow Q' s) \Longrightarrow \Gamma, \Theta \vdash /_F P c \{s. Q s\}, A \Longrightarrow \Gamma, \Theta \vdash /_F P c \{s. Q' s\}, A$   
**by** (*rule conseqPost*) *auto*

**lemma** [*intro?*]:  $\Gamma, \Theta \vdash /_F P \text{ Skip } P, A$   
**by** (*rule Skip*) *auto*

**lemma** *CondInt* [*trans, intro?*]:  
 $\llbracket \Gamma, \Theta \vdash /_F (P \cap b) c1 Q, A; \Gamma, \Theta \vdash /_F (P \cap \neg b) c2 Q, A \rrbracket$   
 $\Longrightarrow$   
 $\Gamma, \Theta \vdash /_F P \text{ (Cond } b \text{ c1 c2) } Q, A$   
**by** (*rule Cond*) *auto*

**lemma** *CondConj* [*trans, intro?*]:  
 $\llbracket \Gamma, \Theta \vdash /_F \{s. P s \wedge b s\} c1 Q, A; \Gamma, \Theta \vdash /_F \{s. P s \wedge \neg b s\} c2 Q, A \rrbracket$   
 $\Longrightarrow$   
 $\Gamma, \Theta \vdash /_F \{s. P s\} \text{ (Cond } \{s. b s\} \text{ c1 c2) } Q, A$   
**by** (*rule Cond*) *auto*

**lemma** *WhileInvInt* [*intro?*]:  
 $\Gamma, \Theta \vdash /_F (P \cap b) c P, A \Longrightarrow \Gamma, \Theta \vdash /_F P \text{ (whileAnno b P V c) } (P \cap \neg b), A$

by (rule While) auto

**lemma** *WhileInt* [intro?]:

$\Gamma, \Theta \vdash_{/F} (P \cap b) \ c \ P, A$

$\implies$

$\Gamma, \Theta \vdash_{/F} P \ (whileAnno \ b \ \{s. \ undefined\} \ V \ c) \ (P \cap \neg b), A$

by (unfold whileAnno-def)

(rule HoarePartialDef.While [THEN conseqPrePost], auto)

**lemma** *WhileInvConj* [intro?]:

$\Gamma, \Theta \vdash_{/F} \{s. P \ s \wedge b \ s\} \ c \ \{s. P \ s\}, A$

$\implies \Gamma, \Theta \vdash_{/F} \{s. P \ s\} \ (whileAnno \ \{s. b \ s\} \ \{s. P \ s\} \ V \ c) \ \{s. P \ s \wedge \neg b \ s\}, A$

by (simp add: While Collect-conj-eq Collect-neg-eq)

**lemma** *WhileConj* [intro?]:

$\Gamma, \Theta \vdash_{/F} \{s. P \ s \wedge b \ s\} \ c \ \{s. P \ s\}, A$

$\implies$

$\Gamma, \Theta \vdash_{/F} \{s. P \ s\} \ (whileAnno \ \{s. b \ s\} \ \{s. \ undefined\} \ V \ c) \ \{s. P \ s \wedge \neg b \ s\}, A$

by (unfold whileAnno-def)

(simp add: HoarePartialDef.While [THEN conseqPrePost]  
Collect-conj-eq Collect-neg-eq)

end

## 7 Terminating Programs

**theory** *Termination* imports *Semantic* begin

### 7.1 Inductive Characterisation: $\Gamma \vdash c \downarrow s$

**inductive** *terminates*::('s,'p,'f) body  $\Rightarrow$  ('s,'p,'f) com  $\Rightarrow$  ('s,'f) xstate  $\Rightarrow$  bool

( $\langle \vdash \downarrow \rangle$  [60,20,60] 89)

for  $\Gamma$ ::('s,'p,'f) body

where

*Skip*:  $\Gamma \vdash \text{Skip} \downarrow (\text{Normal } s)$

| *Basic*:  $\Gamma \vdash \text{Basic } f \downarrow (\text{Normal } s)$

| *Spec*:  $\Gamma \vdash \text{Spec } r \downarrow (\text{Normal } s)$

| *Guard*:  $\llbracket s \in g; \Gamma \vdash c \downarrow (\text{Normal } s) \rrbracket$

$\implies$

$\Gamma \vdash \text{Guard } f \ g \ c \downarrow (\text{Normal } s)$

| *GuardFault*:  $s \notin g$

$\implies$

$\Gamma \vdash \text{Guard } f \ g \ c \downarrow (\text{Normal } s)$

| *Fault* [*intro,simp*]:  $\Gamma \vdash c \downarrow \text{Fault } f$

| *Seq*:  $\llbracket \Gamma \vdash c_1 \downarrow \text{Normal } s; \forall s'. \Gamma \vdash \langle c_1, \text{Normal } s \rangle \Rightarrow s' \longrightarrow \Gamma \vdash c_2 \downarrow s \rrbracket$   
 $\implies$   
 $\Gamma \vdash \text{Seq } c_1 \ c_2 \downarrow (\text{Normal } s)$

| *CondTrue*:  $\llbracket s \in b; \Gamma \vdash c_1 \downarrow (\text{Normal } s) \rrbracket$   
 $\implies$   
 $\Gamma \vdash \text{Cond } b \ c_1 \ c_2 \downarrow (\text{Normal } s)$

| *CondFalse*:  $\llbracket s \notin b; \Gamma \vdash c_2 \downarrow (\text{Normal } s) \rrbracket$   
 $\implies$   
 $\Gamma \vdash \text{Cond } b \ c_1 \ c_2 \downarrow (\text{Normal } s)$

| *WhileTrue*:  $\llbracket s \in b; \Gamma \vdash c \downarrow (\text{Normal } s);$   
 $\forall s'. \Gamma \vdash \langle c, \text{Normal } s \rangle \Rightarrow s' \longrightarrow \Gamma \vdash \text{While } b \ c \downarrow s \rrbracket$   
 $\implies$   
 $\Gamma \vdash \text{While } b \ c \downarrow (\text{Normal } s)$

| *WhileFalse*:  $\llbracket s \notin b \rrbracket$   
 $\implies$   
 $\Gamma \vdash \text{While } b \ c \downarrow (\text{Normal } s)$

| *Call*:  $\llbracket \Gamma \ p = \text{Some } bdy; \Gamma \vdash bdy \downarrow (\text{Normal } s) \rrbracket$   
 $\implies$   
 $\Gamma \vdash \text{Call } p \downarrow (\text{Normal } s)$

| *CallUndefined*:  $\llbracket \Gamma \ p = \text{None} \rrbracket$   
 $\implies$   
 $\Gamma \vdash \text{Call } p \downarrow (\text{Normal } s)$

| *Stuck* [*intro,simp*]:  $\Gamma \vdash c \downarrow \text{Stuck}$

| *DynCom*:  $\llbracket \Gamma \vdash (c \ s) \downarrow (\text{Normal } s) \rrbracket$   
 $\implies$   
 $\Gamma \vdash \text{DynCom } c \downarrow (\text{Normal } s)$

| *Throw*:  $\Gamma \vdash \text{Throw} \downarrow (\text{Normal } s)$

| *Abrupt* [*intro,simp*]:  $\Gamma \vdash c \downarrow \text{Abrupt } s$

| *Catch*:  $\llbracket \Gamma \vdash c_1 \downarrow \text{Normal } s;$   
 $\forall s'. \Gamma \vdash \langle c_1, \text{Normal } s \rangle \Rightarrow \text{Abrupt } s' \longrightarrow \Gamma \vdash c_2 \downarrow \text{Normal } s \rrbracket$

$$\begin{array}{l} \Longrightarrow \\ \Gamma \vdash \text{Catch } c_1 \ c_2 \downarrow \text{Normal } s \end{array}$$

**inductive-cases** *terminates-elim-cases* [*cases set*]:

$$\begin{array}{l} \Gamma \vdash \text{Skip} \downarrow s \\ \Gamma \vdash \text{Guard } f \ g \ c \downarrow s \\ \Gamma \vdash \text{Basic } f \downarrow s \\ \Gamma \vdash \text{Spec } r \downarrow s \\ \Gamma \vdash \text{Seq } c_1 \ c_2 \downarrow s \\ \Gamma \vdash \text{Cond } b \ c_1 \ c_2 \downarrow s \\ \Gamma \vdash \text{While } b \ c \downarrow s \\ \Gamma \vdash \text{Call } p \downarrow s \\ \Gamma \vdash \text{DynCom } c \downarrow s \\ \Gamma \vdash \text{Throw} \downarrow s \\ \Gamma \vdash \text{Catch } c_1 \ c_2 \downarrow s \end{array}$$

**inductive-cases** *terminates-Normal-elim-cases* [*cases set*]:

$$\begin{array}{l} \Gamma \vdash \text{Skip} \downarrow \text{Normal } s \\ \Gamma \vdash \text{Guard } f \ g \ c \downarrow \text{Normal } s \\ \Gamma \vdash \text{Basic } f \downarrow \text{Normal } s \\ \Gamma \vdash \text{Spec } r \downarrow \text{Normal } s \\ \Gamma \vdash \text{Seq } c_1 \ c_2 \downarrow \text{Normal } s \\ \Gamma \vdash \text{Cond } b \ c_1 \ c_2 \downarrow \text{Normal } s \\ \Gamma \vdash \text{While } b \ c \downarrow \text{Normal } s \\ \Gamma \vdash \text{Call } p \downarrow \text{Normal } s \\ \Gamma \vdash \text{DynCom } c \downarrow \text{Normal } s \\ \Gamma \vdash \text{Throw} \downarrow \text{Normal } s \\ \Gamma \vdash \text{Catch } c_1 \ c_2 \downarrow \text{Normal } s \end{array}$$

**lemma** *terminates-Skip'*:  $\Gamma \vdash \text{Skip} \downarrow s$

**by** (*cases s*) (*auto intro: terminates.intros*)

**lemma** *terminates-Call-body*:

$$\Gamma \ p = \text{Some } \text{bdy} \Longrightarrow \Gamma \vdash \text{Call } p \downarrow s = \Gamma \vdash (\text{the } (\Gamma \ p)) \downarrow s$$

**by** (*cases s*)

(*auto elim: terminates-Normal-elim-cases intro: terminates.intros*)

**lemma** *terminates-Normal-Call-body*:

$$p \in \text{dom } \Gamma \Longrightarrow$$

$$\Gamma \vdash \text{Call } p \downarrow \text{Normal } s = \Gamma \vdash (\text{the } (\Gamma \ p)) \downarrow \text{Normal } s$$

**by** (*auto elim: terminates-Normal-elim-cases intro: terminates.intros*)

**lemma** *terminates-implies-exec*:

**assumes** *terminates*:  $\Gamma \vdash c \downarrow s$

**shows**  $\exists t. \Gamma \vdash \langle c, s \rangle \Rightarrow t$

**using** *terminates*

**proof** (*induct*)

**case** *Skip* **thus** ?*case* **by** (*iprover intro: exec.intros*)



```

next
  case Basic thus ?case by (iprover intro: exec.intros)
next
  case (Spec r s) thus ?case
    by (cases  $\exists t. (s,t) \in r$ ) (auto intro: exec.intros)
next
  case Guard thus ?case by (iprover intro: exec.intros)
next
  case GuardFault thus ?case by (iprover intro: exec.intros)
next
  case Fault thus ?case by (iprover intro: exec.intros)
next
  case Seq thus ?case by (iprover intro: exec-Seq')
next
  case CondTrue thus ?case by (iprover intro: exec.intros)
next
  case CondFalse thus ?case by (iprover intro: exec.intros)
next
  case WhileTrue thus ?case by (iprover intro: exec.intros)
next
  case WhileFalse thus ?case by (iprover intro: exec.intros)
next
  case (Call p bdy s)
  then obtain s' where
     $\Gamma \vdash \langle bdy, Normal\ s \rangle \Rightarrow s'$ 
    by iprover
  moreover have  $\Gamma\ p = Some\ bdy$  by fact
  ultimately show ?case
    by (cases s') (iprover intro: exec.intros)+
next
  case CallUndefined thus ?case by (iprover intro: exec.intros)
next
  case Stuck thus ?case by (iprover intro: exec.intros)
next
  case DynCom thus ?case by (iprover intro: exec.intros)
next
  case Throw thus ?case by (iprover intro: exec.intros)
next
  case Abrupt thus ?case by (iprover intro: exec.intros)
next
  case (Catch c1 s c2)
  then obtain s' where exec-c1:  $\Gamma \vdash \langle c1, Normal\ s \rangle \Rightarrow s'$ 
    by iprover
  thus ?case
  proof (cases s')
    case (Normal s'')
      with exec-c1 show ?thesis by (auto intro!: exec.intros)
  next
    case (Abrupt s'')

```

```

with exec-c1 Catch.hyps
obtain t where  $\Gamma \vdash \langle c2, Normal\ s'' \rangle \Rightarrow t$ 
  by auto
with exec-c1 Abrupt show ?thesis by (auto intro: exec.intros)
next
case Fault
with exec-c1 show ?thesis by (auto intro!: exec.CatchMiss)
next
case Stuck
with exec-c1 show ?thesis by (auto intro!: exec.CatchMiss)
qed
qed

```

```

lemma terminates-block-exn:
[[ $\Gamma \vdash bdy \downarrow Normal\ (init\ s)$ ;
 $\forall t. \Gamma \vdash \langle bdy, Normal\ (init\ s) \rangle \Rightarrow Normal\ t \longrightarrow \Gamma \vdash c\ s\ t \downarrow Normal\ (return\ s\ t)$ ]]
 $\implies \Gamma \vdash block\text{-}exn\ init\ bdy\ return\ result\text{-}exn\ c \downarrow Normal\ s$ 
apply (unfold block-exn-def)
apply (fastforce intro: terminates.intros elim!: exec-Normal-elim-cases
dest!: not-isAbrD)
done

```

```

lemma terminates-block:
[[ $\Gamma \vdash bdy \downarrow Normal\ (init\ s)$ ;
 $\forall t. \Gamma \vdash \langle bdy, Normal\ (init\ s) \rangle \Rightarrow Normal\ t \longrightarrow \Gamma \vdash c\ s\ t \downarrow Normal\ (return\ s\ t)$ ]]
 $\implies \Gamma \vdash block\ init\ bdy\ return\ c \downarrow Normal\ s$ 
unfolding block-def
by (rule terminates-block-exn)

```

```

lemma terminates-block-exn-elim [cases set, consumes 1]:
assumes termi:  $\Gamma \vdash block\text{-}exn\ init\ bdy\ return\ result\text{-}exn\ c \downarrow Normal\ s$ 
assumes e: [[ $\Gamma \vdash bdy \downarrow Normal\ (init\ s)$ ;
 $\forall t. \Gamma \vdash \langle bdy, Normal\ (init\ s) \rangle \Rightarrow Normal\ t \longrightarrow \Gamma \vdash c\ s\ t \downarrow Normal\ (return\ s\ t)$ ]]
shows  $P$ 

```

```

proof -
have  $\Gamma \vdash \langle Basic\ init, Normal\ s \rangle \Rightarrow Normal\ (init\ s)$ 
  by (auto intro: exec.intros)
with termi
have  $\Gamma \vdash bdy \downarrow Normal\ (init\ s)$ 
  apply (unfold block-exn-def)
  apply (elim terminates-Normal-elim-cases)
  by simp
moreover
{
fix t
assume exec-bdy:  $\Gamma \vdash \langle bdy, Normal\ (init\ s) \rangle \Rightarrow Normal\ t$ 
have  $\Gamma \vdash c\ s\ t \downarrow Normal\ (return\ s\ t)$ 

```

```

proof –
  from exec-bdy
  have  $\Gamma \vdash \langle \text{Catch } (\text{Seq } (\text{Basic } \text{init}) \text{ bdy})$ 
       $(\text{Seq } (\text{Basic } (\lambda t. \text{result-exn } (\text{return } s \ t) \ t)) \ \text{Throw}), \text{Normal}$ 
 $s \rangle \Rightarrow \text{Normal } t$ 
    by (fastforce intro: exec.intros)
  with termi have  $\Gamma \vdash \text{DynCom } (\lambda t. \text{Seq } (\text{Basic } (\text{return } s)) \ (c \ s \ t)) \downarrow \text{Normal } t$ 
    apply (unfold block-exn-def)
    apply (elim terminates-Normal-elim-cases)
    by simp
  thus ?thesis
    apply (elim terminates-Normal-elim-cases)
    apply (auto intro: exec.intros)
  done
qed
}
ultimately show  $P$  by (iprover intro: e)
qed

```

```

lemma terminates-block-elim [cases set, consumes 1]:
assumes termi:  $\Gamma \vdash \text{block } \text{init } \text{bdy } \text{return } c \downarrow \text{Normal } s$ 
assumes e:  $\llbracket \Gamma \vdash \text{bdy} \downarrow \text{Normal } (\text{init } s);$ 
     $\forall t. \Gamma \vdash \langle \text{bdy}, \text{Normal } (\text{init } s) \rangle \Rightarrow \text{Normal } t \longrightarrow \Gamma \vdash c \ s \ t \downarrow \text{Normal } (\text{return } s$ 
 $t) \rrbracket \Longrightarrow P$ 
shows  $P$ 
  using termi e unfolding block-def by (rule terminates-block-exn-elim)

```

```

lemma terminates-call:
 $\llbracket \Gamma \ p = \text{Some } \text{bdy}; \Gamma \vdash \text{bdy} \downarrow \text{Normal } (\text{init } s);$ 
 $\forall t. \Gamma \vdash \langle \text{bdy}, \text{Normal } (\text{init } s) \rangle \Rightarrow \text{Normal } t \longrightarrow \Gamma \vdash c \ s \ t \downarrow \text{Normal } (\text{return } s \ t) \rrbracket$ 
 $\Longrightarrow \Gamma \vdash \text{call } \text{init } p \ \text{return } c \downarrow \text{Normal } s$ 
  apply (unfold call-def)
  apply (rule terminates-block)
  apply (iprover intro: terminates.intros)
  apply (auto elim: exec-Normal-elim-cases)
  done

```

```

lemma terminates-call-exn:
 $\llbracket \Gamma \ p = \text{Some } \text{bdy}; \Gamma \vdash \text{bdy} \downarrow \text{Normal } (\text{init } s);$ 
 $\forall t. \Gamma \vdash \langle \text{bdy}, \text{Normal } (\text{init } s) \rangle \Rightarrow \text{Normal } t \longrightarrow \Gamma \vdash c \ s \ t \downarrow \text{Normal } (\text{return } s \ t) \rrbracket$ 
 $\Longrightarrow \Gamma \vdash \text{call-exn } \text{init } p \ \text{return } \text{result-exn } c \downarrow \text{Normal } s$ 
  apply (unfold call-exn-def)
  apply (rule terminates-block-exn)
  apply (iprover intro: terminates.intros)
  apply (auto elim: exec-Normal-elim-cases)
  done

```

**lemma** *terminates-callUndefined*:

```
[[ $\Gamma$   $p = \text{None}$ ]]  
   $\implies \Gamma \vdash \text{call init } p \text{ return result } \downarrow \text{Normal } s$   
  apply (unfold call-def)  
  apply (rule terminates-block)  
  apply (iprover intro: terminates.intros)  
  apply (auto elim: exec-Normal-elim-cases)  
  done
```

**lemma** *terminates-call-exnUndefined*:

```
[[ $\Gamma$   $p = \text{None}$ ]]  
   $\implies \Gamma \vdash \text{call-exn init } p \text{ return result-exn result } \downarrow \text{Normal } s$   
  apply (unfold call-exn-def)  
  apply (rule terminates-block-exn)  
  apply (iprover intro: terminates.intros)  
  apply (auto elim: exec-Normal-elim-cases)  
  done
```

**lemma** *terminates-call-exn-elim* [*cases set, consumes 1*]:

```
assumes termi:  $\Gamma \vdash \text{call-exn init } p \text{ return result-exn } c \downarrow \text{Normal } s$   
assumes bdy:  $\bigwedge \text{bdy}. \llbracket \Gamma \text{ } p = \text{Some bdy}; \Gamma \vdash \text{bdy} \downarrow \text{Normal } (\text{init } s);$   
   $\forall t. \Gamma \vdash \langle \text{bdy}, \text{Normal } (\text{init } s) \rangle \Rightarrow \text{Normal } t \longrightarrow \Gamma \vdash c \text{ } s \text{ } t \downarrow \text{Normal } (\text{return } s \text{ } t) \rrbracket$   
 $\implies P$   
assumes undef:  $\llbracket \Gamma \text{ } p = \text{None} \rrbracket \implies P$   
shows  $P$   
apply (cases  $\Gamma \text{ } p$ )  
apply (erule undef)  
using termi  
apply (unfold call-exn-def)  
apply (erule terminates-block-exn-elim)  
apply (erule terminates-Normal-elim-cases)  
apply simp  
apply (frule (1) bdy)  
apply (fastforce intro: exec.intros)  
apply assumption  
apply simp  
done
```

**lemma** *terminates-call-elim* [*cases set, consumes 1*]:

```
assumes termi:  $\Gamma \vdash \text{call init } p \text{ return } c \downarrow \text{Normal } s$   
assumes bdy:  $\bigwedge \text{bdy}. \llbracket \Gamma \text{ } p = \text{Some bdy}; \Gamma \vdash \text{bdy} \downarrow \text{Normal } (\text{init } s);$   
   $\forall t. \Gamma \vdash \langle \text{bdy}, \text{Normal } (\text{init } s) \rangle \Rightarrow \text{Normal } t \longrightarrow \Gamma \vdash c \text{ } s \text{ } t \downarrow \text{Normal } (\text{return } s \text{ } t) \rrbracket$   
 $\implies P$   
assumes undef:  $\llbracket \Gamma \text{ } p = \text{None} \rrbracket \implies P$   
shows  $P$   
  using termi bdy undef unfolding call-call-exn by (rule terminates-call-exn-elim)
```

**lemma** *terminates-dynCall*:

$\llbracket \Gamma \vdash \text{call init } (p \ s) \ \text{return } c \downarrow \text{Normal } s \rrbracket$   
 $\implies \Gamma \vdash \text{dynCall init } p \ \text{return } c \downarrow \text{Normal } s$   
**apply** (*unfold dynCall-def*)  
**apply** (*auto intro: terminates.intros terminates-call*)  
**done**

**lemma** *terminates-guards*:  $\Gamma \vdash c \downarrow \text{Normal } s \implies \Gamma \vdash \text{guards } gs \ c \downarrow \text{Normal } s$   
**by** (*induct gs*) (*auto intro: terminates.intros*)

**lemma** *terminates-guards-Fault*:  $\text{find } (\lambda(f, g). s \notin g) \ gs = \text{Some } (f, g) \implies \Gamma \vdash \text{guards } gs \ c \downarrow \text{Normal } s$   
**by** (*induct gs*) (*auto intro: terminates.intros split: if-split-asm prod.splits*)

**lemma** *terminates-maybe-guard-Fault*:  $s \notin g \implies \Gamma \vdash \text{maybe-guard } f \ g \ c \downarrow \text{Normal } s$   
**by** (*metis UNIV-I maybe-guard-def terminates.GuardFault*)

**lemma** *terminates-guards-DynCom*:  $\Gamma \vdash (c \ s) \downarrow \text{Normal } s \implies \Gamma \vdash \text{guards } gs \ (\text{DynCom } c) \downarrow \text{Normal } s$   
**by** (*induct gs*) (*auto intro: terminates.intros*)

**lemma** *terminates-maybe-guard-DynCom*:  $\Gamma \vdash (c \ s) \downarrow \text{Normal } s \implies \Gamma \vdash \text{maybe-guard } f \ g \ (\text{DynCom } c) \downarrow \text{Normal } s$   
**by** (*metis maybe-guard-def terminates.DynCom terminates.Guard terminates.GuardFault*)

**lemma** *terminates-dynCall-exn*:  
 $\llbracket \Gamma \vdash \text{call-exn init } (p \ s) \ \text{return result-exn } c \downarrow \text{Normal } s \rrbracket$   
 $\implies \Gamma \vdash \text{dynCall-exn } f \ g \ \text{init } p \ \text{return result-exn } c \downarrow \text{Normal } s$   
**apply** (*unfold dynCall-exn-def*)  
**by** (*rule terminates-maybe-guard-DynCom*)

**lemma** *terminates-dynCall-elim* [*cases set, consumes 1*]:  
**assumes** *termi*:  $\Gamma \vdash \text{dynCall init } p \ \text{return } c \downarrow \text{Normal } s$   
**assumes**  $\llbracket \Gamma \vdash \text{call init } (p \ s) \ \text{return } c \downarrow \text{Normal } s \rrbracket \implies P$   
**shows**  $P$   
**using** *termi*  
**apply** (*unfold dynCall-def*)  
**apply** (*elim terminates-Normal-elim-cases*)  
**apply** *fact*  
**done**

**lemma** *terminates-guards-elim* [*cases set, consumes 1, case-names noFault someFault*]:  
**assumes** *termi*:  $\Gamma \vdash \text{guards } gs \ c \downarrow \text{Normal } s$   
**assumes** *noFault*:  $\llbracket \forall f \ g. (f, g) \in \text{set } gs \longrightarrow s \in g; \Gamma \vdash c \downarrow \text{Normal } s \rrbracket \implies P$   
**assumes** *someFault*:  $\bigwedge f \ g. \text{find } (\lambda(f, g). s \notin g) \ gs = \text{Some } (f, g) \implies P$   
**shows**  $P$   
**using** *termi noFault someFault*  
**by** (*induct gs*)

(*auto elim: terminates-Normal-elim-cases split: if-split-asm prod.splits*)

**lemma** *terminates-maybe-guard-elim* [*cases set, consumes 1, case-names noFault someFault*]:

**assumes** *termi*:  $\Gamma \vdash \text{maybe-guard } f \ g \ c \downarrow \text{Normal } s$   
**assumes** *noFault*:  $\llbracket s \in g; \Gamma \vdash c \downarrow \text{Normal } s \rrbracket \implies P$   
**assumes** *someFault*:  $s \notin g \implies P$   
**shows** *P*  
**using** *termi noFault someFault*  
**by** (*metis maybe-guard-def terminates-Normal-elim-cases(2)*)

**lemma** *terminates-dynCall-exn-elim* [*cases set, consumes 1, case-names noFault someFault*]:

**assumes** *termi*:  $\Gamma \vdash \text{dynCall-exn } f \ g \ \text{init } p \ \text{return result-exn } c \downarrow \text{Normal } s$   
**assumes** *noFault*:  $\llbracket s \in g; \Gamma \vdash \text{call-exn init } (p \ s) \ \text{return result-exn } c \downarrow \text{Normal } s \rrbracket \implies P$   
**assumes** *someFault*:  $s \notin g \implies P$   
**shows** *P*  
**using** *termi noFault someFault*  
**apply** (*unfold dynCall-exn-def*)  
**apply** (*erule terminates-maybe-guard-elim*)  
**apply** (*auto elim: terminates-Normal-elim-cases*)  
**done**

## 7.2 Lemmas about *sequence, flatten and Language.normalize*

**lemma** *terminates-sequence-app*:

$\bigwedge s. \llbracket \Gamma \vdash \text{sequence Seq } xs \downarrow \text{Normal } s; \forall s'. \Gamma \vdash \langle \text{sequence Seq } xs, \text{Normal } s \rangle \Rightarrow s' \longrightarrow \Gamma \vdash \text{sequence Seq } ys \downarrow s' \rrbracket$   
 $\implies \Gamma \vdash \text{sequence Seq } (xs \ @ \ ys) \downarrow \text{Normal } s$   
**proof** (*induct xs*)  
**case** *Nil*  
**thus** *?case by (auto intro: exec.intros)*  
**next**  
**case** (*Cons x xs*)  
**have** *termi-x-xs*:  $\Gamma \vdash \text{sequence Seq } (x \ \# \ xs) \downarrow \text{Normal } s$  **by** *fact*  
**have** *termi-ys*:  $\forall s'. \Gamma \vdash \langle \text{sequence Seq } (x \ \# \ xs), \text{Normal } s \rangle \Rightarrow s' \longrightarrow \Gamma \vdash \text{sequence Seq } ys \downarrow s'$  **by** *fact*  
**show** *?case*  
**proof** (*cases xs*)  
**case** *Nil*  
**with** *termi-x-xs termi-ys* **show** *?thesis*  
**by** (*cases ys*) (*auto intro: terminates.intros*)  
**next**  
**case** *Cons*  
**from** *termi-x-xs Cons*  
**have**  $\Gamma \vdash x \downarrow \text{Normal } s$   
**by** (*auto elim: terminates-Normal-elim-cases*)  
**moreover**

```

{
  fix s'
  assume exec-x:  $\Gamma \vdash \langle x, \text{Normal } s \rangle \Rightarrow s'$ 
  have  $\Gamma \vdash \text{sequence Seq } (xs @ ys) \downarrow s'$ 
  proof -
    from exec-x termi-x-xs Cons
    have termi-xs:  $\Gamma \vdash \text{sequence Seq } xs \downarrow s'$ 
      by (auto elim: terminates-Normal-elim-cases)
    show ?thesis
    proof (cases s')
      case (Normal s'')
      with exec-x termi-ys Cons
      have  $\forall s'. \Gamma \vdash \langle \text{sequence Seq } xs, \text{Normal } s'' \rangle \Rightarrow s' \longrightarrow \Gamma \vdash \text{sequence Seq } ys \downarrow$ 
        s'
        by (auto intro: exec.intros)
      from Cons.hyps [OF termi-xs [simplified Normal] this]
      have  $\Gamma \vdash \text{sequence Seq } (xs @ ys) \downarrow \text{Normal } s''$ .
      with Normal show ?thesis by simp
    next
      case Abrupt thus ?thesis by (auto intro: terminates.intros)
    next
      case Fault thus ?thesis by (auto intro: terminates.intros)
    next
      case Stuck thus ?thesis by (auto intro: terminates.intros)
    qed
  qed
}
ultimately show ?thesis
using Cons
by (auto intro: terminates.intros)
qed
qed

lemma terminates-sequence-appD:
 $\bigwedge s. \Gamma \vdash \text{sequence Seq } (xs @ ys) \downarrow \text{Normal } s$ 
 $\implies \Gamma \vdash \text{sequence Seq } xs \downarrow \text{Normal } s \wedge$ 
 $(\forall s'. \Gamma \vdash \langle \text{sequence Seq } xs, \text{Normal } s \rangle \Rightarrow s' \longrightarrow \Gamma \vdash \text{sequence Seq } ys \downarrow s')$ 
proof (induct xs)
  case Nil
  thus ?case
  by (auto elim: terminates-Normal-elim-cases exec-Normal-elim-cases
    intro: terminates.intros)
next
  case (Cons x xs)
  have termi-x-xs-ys:  $\Gamma \vdash \text{sequence Seq } ((x \# xs) @ ys) \downarrow \text{Normal } s$  by fact
  show ?case
  proof (cases xs)
    case Nil
    with termi-x-xs-ys show ?thesis

```

```

by (cases ys)
  (auto elim: terminates-Normal-elim-cases exec-Normal-elim-cases
    intro: terminates-Skip')
next
case Cons
with termi-x-xs-ys
obtain termi-x:  $\Gamma \vdash x \downarrow \text{Normal } s$  and
  termi-xs-ys:  $\forall s'. \Gamma \vdash \langle x, \text{Normal } s \rangle \Rightarrow s' \longrightarrow \Gamma \vdash \text{sequence Seq } (xs@ys) \downarrow s'$ 
by (auto elim: terminates-Normal-elim-cases)

have  $\Gamma \vdash \text{Seq } x \text{ (sequence Seq } xs) \downarrow \text{Normal } s$ 
proof (rule terminates.Seq [rule-format])
  show  $\Gamma \vdash x \downarrow \text{Normal } s$  by (rule termi-x)
next
fix s'
assume exec-x:  $\Gamma \vdash \langle x, \text{Normal } s \rangle \Rightarrow s'$ 
show  $\Gamma \vdash \text{sequence Seq } xs \downarrow s'$ 
proof -
  from termi-xs-ys [rule-format, OF exec-x]
  have termi-xs-ys':  $\Gamma \vdash \text{sequence Seq } (xs@ys) \downarrow s'$ .
  show ?thesis
  proof (cases s')
    case (Normal s'')
    from Cons.hyps [OF termi-xs-ys' [simplified Normal]]
    show ?thesis
    using Normal by auto
  next
  case Abrupt thus ?thesis by (auto intro: terminates.intros)
  next
  case Fault thus ?thesis by (auto intro: terminates.intros)
  next
  case Stuck thus ?thesis by (auto intro: terminates.intros)
qed
qed
moreover
{
  fix s'
  assume exec-x-xs:  $\Gamma \vdash \langle \text{Seq } x \text{ (sequence Seq } xs), \text{Normal } s \rangle \Rightarrow s'$ 
  have  $\Gamma \vdash \text{sequence Seq } ys \downarrow s'$ 
  proof -
    from exec-x-xs obtain t where
      exec-x:  $\Gamma \vdash \langle x, \text{Normal } s \rangle \Rightarrow t$  and
      exec-xs:  $\Gamma \vdash \langle \text{sequence Seq } xs, t \rangle \Rightarrow s'$ 
    by cases
  show ?thesis
  proof (cases t)
    case (Normal t')
    with exec-x termi-xs-ys have  $\Gamma \vdash \text{sequence Seq } (xs@ys) \downarrow \text{Normal } t'$ 

```



```

      by auto
      from Cons.hyps [OF this] exec-xs Normal
      show ?thesis
      by auto
    next
      case (Abrupt t')
      with exec-xs have s'=Abrupt t'
      by (auto dest: Abrupt-end)
      thus ?thesis by (auto intro: terminates.intros)
    next
      case (Fault f)
      with exec-xs have s'=Fault f
      by (auto dest: Fault-end)
      thus ?thesis by (auto intro: terminates.intros)
    next
      case Stuck
      with exec-xs have s'=Stuck
      by (auto dest: Stuck-end)
      thus ?thesis by (auto intro: terminates.intros)
  qed
}
ultimately show ?thesis
using Cons
by auto
qed
qed

lemma terminates-sequence-appE [consumes 1]:
  [[ $\Gamma \vdash \text{sequence Seq } (xs @ ys) \downarrow \text{Normal } s$ ;
  [[ $\Gamma \vdash \text{sequence Seq } xs \downarrow \text{Normal } s$ ;
   $\forall s'. \Gamma \vdash \langle \text{sequence Seq } xs, \text{Normal } s \rangle \Rightarrow s' \longrightarrow \Gamma \vdash \text{sequence Seq } ys \downarrow s']]] \Longrightarrow P]]
  \Longrightarrow P
  by (auto dest: terminates-sequence-appD)

lemma terminates-to-terminates-sequence-flatten:
  assumes termi:  $\Gamma \vdash c \downarrow s$ 
  shows  $\Gamma \vdash \text{sequence Seq } (\text{flatten } c) \downarrow s$ 
using termi
by (induct)
  (auto intro: terminates.intros terminates-sequence-app
  exec-sequence-flatten-to-exec)

lemma terminates-to-terminates-normalize:
  assumes termi:  $\Gamma \vdash c \downarrow s$ 
  shows  $\Gamma \vdash \text{normalize } c \downarrow s$ 
using termi
proof induct
  case Seq$ 
```

```

thus ?case
  by (fastforce intro: terminates.intros terminates-sequence-app
      terminates-to-terminates-sequence-flatten
      dest: exec-sequence-flatten-to-exec exec-normalize-to-exec)
next
  case WhileTrue
  thus ?case
    by (fastforce intro: terminates.intros terminates-sequence-app
        terminates-to-terminates-sequence-flatten
        dest: exec-sequence-flatten-to-exec exec-normalize-to-exec)
next
  case Catch
  thus ?case
    by (fastforce intro: terminates.intros terminates-sequence-app
        terminates-to-terminates-sequence-flatten
        dest: exec-sequence-flatten-to-exec exec-normalize-to-exec)
qed (auto intro: terminates.intros)

lemma terminates-sequence-flatten-to-terminates:
  shows  $\bigwedge s. \Gamma \vdash \text{sequence Seq (flatten } c) \downarrow s \implies \Gamma \vdash c \downarrow s$ 
proof (induct c)
  case (Seq c1 c2)
  have  $\Gamma \vdash \text{sequence Seq (flatten (Seq c1 c2))} \downarrow s$  by fact
  hence termi-app:  $\Gamma \vdash \text{sequence Seq (flatten } c1 \text{ @ flatten } c2) \downarrow s$  by simp
  show ?case
  proof (cases s)
    case (Normal s')
    have  $\Gamma \vdash \text{Seq } c1 \text{ } c2 \downarrow \text{Normal } s'$ 
    proof (rule terminates.Seq [rule-format])
      from termi-app [simplified Normal]
      have  $\Gamma \vdash \text{sequence Seq (flatten } c1) \downarrow \text{Normal } s'$ 
        by (cases rule: terminates-sequence-appE)
      with Seq.hyps
      show  $\Gamma \vdash c1 \downarrow \text{Normal } s'$ 
        by simp
    next
    fix s''
    assume  $\Gamma \vdash \langle c1, \text{Normal } s' \rangle \Rightarrow s''$ 
    from termi-app [simplified Normal] exec-to-exec-sequence-flatten [OF this]
    have  $\Gamma \vdash \text{sequence Seq (flatten } c2) \downarrow s''$ 
      by (cases rule: terminates-sequence-appE) auto
    with Seq.hyps
    show  $\Gamma \vdash c2 \downarrow s''$ 
      by simp
    qed
  with Normal show ?thesis
    by simp
  qed (auto intro: terminates.intros)
qed (auto intro: terminates.intros)

```

```

lemma terminates-normalize-to-terminates:
  shows  $\bigwedge s. \Gamma \vdash \text{normalize } c \downarrow s \implies \Gamma \vdash c \downarrow s$ 
proof (induct c)
  case Skip thus ?case by (auto intro: terminates-Skip)
next
  case Basic thus ?case by (cases s) (auto intro: terminates.intros)
next
  case Spec thus ?case by (cases s) (auto intro: terminates.intros)
next
  case (Seq c1 c2)
  have  $\Gamma \vdash \text{normalize } (\text{Seq } c1 \text{ } c2) \downarrow s$  by fact
  hence termi-app:  $\Gamma \vdash \text{sequence Seq (flatten (normalize } c1) \text{ @ flatten (normalize } c2)) \downarrow s$ 
  by simp
  show ?case
  proof (cases s)
  case (Normal s')
  have  $\Gamma \vdash \text{Seq } c1 \text{ } c2 \downarrow \text{Normal } s'$ 
  proof (rule terminates.Seq [rule-format])
  from termi-app [simplified Normal]
  have  $\Gamma \vdash \text{sequence Seq (flatten (normalize } c1)) \downarrow \text{Normal } s'$ 
  by (cases rule: terminates-sequence-appE)
  from terminates-sequence-flatten-to-terminates [OF this] Seq.hyps
  show  $\Gamma \vdash c1 \downarrow \text{Normal } s'$ 
  by simp
next
  fix s''
  assume  $\Gamma \vdash \langle c1, \text{Normal } s' \rangle \Rightarrow s''$ 
  from exec-to-exec-normalize [OF this]
  have  $\Gamma \vdash \langle \text{normalize } c1, \text{Normal } s' \rangle \Rightarrow s''$  .
  from termi-app [simplified Normal] exec-to-exec-sequence-flatten [OF this]
  have  $\Gamma \vdash \text{sequence Seq (flatten (normalize } c2)) \downarrow s''$ 
  by (cases rule: terminates-sequence-appE) auto
  from terminates-sequence-flatten-to-terminates [OF this] Seq.hyps
  show  $\Gamma \vdash c2 \downarrow s''$ 
  by simp
  qed
  with Normal show ?thesis by simp
qed (auto intro: terminates.intros)
next
  case (Cond b c1 c2)
  thus ?case
  by (cases s)
  (auto intro: terminates.intros elim!: terminates-Normal-elim-cases)
next
  case (While b c)
  have  $\Gamma \vdash \text{normalize } (\text{While } b \text{ } c) \downarrow s$  by fact
  hence termi-norm-w:  $\Gamma \vdash \text{While } b \text{ (normalize } c) \downarrow s$  by simp

```

```

{
  fix t w
  assume termi-w:  $\Gamma \vdash w \downarrow t$ 
  have w=While b (normalize c)  $\implies \Gamma \vdash \text{While } b \ c \downarrow t$ 
  using termi-w
  proof (induct)
    case (WhileTrue t' b' c')
    from WhileTrue obtain
      t'-b:  $t' \in b$  and
      termi-norm-c:  $\Gamma \vdash \text{normalize } c \downarrow \text{Normal } t'$  and
      termi-norm-w':  $\forall s'. \Gamma \vdash \langle \text{normalize } c, \text{Normal } t' \rangle \Rightarrow s' \longrightarrow \Gamma \vdash \text{While } b \ c \downarrow s'$ 
    by auto
    from While.hyps [OF termi-norm-c]
    have  $\Gamma \vdash c \downarrow \text{Normal } t'$ .
    moreover
    from termi-norm-w'
    have  $\forall s'. \Gamma \vdash \langle c, \text{Normal } t' \rangle \Rightarrow s' \longrightarrow \Gamma \vdash \text{While } b \ c \downarrow s'$ 
      by (auto intro: exec-to-exec-normalize)
    ultimately show ?case
      using t'-b
      by (auto intro: terminates.intros)
    qed (auto intro: terminates.intros)
  }
  from this [OF termi-norm-w]
  show ?case
    by auto
next
  case Call thus ?case by simp
next
  case DynCom thus ?case
    by (cases s) (auto intro: terminates.intros rangeI elim: terminates-Normal-elim-cases)
next
  case Guard thus ?case
    by (cases s) (auto intro: terminates.intros elim: terminates-Normal-elim-cases)
next
  case Throw thus ?case by (cases s) (auto intro: terminates.intros)
next
  case Catch
  thus ?case
    by (cases s)
      (auto dest: exec-to-exec-normalize elim!: terminates-Normal-elim-cases
        intro!: terminates.Catch)
qed

lemma terminates-iff-terminates-normalize:
 $\Gamma \vdash \text{normalize } c \downarrow s = \Gamma \vdash c \downarrow s$ 
  by (auto intro: terminates-to-terminates-normalize
    terminates-normalize-to-terminates)

```

### 7.3 Lemmas about *strip-guards*

**lemma** *terminates-strip-guards-to-terminates*:  $\bigwedge s. \Gamma \vdash \text{strip-guards } F \ c \downarrow s \implies \Gamma \vdash c \downarrow s$

**proof** (*induct c*)

**case** *Skip* **thus** *?case* **by** *simp*

**next**

**case** *Basic* **thus** *?case* **by** *simp*

**next**

**case** *Spec* **thus** *?case* **by** *simp*

**next**

**case** (*Seq c1 c2*)

**hence**  $\Gamma \vdash \text{Seq } (\text{strip-guards } F \ c1) \ (\text{strip-guards } F \ c2) \downarrow s$  **by** *simp*

**thus**  $\Gamma \vdash \text{Seq } c1 \ c2 \downarrow s$

**proof** (*cases*)

**fix** *f* **assume**  $s = \text{Fault } f$  **thus** *?thesis* **by** *simp*

**next**

**assume**  $s = \text{Stuck}$  **thus** *?thesis* **by** *simp*

**next**

**fix** *s'* **assume**  $s = \text{Abrupt } s'$  **thus** *?thesis* **by** *simp*

**next**

**fix** *s'*

**assume**  $s = \text{Normal } s'$

**assume**  $\Gamma \vdash \text{strip-guards } F \ c1 \downarrow \text{Normal } s'$

**hence**  $\Gamma \vdash c1 \downarrow \text{Normal } s'$

**by** (*rule Seq.hyps*)

**moreover**

**assume** *c2*:

$\forall s''. \Gamma \vdash \langle \text{strip-guards } F \ c1, \text{Normal } s' \rangle \Rightarrow s'' \longrightarrow \Gamma \vdash \text{strip-guards } F \ c2 \downarrow s''$

  {

**fix** *s''* **assume** *exec-c1*:  $\Gamma \vdash \langle c1, \text{Normal } s' \rangle \Rightarrow s''$

**have**  $\Gamma \vdash c2 \downarrow s''$

**proof** (*cases s''*)

**case** (*Normal s'''*)

**with** *exec-c1*

**have**  $\Gamma \vdash \langle \text{strip-guards } F \ c1, \text{Normal } s' \rangle \Rightarrow s''$

**by** (*auto intro: exec-to-exec-strip-guards*)

**with** *c2*

**show** *?thesis*

**by** (*iprover intro: Seq.hyps*)

**next**

**case** (*Abrupt s'''*)

**with** *exec-c1*

**have**  $\Gamma \vdash \langle \text{strip-guards } F \ c1, \text{Normal } s' \rangle \Rightarrow s''$

**by** (*auto intro: exec-to-exec-strip-guards*)

**with** *c2*

**show** *?thesis*

**by** (*iprover intro: Seq.hyps*)

**next**

**case** *Fault* **thus** *?thesis* **by** *simp*

**next**

```

      case Stuck thus ?thesis by simp
    qed
  }
  ultimately show ?thesis
  using s
  by (iprover intro: terminates.intros)
qed
next
case (Cond b c1 c2)
hence  $\Gamma \vdash \text{Cond } b \text{ (strip-guards } F \text{ } c1) \text{ (strip-guards } F \text{ } c2) \downarrow s$  by simp
thus  $\Gamma \vdash \text{Cond } b \text{ } c1 \text{ } c2 \downarrow s$ 
proof (cases)
  fix f assume s=Fault f thus ?thesis by simp
next
  assume s=Stuck thus ?thesis by simp
next
  fix s' assume s=Abrupt s' thus ?thesis by simp
next
  fix s'
  assume  $s' \in b \Gamma \vdash \text{strip-guards } F \text{ } c1 \downarrow \text{Normal } s' \text{ } s = \text{Normal } s'$ 
  thus ?thesis
  by (iprover intro: terminates.intros Cond.hyps)
next
  fix s'
  assume  $s' \notin b \Gamma \vdash \text{strip-guards } F \text{ } c2 \downarrow \text{Normal } s' \text{ } s = \text{Normal } s'$ 
  thus ?thesis
  by (iprover intro: terminates.intros Cond.hyps)
qed
next
case (While b c)
have hyp-c:  $\bigwedge s. \Gamma \vdash \text{strip-guards } F \text{ } c \downarrow s \implies \Gamma \vdash c \downarrow s$  by fact
have  $\Gamma \vdash \text{While } b \text{ (strip-guards } F \text{ } c) \downarrow s$  using While.prems by simp
moreover
{
  fix sw
  assume  $\Gamma \vdash sw \downarrow s$ 
  then have sw=While b (strip-guards F c)  $\implies$ 
 $\Gamma \vdash \text{While } b \text{ } c \downarrow s$ 
  proof (induct)
    case (WhileTrue s b' c')
    have eqs: While b' c' = While b (strip-guards F c) by fact
    with  $\langle s \in b' \rangle$  have b:  $s \in b$  by simp
    from eqs  $\langle \Gamma \vdash c' \downarrow \text{Normal } s \rangle$  have  $\Gamma \vdash \text{strip-guards } F \text{ } c \downarrow \text{Normal } s$ 
    by simp
    hence term-c:  $\Gamma \vdash c \downarrow \text{Normal } s$ 
    by (rule hyp-c)
  }
  moreover
  {
    fix t

```

```

assume exec-c:  $\Gamma \vdash \langle c, \text{Normal } s \rangle \Rightarrow t$ 
have  $\Gamma \vdash \text{While } b \ c \ \downarrow \ t$ 
proof (cases t)
  case Fault
  thus ?thesis by simp
next
  case Stuck
  thus ?thesis by simp
next
  case (Abrupt t')
  thus ?thesis by simp
next
  case (Normal t')
  with exec-c
  have  $\Gamma \vdash \langle \text{strip-guards } F \ c, \text{Normal } s \rangle \Rightarrow \text{Normal } t'$ 
    by (auto intro: exec-to-exec-strip-guards)
  with WhileTrue.hyps eqs Normal
  show ?thesis
    by fastforce
  qed
}
ultimately
show ?case
  using b
  by (auto intro: terminates.intros)
next
  case WhileFalse thus ?case by (auto intro: terminates.intros)
qed simp-all
}
ultimately show  $\Gamma \vdash \text{While } b \ c \ \downarrow \ s$ 
  by auto
next
  case Call thus ?case by simp
next
  case DynCom thus ?case
    by (cases s) (auto elim: terminates-Normal-elim-cases intro: terminates.intros
rangeI)
next
  case Guard
  thus ?case
    by (cases s) (auto elim: terminates-Normal-elim-cases intro: terminates.intros
split: if-split-asm)
next
  case Throw thus ?case by simp
next
  case (Catch c1 c2)
  hence  $\Gamma \vdash \text{Catch } (\text{strip-guards } F \ c1) \ (\text{strip-guards } F \ c2) \ \downarrow \ s$  by simp
  thus  $\Gamma \vdash \text{Catch } c1 \ c2 \ \downarrow \ s$ 
  proof (cases)

```

```

    fix f assume s=Fault f thus ?thesis by simp
next
    assume s=Stuck thus ?thesis by simp
next
    fix s' assume s=Abrupt s' thus ?thesis by simp
next
    fix s'
    assume s: s=Normal s'
    assume  $\Gamma \vdash \text{strip-guards } F \ c1 \downarrow \text{Normal } s'$ 
    hence  $\Gamma \vdash c1 \downarrow \text{Normal } s'$ 
      by (rule Catch.hyps)
    moreover
    assume c2:
       $\forall s''. \Gamma \langle \text{strip-guards } F \ c1, \text{Normal } s' \rangle \Rightarrow \text{Abrupt } s''$ 
       $\longrightarrow \Gamma \vdash \text{strip-guards } F \ c2 \downarrow \text{Normal } s''$ 
    {
      fix s'' assume exec-c1:  $\Gamma \langle c1, \text{Normal } s' \rangle \Rightarrow \text{Abrupt } s''$ 
      have  $\Gamma \vdash c2 \downarrow \text{Normal } s''$ 
      proof -
        from exec-c1
        have  $\Gamma \langle \text{strip-guards } F \ c1, \text{Normal } s' \rangle \Rightarrow \text{Abrupt } s''$ 
          by (auto intro: exec-to-exec-strip-guards)
        with c2
        show ?thesis
          by (auto intro: Catch.hyps)
      qed
    }
    ultimately show ?thesis
      using s
      by (iprover intro: terminates.intros)
  qed
qed

```

```

lemma terminates-strip-to-terminates:
  assumes termi-strip: strip F  $\Gamma \vdash c \downarrow s$ 
  shows  $\Gamma \vdash c \downarrow s$ 
using termi-strip
proof induct
  case (Seq c1 s c2)
  have  $\Gamma \vdash c1 \downarrow \text{Normal } s$  by fact
  moreover
  {
    fix s'
    assume exec:  $\Gamma \langle c1, \text{Normal } s \rangle \Rightarrow s'$ 
    have  $\Gamma \vdash c2 \downarrow s'$ 
    proof (cases isFault s')
      case True
      thus ?thesis
        by (auto elim: isFaultE)
    }
  }

```



```

next
  case False
  from exec-to-exec-strip [OF exec this] Seq.hyps
  show ?thesis
  by auto
qed
}
ultimately show ?case
  by (auto intro: terminates.intros)
next
case (WhileTrue s b c)
have  $\Gamma \vdash c \downarrow \text{Normal } s$  by fact
moreover
{
  fix s'
  assume exec:  $\Gamma \vdash \langle c, \text{Normal } s \rangle \Rightarrow s'$ 
  have  $\Gamma \vdash \text{While } b \ c \downarrow s'$ 
  proof (cases isFault s')
  case True
  thus ?thesis
  by (auto elim: isFaultE)
next
case False
from exec-to-exec-strip [OF exec this] WhileTrue.hyps
show ?thesis
by auto
qed
}
ultimately show ?case
  by (auto intro: terminates.intros)
next
case (Catch c1 s c2)
have  $\Gamma \vdash c1 \downarrow \text{Normal } s$  by fact
moreover
{
  fix s'
  assume exec:  $\Gamma \vdash \langle c1, \text{Normal } s \rangle \Rightarrow \text{Abrupt } s'$ 
  from exec-to-exec-strip [OF exec] Catch.hyps
  have  $\Gamma \vdash c2 \downarrow \text{Normal } s'$ 
  by auto
}
ultimately show ?case
  by (auto intro: terminates.intros)
next
case Call thus ?case
  by (auto intro: terminates.intros terminates-strip-guards-to-terminates)
qed (auto intro: terminates.intros)

```

## 7.4 Lemmas about $c_1 \cap_g c_2$

**lemma** *inter-guards-terminates*:

$$\bigwedge c \ c2 \ s. \llbracket (c1 \cap_g c2) = \text{Some } c; \Gamma \vdash c1 \downarrow s \rrbracket \\ \implies \Gamma \vdash c \downarrow s$$

**proof** (*induct c1*)

**case** *Skip* **thus** ?*case* **by** (*fastforce simp add: inter-guards-Skip*)

**next**

**case** (*Basic f*) **thus** ?*case* **by** (*fastforce simp add: inter-guards-Basic*)

**next**

**case** (*Spec r*) **thus** ?*case* **by** (*fastforce simp add: inter-guards-Spec*)

**next**

**case** (*Seq a1 a2*)

**have** (*Seq a1 a2*  $\cap_g$  *c2*) = *Some c* **by** *fact*

**then obtain** *b1 b2 d1 d2* **where**

*c2*: *c2*=*Seq b1 b2* **and**

*d1*: (*a1*  $\cap_g$  *b1*) = *Some d1* **and** *d2*: (*a2*  $\cap_g$  *b2*) = *Some d2* **and**

*c*: *c*=*Seq d1 d2*

**by** (*auto simp add: inter-guards-Seq*)

**have** *termi-c1*:  $\Gamma \vdash \text{Seq } a1 \ a2 \downarrow s$  **by** *fact*

**have**  $\Gamma \vdash \text{Seq } d1 \ d2 \downarrow s$

**proof** (*cases s*)

**case** *Fault* **thus** ?*thesis* **by** *simp*

**next**

**case** *Stuck* **thus** ?*thesis* **by** *simp*

**next**

**case** *Abrupt* **thus** ?*thesis* **by** *simp*

**next**

**case** (*Normal s'*)

**note** *Normal-s* = *this*

**with** *d1 termi-c1*

**have**  $\Gamma \vdash d1 \downarrow \text{Normal } s'$

**by** (*auto elim: terminates-Normal-elim-cases intro: Seq.hyps*)

**moreover**

{

**fix** *t*

**assume** *exec-d1*:  $\Gamma \vdash \langle d1, \text{Normal } s' \rangle \Rightarrow t$

**have**  $\Gamma \vdash d2 \downarrow t$

**proof** (*cases t*)

**case** *Fault* **thus** ?*thesis* **by** *simp*

**next**

**case** *Stuck* **thus** ?*thesis* **by** *simp*

**next**

**case** *Abrupt* **thus** ?*thesis* **by** *simp*

**next**

**case** (*Normal t'*)

**with** *inter-guards-exec-noFault* [*OF d1 exec-d1*]

**have**  $\Gamma \vdash \langle a1, \text{Normal } s' \rangle \Rightarrow \text{Normal } t'$

**by** *simp*

**with** *termi-c1 Normal-s* **have**  $\Gamma \vdash a2 \downarrow \text{Normal } t'$

```

    by (auto elim: terminates-Normal-elim-cases)
  with d2 have  $\Gamma \vdash d2 \downarrow$  Normal t'
    by (auto intro: Seq.hyps)
  with Normal show ?thesis by simp
qed
}
ultimately have  $\Gamma \vdash \text{Seq } d1 \ d2 \downarrow$  Normal s'
  by (fastforce intro: terminates.intros)
  with Normal show ?thesis by simp
qed
with c show ?case by simp
next
case Cond thus ?case
  by - (cases s,
        auto intro: terminates.intros elim!: terminates-Normal-elim-cases
        simp add: inter-guards-Cond)
next
case (While b bdy1)
have (While b bdy1  $\cap_g$  c2) = Some c by fact
then obtain bdy2 bdy where
  c2: c2 = While b bdy2 and
  bdy: (bdy1  $\cap_g$  bdy2) = Some bdy and
  c: c = While b bdy
  by (auto simp add: inter-guards-While)
have  $\Gamma \vdash \text{While } b \ bdy1 \downarrow s$  by fact
moreover
{
  fix s w w1 w2
  assume termi-w:  $\Gamma \vdash w \downarrow s$ 
  assume w: w = While b bdy1
  from termi-w w
  have  $\Gamma \vdash \text{While } b \ bdy \downarrow s$ 
  proof (induct)
    case (WhileTrue s b' bdy1')
    have eqs: While b' bdy1' = While b bdy1 by fact
    from WhileTrue have s-in-b:  $s \in b$  by simp
    from WhileTrue have termi-bdy1:  $\Gamma \vdash bdy1 \downarrow$  Normal s by simp
    show ?case
    proof -
      from bdy termi-bdy1
      have  $\Gamma \vdash bdy \downarrow$  (Normal s)
        by (rule While.hyps)
      moreover
      {
        fix t
        assume exec-bdy:  $\Gamma \vdash \langle bdy, \text{Normal } s \rangle \Rightarrow t$ 
        have  $\Gamma \vdash \text{While } b \ bdy \downarrow t$ 
        proof (cases t)
          case Fault thus ?thesis by simp

```

```

next
  case Stuck thus ?thesis by simp
next
  case Abrupt thus ?thesis by simp
next
  case (Normal t')
  with inter-guards-exec-noFault [OF bdy exec-bdy]
  have  $\Gamma \vdash \langle \text{bdy1}, \text{Normal } s \rangle \Rightarrow \text{Normal } t'$ 
  by simp
  with WhileTrue have  $\Gamma \vdash \text{While } b \text{ bdy} \downarrow \text{Normal } t'$ 
  by simp
  with Normal show ?thesis by simp
qed
}
ultimately show ?thesis
  using s-in-b
  by (blast intro: terminates.WhileTrue)
qed
next
  case WhileFalse thus ?case
  by (blast intro: terminates.WhileFalse)
qed (simp-all)
}
ultimately
show ?case using c by simp
next
  case Call thus ?case by (simp add: inter-guards-Call)
next
  case (DynCom f1)
  have  $(\text{DynCom } f1 \cap_g c2) = \text{Some } c$  by fact
  then obtain f2 f where
    c2: c2=DynCom f2 and
    f-defined:  $\forall s. ((f1 \ s) \cap_g (f2 \ s)) \neq \text{None}$  and
    c: c=DynCom ( $\lambda s. \text{the } ((f1 \ s) \cap_g (f2 \ s))$ )
  by (auto simp add: inter-guards-DynCom)
  have termi:  $\Gamma \vdash \text{DynCom } f1 \downarrow s$  by fact
  show ?case
  proof (cases s)
    case Fault thus ?thesis by simp
  next
    case Stuck thus ?thesis by simp
  next
    case Abrupt thus ?thesis by simp
  next
    case (Normal s')
    from f-defined obtain f where f:  $((f1 \ s') \cap_g (f2 \ s')) = \text{Some } f$ 
    by auto
    from Normal termi
    have  $\Gamma \vdash f1 \ s' \downarrow (\text{Normal } s')$ 

```

```

    by (auto elim: terminates-Normal-elim-cases)
  from DynCom.hyps f this
  have  $\Gamma \vdash f \downarrow$  (Normal s')
    by blast
  with c f Normal
  show ?thesis
    by (auto intro: terminates.intros)
qed
next
case (Guard f g1 bdy1)
have (Guard f g1 bdy1  $\cap_g$  c2) = Some c by fact
then obtain g2 bdy2 bdy where
  c2: c2 = Guard f g2 bdy2 and
  bdy: (bdy1  $\cap_g$  bdy2) = Some bdy and
  c: c = Guard f (g1  $\cap$  g2) bdy
  by (auto simp add: inter-guards-Guard)
have termi-c1:  $\Gamma \vdash$  Guard f g1 bdy1  $\downarrow$  s by fact
show ?case
proof (cases s)
  case Fault thus ?thesis by simp
next
  case Stuck thus ?thesis by simp
next
  case Abrupt thus ?thesis by simp
next
  case (Normal s')
  show ?thesis
  proof (cases s'  $\in$  g1)
    case False
    with Normal c show ?thesis by (auto intro: terminates.GuardFault)
  next
    case True
    note s-in-g1 = this
    show ?thesis
    proof (cases s'  $\in$  g2)
      case False
      with Normal c show ?thesis by (auto intro: terminates.GuardFault)
    next
      case True
      with termi-c1 s-in-g1 Normal have  $\Gamma \vdash$  bdy1  $\downarrow$  Normal s'
        by (auto elim: terminates-Normal-elim-cases)
      with c bdy Guard.hyps Normal True s-in-g1
      show ?thesis by (auto intro: terminates.Guard)
    qed
  qed
qed
next
case Throw thus ?case
  by (auto simp add: inter-guards-Throw)

```

```

next
case (Catch a1 a2)
have (Catch a1 a2  $\cap_g$  c2) = Some c by fact
then obtain b1 b2 d1 d2 where
  c2: c2=Catch b1 b2 and
  d1: (a1  $\cap_g$  b1) = Some d1 and d2: (a2  $\cap_g$  b2) = Some d2 and
  c: c=Catch d1 d2
  by (auto simp add: inter-guards-Catch)
have termi-c1:  $\Gamma \vdash$  Catch a1 a2  $\downarrow$  s by fact
have  $\Gamma \vdash$  Catch d1 d2  $\downarrow$  s
proof (cases s)
  case Fault thus ?thesis by simp
next
  case Stuck thus ?thesis by simp
next
  case Abrupt thus ?thesis by simp
next
  case (Normal s')
  note Normal-s = this
  with d1 termi-c1
  have  $\Gamma \vdash$  d1  $\downarrow$  Normal s'
  by (auto elim: terminates-Normal-elim-cases intro: Catch.hyps)
  moreover
  {
    fix t
    assume exec-d1:  $\Gamma \vdash \langle d1, Normal s' \rangle \Rightarrow$  Abrupt t
    have  $\Gamma \vdash$  d2  $\downarrow$  Normal t
    proof -
      from inter-guards-exec-noFault [OF d1 exec-d1]
      have  $\Gamma \vdash \langle a1, Normal s' \rangle \Rightarrow$  Abrupt t
      by simp
      with termi-c1 Normal-s have  $\Gamma \vdash$  a2  $\downarrow$  Normal t
      by (auto elim: terminates-Normal-elim-cases)
      with d2 have  $\Gamma \vdash$  d2  $\downarrow$  Normal t
      by (auto intro: Catch.hyps)
      with Normal show ?thesis by simp
    qed
  }
  ultimately have  $\Gamma \vdash$  Catch d1 d2  $\downarrow$  Normal s'
  by (fastforce intro: terminates.intros)
  with Normal show ?thesis by simp
qed
with c show ?case by simp
qed

lemma inter-guards-terminates':
  assumes c: (c1  $\cap_g$  c2) = Some c
  assumes termi-c2:  $\Gamma \vdash$  c2  $\downarrow$  s
  shows  $\Gamma \vdash$  c  $\downarrow$  s

```

```

proof –
  from  $c$  have  $(c2 \cap_g c1) = \text{Some } c$ 
    by (rule inter-guards-sym)
  from this termi-c2 show ?thesis
    by (rule inter-guards-terminates)
qed

```

## 7.5 Lemmas about *mark-guards*

**lemma** *terminates-to-terminates-mark-guards*:

```

  assumes termi:  $\Gamma \vdash c \downarrow s$ 
  shows  $\Gamma \vdash \text{mark-guards } f \ c \downarrow s$ 
using termi
proof (induct)
  case Skip thus ?case by (fastforce intro: terminates.intros)
next
  case Basic thus ?case by (fastforce intro: terminates.intros)
next
  case Spec thus ?case by (fastforce intro: terminates.intros)
next
  case Guard thus ?case by (fastforce intro: terminates.intros)
next
  case GuardFault thus ?case by (fastforce intro: terminates.intros)
next
  case Fault thus ?case by (fastforce intro: terminates.intros)
next
  case (Seq c1 s c2)
  have  $\Gamma \vdash \text{mark-guards } f \ c1 \downarrow \text{Normal } s$  by fact
  moreover
  {
    fix  $t$ 
    assume exec-mark:  $\Gamma \vdash \langle \text{mark-guards } f \ c1, \text{Normal } s \rangle \Rightarrow t$ 
    have  $\Gamma \vdash \text{mark-guards } f \ c2 \downarrow t$ 
    proof –
      from exec-mark-guards-to-exec [OF exec-mark] obtain  $t'$  where
        exec-c1:  $\Gamma \vdash \langle c1, \text{Normal } s \rangle \Rightarrow t'$  and
        t-Fault:  $\text{isFault } t \longrightarrow \text{isFault } t'$  and
        t'-Fault-f:  $t' = \text{Fault } f \longrightarrow t' = t$  and
        t'-Fault:  $\text{isFault } t' \longrightarrow \text{isFault } t$  and
        t'-noFault:  $\neg \text{isFault } t' \longrightarrow t' = t$ 
      by blast
    show ?thesis
    proof (cases isFault t')
    case True
    with t'-Fault have  $\text{isFault } t$  by simp
    thus ?thesis
    by (auto elim: isFaultE)
  }
next
  case False

```

```

    with  $t'$ -noFault have  $t'=t$  by simp
    with exec-c1 Seq.hyps
    show ?thesis
      by auto
    qed
  qed
}
ultimately show ?case
  by (auto intro: terminates.intros)
next
case CondTrue thus ?case by (fastforce intro: terminates.intros)
next
case CondFalse thus ?case by (fastforce intro: terminates.intros)
next
case (WhileTrue  $s b c$ )
have s-in-b:  $s \in b$  by fact
have  $\Gamma \vdash \text{mark-guards } f c \downarrow \text{Normal } s$  by fact
moreover
{
  fix  $t$ 
  assume exec-mark:  $\Gamma \vdash \langle \text{mark-guards } f c, \text{Normal } s \rangle \Rightarrow t$ 
  have  $\Gamma \vdash \text{mark-guards } f (While\ b\ c) \downarrow t$ 
  proof -
    from exec-mark-guards-to-exec [OF exec-mark] obtain  $t'$  where
      exec-c1:  $\Gamma \vdash \langle c, \text{Normal } s \rangle \Rightarrow t'$  and
      t-Fault:  $\text{isFault } t \longrightarrow \text{isFault } t'$  and
      t'-Fault-f:  $t' = \text{Fault } f \longrightarrow t' = t$  and
      t'-Fault:  $\text{isFault } t' \longrightarrow \text{isFault } t$  and
      t'-noFault:  $\neg \text{isFault } t' \longrightarrow t' = t$ 
    by blast
  show ?thesis
  proof (cases  $\text{isFault } t'$ )
    case True
    with t'-Fault have  $\text{isFault } t$  by simp
    thus ?thesis
      by (auto elim: isFaultE)
    next
    case False
    with t'-noFault have  $t'=t$  by simp
    with exec-c1 WhileTrue.hyps
    show ?thesis
      by auto
    qed
  qed
}
ultimately show ?case
  by (auto intro: terminates.intros)
next
case WhileFalse thus ?case by (fastforce intro: terminates.intros)

```



```

next
  case Call thus ?case by (fastforce intro: terminates.intros)
next
  case CallUndefined thus ?case by (fastforce intro: terminates.intros)
next
  case Stuck thus ?case by (fastforce intro: terminates.intros)
next
  case DynCom thus ?case by (fastforce intro: terminates.intros)
next
  case Throw thus ?case by (fastforce intro: terminates.intros)
next
  case Abrupt thus ?case by (fastforce intro: terminates.intros)
next
  case (Catch c1 s c2)
  have  $\Gamma \vdash \text{mark-guards } f \ c1 \downarrow \text{Normal } s \text{ by fact}$ 
  moreover
  {
    fix t
    assume exec-mark:  $\Gamma \vdash \langle \text{mark-guards } f \ c1, \text{Normal } s \rangle \Rightarrow \text{Abrupt } t$ 
    have  $\Gamma \vdash \text{mark-guards } f \ c2 \downarrow \text{Normal } t$ 
    proof –
      from exec-mark-guards-to-exec [OF exec-mark] obtain t' where
        exec-c1:  $\Gamma \vdash \langle c1, \text{Normal } s \rangle \Rightarrow t'$  and
        t'-Fault-f:  $t' = \text{Fault } f \longrightarrow t' = \text{Abrupt } t$  and
        t'-Fault:  $\text{isFault } t' \longrightarrow \text{isFault } (\text{Abrupt } t)$  and
        t'-noFault:  $\neg \text{isFault } t' \longrightarrow t' = \text{Abrupt } t$ 
      by fastforce
    show ?thesis
    proof (cases isFault t')
      case True
        with t'-Fault have  $\text{isFault } (\text{Abrupt } t)$  by simp
        thus ?thesis by simp
      next
        case False
          with t'-noFault have  $t' = \text{Abrupt } t$  by simp
          with exec-c1 Catch.hyps
          show ?thesis
            by auto
    qed
  }
  ultimately show ?case
    by (auto intro: terminates.intros)
qed

```

**lemma** *terminates-mark-guards-to-terminates-Normal*:

$\bigwedge s. \Gamma \vdash \text{mark-guards } f \ c \downarrow \text{Normal } s \implies \Gamma \vdash c \downarrow \text{Normal } s$

**proof** (*induct c*)

case *Skip thus ?case by (fastforce intro: terminates.intros)*

```

next
  case Basic thus ?case by (fastforce intro: terminates.intros)
next
  case Spec thus ?case by (fastforce intro: terminates.intros)
next
  case (Seq c1 c2)
  have  $\Gamma \vdash \text{mark-guards } f \text{ (Seq } c1 \ c2) \downarrow \text{Normal } s$  by fact
  then obtain
    termi-merge-c1:  $\Gamma \vdash \text{mark-guards } f \ c1 \downarrow \text{Normal } s$  and
    termi-merge-c2:  $\forall s'. \Gamma \vdash \langle \text{mark-guards } f \ c1, \text{Normal } s \rangle \Rightarrow s' \longrightarrow$ 
       $\Gamma \vdash \text{mark-guards } f \ c2 \downarrow s'$ 
    by (auto elim: terminates-Normal-elim-cases)
  from termi-merge-c1 Seq.hyps
  have  $\Gamma \vdash c1 \downarrow \text{Normal } s$  by iprover
  moreover
  {
    fix s'
    assume exec-c1:  $\Gamma \vdash \langle c1, \text{Normal } s \rangle \Rightarrow s'$ 
    have  $\Gamma \vdash c2 \downarrow s'$ 
    proof (cases isFault s')
      case True
      thus ?thesis by (auto elim: isFaultE)
    next
      case False
      from exec-to-exec-mark-guards [OF exec-c1 False]
      have  $\Gamma \vdash \langle \text{mark-guards } f \ c1, \text{Normal } s \rangle \Rightarrow s'$  .
      from termi-merge-c2 [rule-format, OF this] Seq.hyps
      show ?thesis
      by (cases s') (auto)
    qed
  }
  ultimately show ?case by (auto intro: terminates.intros)
next
  case Cond thus ?case
  by (fastforce intro: terminates.intros elim: terminates-Normal-elim-cases)
next
  case (While b c)
  {
    fix u c'
    assume termi-c':  $\Gamma \vdash c' \downarrow \text{Normal } u$ 
    assume c':  $c' = \text{mark-guards } f \text{ (While } b \ c)$ 
    have  $\Gamma \vdash \text{While } b \ c \downarrow \text{Normal } u$ 
    using termi-c' c'
    proof (induct)
      case (WhileTrue s b' c')
      have s-in-b:  $s \in b$  using WhileTrue by simp
      have  $\Gamma \vdash \text{mark-guards } f \ c \downarrow \text{Normal } s$ 
      using WhileTrue by (auto elim: terminates-Normal-elim-cases)
      with While.hyps have  $\Gamma \vdash c \downarrow \text{Normal } s$ 
  }

```

```

    by auto
  moreover
  have hyp-w:  $\forall w. \Gamma \vdash \langle \text{mark-guards } f \ c, \text{Normal } s \rangle \Rightarrow w \longrightarrow \Gamma \vdash \text{While } b \ c \ \downarrow \ w$ 
    using WhileTrue by simp
  hence  $\forall w. \Gamma \vdash \langle c, \text{Normal } s \rangle \Rightarrow w \longrightarrow \Gamma \vdash \text{While } b \ c \ \downarrow \ w$ 
    apply -
    apply (rule allI)
    apply (case-tac w)
    apply (auto dest: exec-to-exec-mark-guards)
    done
  ultimately show ?case
    using s-in-b
    by (auto intro: terminates.intros)
next
  case WhileFalse thus ?case by (auto intro: terminates.intros)
qed auto
}
with While show ?case by simp
next
  case Call thus ?case
    by (fastforce intro: terminates.intros )
next
  case DynCom thus ?case
    by (fastforce intro: terminates.intros elim: terminates-Normal-elim-cases)
next
  case (Guard f g c)
  thus ?case by (fastforce intro: terminates.intros elim: terminates-Normal-elim-cases)
next
  case Throw thus ?case
    by (fastforce intro: terminates.intros )
next
  case (Catch c1 c2)
  have  $\Gamma \vdash \text{mark-guards } f \ (\text{Catch } c1 \ c2) \ \downarrow \ \text{Normal } s$  by fact
  then obtain
    termi-merge-c1:  $\Gamma \vdash \text{mark-guards } f \ c1 \ \downarrow \ \text{Normal } s$  and
    termi-merge-c2:  $\forall s'. \Gamma \vdash \langle \text{mark-guards } f \ c1, \text{Normal } s \rangle \Rightarrow \text{Abrupt } s' \longrightarrow$ 
       $\Gamma \vdash \text{mark-guards } f \ c2 \ \downarrow \ \text{Normal } s'$ 
  by (auto elim: terminates-Normal-elim-cases)
  from termi-merge-c1 Catch.hyps
  have  $\Gamma \vdash c1 \ \downarrow \ \text{Normal } s$  by iprover
  moreover
  {
    fix s'
    assume exec-c1:  $\Gamma \vdash \langle c1, \text{Normal } s \rangle \Rightarrow \text{Abrupt } s'$ 
    have  $\Gamma \vdash c2 \ \downarrow \ \text{Normal } s'$ 
    proof -
      from exec-to-exec-mark-guards [OF exec-c1]
      have  $\Gamma \vdash \langle \text{mark-guards } f \ c1, \text{Normal } s \rangle \Rightarrow \text{Abrupt } s'$  by simp
      from termi-merge-c2 [rule-format, OF this] Catch.hyps

```

```

      show ?thesis
      by iprover
    qed
  }
  ultimately show ?case by (auto intro: terminates.intros)
qed

```

**lemma** *terminates-mark-guards-to-terminates*:  
 $\Gamma \vdash \text{mark-guards } f \ c \downarrow s \implies \Gamma \vdash c \downarrow s$   
**by** (cases s) (auto intro: terminates-mark-guards-to-terminates-Normal)

## 7.6 Lemmas about *merge-guards*

**lemma** *terminates-to-terminates-merge-guards*:  
**assumes** *termi*:  $\Gamma \vdash c \downarrow s$   
**shows**  $\Gamma \vdash \text{merge-guards } c \downarrow s$   
**using** *termi*  
**proof** (*induct*)  
**case** (*Guard s g c f*)  
**have** *s-in-g*:  $s \in g$  **by** *fact*  
**have** *termi-merge-c*:  $\Gamma \vdash \text{merge-guards } c \downarrow \text{Normal } s$  **by** *fact*  
**show** ?case  
**proof** (cases  $\exists f' g' c'. \text{merge-guards } c = \text{Guard } f' g' c'$ )  
**case** *False*  
**hence**  $\text{merge-guards } (\text{Guard } f g c) = \text{Guard } f g (\text{merge-guards } c)$   
**by** (cases *merge-guards c*) (auto simp add: *Let-def*)  
**with** *s-in-g termi-merge-c* **show** ?thesis  
**by** (auto intro: *terminates.intros*)  
**next**  
**case** *True*  
**then obtain**  $f' g' c'$  **where**  
*mc*:  $\text{merge-guards } c = \text{Guard } f' g' c'$   
**by** *blast*  
**show** ?thesis  
**proof** (cases  $f=f'$ )  
**case** *False*  
**with** *mc* **have**  $\text{merge-guards } (\text{Guard } f g c) = \text{Guard } f g (\text{merge-guards } c)$   
**by** (*simp add: Let-def*)  
**with** *s-in-g termi-merge-c* **show** ?thesis  
**by** (auto intro: *terminates.intros*)  
**next**  
**case** *True*  
**with** *mc* **have**  $\text{merge-guards } (\text{Guard } f g c) = \text{Guard } f (g \cap g') c'$   
**by** *simp*  
**with** *s-in-g mc True termi-merge-c*  
**show** ?thesis  
**by** (cases  $s \in g'$ )  
(auto intro: *terminates.intros elim: terminates-Normal-elim-cases*)  
**qed**

```

qed
next
  case (GuardFault s g f c)
  have  $s \notin g$  by fact
  thus ?case
    by (cases merge-guards c)
      (auto intro: terminates.intros split: if-split-asm simp add: Let-def)
qed (fastforce intro: terminates.intros dest: exec-merge-guards-to-exec)+

lemma terminates-merge-guards-to-terminates-Normal:
  shows  $\bigwedge s. \Gamma \vdash \text{merge-guards } c \downarrow \text{Normal } s \implies \Gamma \vdash c \downarrow \text{Normal } s$ 
proof (induct c)
  case Skip thus ?case by (fastforce intro: terminates.intros)
next
  case Basic thus ?case by (fastforce intro: terminates.intros)
next
  case Spec thus ?case by (fastforce intro: terminates.intros)
next
  case (Seq c1 c2)
  have  $\Gamma \vdash \text{merge-guards } (Seq\ c1\ c2) \downarrow \text{Normal } s$  by fact
  then obtain
    termi-merge-c1:  $\Gamma \vdash \text{merge-guards } c1 \downarrow \text{Normal } s$  and
    termi-merge-c2:  $\forall s'. \Gamma \vdash \langle \text{merge-guards } c1, \text{Normal } s \rangle \Rightarrow s' \longrightarrow$ 
       $\Gamma \vdash \text{merge-guards } c2 \downarrow s'$ 
  by (auto elim: terminates-Normal-elim-cases)
from termi-merge-c1 Seq.hyps
have  $\Gamma \vdash c1 \downarrow \text{Normal } s$  by iprover
moreover
  {
    fix s'
    assume exec-c1:  $\Gamma \vdash \langle c1, \text{Normal } s \rangle \Rightarrow s'$ 
    have  $\Gamma \vdash c2 \downarrow s'$ 
    proof –
      from exec-to-exec-merge-guards [OF exec-c1]
      have  $\Gamma \vdash \langle \text{merge-guards } c1, \text{Normal } s \rangle \Rightarrow s'$ .
      from termi-merge-c2 [rule-format, OF this] Seq.hyps
      show ?thesis
      by (cases s') (auto)
    qed
  }
ultimately show ?case by (auto intro: terminates.intros)
next
  case Cond thus ?case
    by (fastforce intro: terminates.intros elim: terminates-Normal-elim-cases)
next
  case (While b c)
  {
    fix u c'
    assume termi-c':  $\Gamma \vdash c' \downarrow \text{Normal } u$ 

```

```

assume  $c'$ :  $c' = \text{merge-guards } (\text{While } b \ c)$ 
have  $\Gamma \vdash \text{While } b \ c \downarrow \text{Normal } u$ 
  using termi-c' c'
proof (induct)
  case (WhileTrue s b' c')
  have s-in-b:  $s \in b$  using WhileTrue by simp
  have  $\Gamma \vdash \text{merge-guards } c \downarrow \text{Normal } s$ 
    using WhileTrue by (auto elim: terminates-Normal-elim-cases)
  with While.hyps have  $\Gamma \vdash c \downarrow \text{Normal } s$ 
    by auto
  moreover
  have hyp-w:  $\forall w. \Gamma \langle \text{merge-guards } c, \text{Normal } s \rangle \Rightarrow w \longrightarrow \Gamma \vdash \text{While } b \ c \downarrow w$ 
    using WhileTrue by simp
  hence  $\forall w. \Gamma \langle c, \text{Normal } s \rangle \Rightarrow w \longrightarrow \Gamma \vdash \text{While } b \ c \downarrow w$ 
    by (simp add: exec-iff-exec-merge-guards [symmetric])
  ultimately show ?case
    using s-in-b
    by (auto intro: terminates.intros)
  next
  case WhileFalse thus ?case by (auto intro: terminates.intros)
  qed auto
}
with While show ?case by simp
next
case Call thus ?case
  by (fastforce intro: terminates.intros)
next
case DynCom thus ?case
  by (fastforce intro: terminates.intros elim: terminates-Normal-elim-cases)
next
case (Guard f g c)
have termi-merge:  $\Gamma \vdash \text{merge-guards } (\text{Guard } f \ g \ c) \downarrow \text{Normal } s$  by fact
show ?case
proof (cases  $\exists f' \ g' \ c'. \text{merge-guards } c = \text{Guard } f' \ g' \ c'$ )
  case False
  hence m:  $\text{merge-guards } (\text{Guard } f \ g \ c) = \text{Guard } f \ g \ (\text{merge-guards } c)$ 
    by (cases merge-guards c) (auto simp add: Let-def)
  from termi-merge Guard.hyps show ?thesis
    by (simp only: m)
    (fastforce intro: terminates.intros elim: terminates-Normal-elim-cases)
  next
  case True
  then obtain  $f' \ g' \ c'$  where
     $mc$ :  $\text{merge-guards } c = \text{Guard } f' \ g' \ c'$ 
    by blast
  show ?thesis
  proof (cases f=f')
    case False
    with mc have  $m$ :  $\text{merge-guards } (\text{Guard } f \ g \ c) = \text{Guard } f \ g \ (\text{merge-guards } c)$ 

```

```

    by (simp add: Let-def)
  from termi-merge Guard.hyps show ?thesis
  by (simp only: m)
    (fastforce intro: terminates.intros elim: terminates-Normal-elim-cases)
next
  case True
  with mc have m: merge-guards (Guard f g c) = Guard f (g  $\cap$  g') c'
    by simp
  from termi-merge Guard.hyps
  show ?thesis
    by (simp only: m mc)
      (auto intro: terminates.intros elim: terminates-Normal-elim-cases)
qed
qed
next
  case Throw thus ?case
    by (fastforce intro: terminates.intros )
next
  case (Catch c1 c2)
  have  $\Gamma \vdash$  merge-guards (Catch c1 c2)  $\downarrow$  Normal s by fact
  then obtain
    termi-merge-c1:  $\Gamma \vdash$  merge-guards c1  $\downarrow$  Normal s and
    termi-merge-c2:  $\forall s'. \Gamma \vdash$  (merge-guards c1, Normal s)  $\Rightarrow$  Abrupt s'  $\longrightarrow$ 
       $\Gamma \vdash$  merge-guards c2  $\downarrow$  Normal s'
    by (auto elim: terminates-Normal-elim-cases)
  from termi-merge-c1 Catch.hyps
  have  $\Gamma \vdash$  c1  $\downarrow$  Normal s by iprover
  moreover
  {
    fix s'
    assume exec-c1:  $\Gamma \vdash$  (c1, Normal s)  $\Rightarrow$  Abrupt s'
    have  $\Gamma \vdash$  c2  $\downarrow$  Normal s'
    proof -
      from exec-to-exec-merge-guards [OF exec-c1]
      have  $\Gamma \vdash$  (merge-guards c1, Normal s)  $\Rightarrow$  Abrupt s' .
      from termi-merge-c2 [rule-format, OF this] Catch.hyps
      show ?thesis
        by iprover
    qed
  }
  ultimately show ?case by (auto intro: terminates.intros)
qed

```

**lemma** *terminates-merge-guards-to-terminates:*

$\Gamma \vdash$  merge-guards  $c \downarrow s \implies \Gamma \vdash c \downarrow s$

**by** (cases s) (auto intro: terminates-merge-guards-to-terminates-Normal)

**theorem** *terminates-iff-terminates-merge-guards:*

$\Gamma \vdash c \downarrow s = \Gamma \vdash$  merge-guards  $c \downarrow s$

by (*iprover intro: terminates-to-terminates-merge-guards*  
*terminates-merge-guards-to-terminates*)

## 7.7 Lemmas about $c_1 \subseteq_g c_2$

**lemma** *terminates-fewer-guards-Normal*:

**shows**  $\bigwedge c s. [\Gamma \vdash c \downarrow \text{Normal } s; c \subseteq_g c'; \Gamma \vdash \langle c', \text{Normal } s \rangle \Rightarrow \notin \text{Fault ' UNIV}]$   
 $\implies \Gamma \vdash c \downarrow \text{Normal } s$

**proof** (*induct c'*)

**case** *Skip* **thus** ?*case* **by** (*auto intro: terminates.intros dest: subseteq-guardsD*)

**next**

**case** *Basic* **thus** ?*case* **by** (*auto intro: terminates.intros dest: subseteq-guardsD*)

**next**

**case** *Spec* **thus** ?*case* **by** (*auto intro: terminates.intros dest: subseteq-guardsD*)

**next**

**case** (*Seq c1' c2'*)

**have** *termi*:  $\Gamma \vdash \text{Seq } c1' c2' \downarrow \text{Normal } s$  **by** *fact*

**then obtain**

*termi-c1'*:  $\Gamma \vdash c1' \downarrow \text{Normal } s$  **and**

*termi-c2'*:  $\forall s'. \Gamma \vdash \langle c1', \text{Normal } s \rangle \Rightarrow s' \longrightarrow \Gamma \vdash c2' \downarrow s'$

**by** (*auto elim: terminates-Normal-elim-cases*)

**have** *noFault*:  $\Gamma \vdash \langle \text{Seq } c1' c2', \text{Normal } s \rangle \Rightarrow \notin \text{Fault ' UNIV}$  **by** *fact*

**hence** *noFault-c1'*:  $\Gamma \vdash \langle c1', \text{Normal } s \rangle \Rightarrow \notin \text{Fault ' UNIV}$

**by** (*auto intro: exec.intros simp add: final-notin-def*)

**have**  $c \subseteq_g \text{Seq } c1' c2'$  **by** *fact*

**from** *subseteq-guards-Seq* [*OF this*] **obtain** *c1 c2* **where**

*c*:  $c = \text{Seq } c1 c2$  **and**

*c1-c1'*:  $c1 \subseteq_g c1'$  **and**

*c2-c2'*:  $c2 \subseteq_g c2'$

**by** *blast*

**from** *termi-c1' c1-c1' noFault-c1'*

**have**  $\Gamma \vdash c1 \downarrow \text{Normal } s$

**by** (*rule Seq.hyps*)

**moreover**

{

**fix** *t*

**assume** *exec-c1*:  $\Gamma \vdash \langle c1, \text{Normal } s \rangle \Rightarrow t$

**have**  $\Gamma \vdash c2 \downarrow t$

**proof** –

**from** *exec-to-exec-subseteq-guards* [*OF c1-c1' exec-c1*] **obtain** *t'* **where**

*exec-c1'*:  $\Gamma \vdash \langle c1', \text{Normal } s \rangle \Rightarrow t'$  **and**

*t-Fault*:  $\text{isFault } t \longrightarrow \text{isFault } t'$  **and**

*t'-noFault*:  $\neg \text{isFault } t' \longrightarrow t' = t$

**by** *blast*

**show** ?*thesis*

**proof** (*cases isFault t'*)

**case** *True*

**with** *exec-c1' noFault-c1'*

**have** *False*



```

    by (fastforce elim: isFaultE dest: Fault-end simp add: final-notin-def)
  thus ?thesis ..
next
case False
with t'-noFault have t': t'=t by simp
with termi-c2' exec-c1'
have termi-c2':  $\Gamma \vdash c2' \downarrow t$ 
  by auto
show ?thesis
proof (cases t)
  case Fault thus ?thesis by auto
next
  case Abrupt thus ?thesis by auto
next
  case Stuck thus ?thesis by auto
next
  case (Normal u)
  with noFault exec-c1' t'
  have  $\Gamma \vdash \langle c2', Normal\ u \rangle \Rightarrow \notin Fault \text{ ' UNIV}$ 
    by (auto intro: exec.intros simp add: final-notin-def)
  from termi-c2' [simplified Normal] c2-c2' this
  have  $\Gamma \vdash c2 \downarrow Normal\ u$ 
    by (rule Seq.hyps)
  with Normal exec-c1
  show ?thesis by simp
qed
qed
qed
}
ultimately show ?case using c by (auto intro: terminates.intros)
next
case (Cond b c1' c2')
have noFault:  $\Gamma \vdash \langle Cond\ b\ c1'\ c2', Normal\ s \rangle \Rightarrow \notin Fault \text{ ' UNIV}$  by fact
have termi:  $\Gamma \vdash Cond\ b\ c1'\ c2' \downarrow Normal\ s$  by fact
have  $c \subseteq_g Cond\ b\ c1'\ c2'$  by fact
from subseteq-guards-Cond [OF this] obtain c1 c2 where
  c:  $c = Cond\ b\ c1\ c2$  and
  c1-c1':  $c1 \subseteq_g c1'$  and
  c2-c2':  $c2 \subseteq_g c2'$ 
  by blast
thus ?case
proof (cases s  $\in$  b)
  case True
  with termi have termi-c1':  $\Gamma \vdash c1' \downarrow Normal\ s$ 
    by (auto elim: terminates-Normal-elim-cases)
  from True noFault have  $\Gamma \vdash \langle c1', Normal\ s \rangle \Rightarrow \notin Fault \text{ ' UNIV}$ 
    by (auto intro: exec.intros simp add: final-notin-def)
  from termi-c1' c1-c1' this
  have  $\Gamma \vdash c1 \downarrow Normal\ s$ 

```

```

    by (rule Cond.hyps)
  with True c show ?thesis
    by (auto intro: terminates.intros)
next
case False
with termi have termi-c2':  $\Gamma \vdash c2' \downarrow Normal\ s$ 
  by (auto elim: terminates-Normal-elim-cases)
from False noFault have  $\Gamma \vdash \langle c2', Normal\ s \rangle \Rightarrow \notin Fault\ ' UNIV$ 
  by (auto intro: exec.intros simp add: final-notin-def)
from termi-c2' c2-c2' this
have  $\Gamma \vdash c2 \downarrow Normal\ s$ 
  by (rule Cond.hyps)
with False c show ?thesis
  by (auto intro: terminates.intros)
qed
next
case (While b c')
have noFault:  $\Gamma \vdash \langle While\ b\ c', Normal\ s \rangle \Rightarrow \notin Fault\ ' UNIV$  by fact
have termi:  $\Gamma \vdash While\ b\ c' \downarrow Normal\ s$  by fact
have  $c \subseteq_g While\ b\ c'$  by fact
from subseteq-guards-While [OF this]
obtain c'' where
  c:  $c = While\ b\ c''$  and
  c''-c':  $c'' \subseteq_g c'$ 
  by blast
{
  fix d u
  assume termi:  $\Gamma \vdash d \downarrow u$ 
  assume d:  $d = While\ b\ c'$ 
  assume noFault:  $\Gamma \vdash \langle While\ b\ c', u \rangle \Rightarrow \notin Fault\ ' UNIV$ 
  have  $\Gamma \vdash While\ b\ c'' \downarrow u$ 
  using termi d noFault
  proof (induct)
    case (WhileTrue u b' c'')
    have u-in-b:  $u \in b$  using WhileTrue by simp
    have termi-c':  $\Gamma \vdash c' \downarrow Normal\ u$  using WhileTrue by simp
    have noFault:  $\Gamma \vdash \langle While\ b\ c', Normal\ u \rangle \Rightarrow \notin Fault\ ' UNIV$  using WhileTrue
  by simp
  hence noFault-c':  $\Gamma \vdash \langle c', Normal\ u \rangle \Rightarrow \notin Fault\ ' UNIV$  using u-in-b
    by (auto intro: exec.intros simp add: final-notin-def)
  from While.hyps [OF termi-c' c''-c' this]
  have  $\Gamma \vdash c'' \downarrow Normal\ u.$ 
  moreover
  from WhileTrue
  have hyp-w:  $\forall s'. \Gamma \vdash \langle c', Normal\ u \rangle \Rightarrow s' \longrightarrow \Gamma \vdash \langle While\ b\ c', s' \rangle \Rightarrow \notin Fault\ ' UNIV$ 
     $\longrightarrow \Gamma \vdash While\ b\ c'' \downarrow s'$ 
    by simp
}

```

```

fix v
assume  $exec\text{-}c'': \Gamma \langle c'', Normal\ u \rangle \Rightarrow v$ 
have  $\Gamma \vdash While\ b\ c'' \downarrow v$ 
proof –
  from  $exec\text{-}to\text{-}exec\text{-}subsetq\text{-}guards\ [OF\ c''\text{-}c'\ exec\text{-}c'']$  obtain  $v'$  where
     $exec\text{-}c': \Gamma \langle c', Normal\ u \rangle \Rightarrow v'$  and
     $v\text{-}Fault: isFault\ v \longrightarrow isFault\ v'$  and
     $v'\text{-}noFault: \neg isFault\ v' \longrightarrow v' = v$ 
    by auto
  show ?thesis
  proof (cases isFault v')
    case True
      with  $exec\text{-}c'\ noFault\ u\text{-}in\text{-}b$ 
      have False
      by (fastforce
        simp add: final-notin-def intro: exec.intros elim: isFaultE)
      thus ?thesis ..
    next
      case False
      with  $v'\text{-}noFault$  have  $v': v'=v$ 
      by simp
      with  $noFault\ exec\text{-}c'\ u\text{-}in\text{-}b$ 
      have  $\Gamma \vdash \langle While\ b\ c', v \rangle \Rightarrow \notin Fault\ 'UNIV$ 
      by (fastforce simp add: final-notin-def intro: exec.intros)
      from hyp-w [rule-format, OF exec-c' [simplified v'] this]
      show  $\Gamma \vdash While\ b\ c'' \downarrow v .$ 
      qed
    qed
  }
  ultimately
  show ?case using u-in-b
  by (auto intro: terminates.intros)
  next
  case WhileFalse thus ?case by (auto intro: terminates.intros)
  qed auto
}
with  $c\ noFault\ termi$  show ?case
by auto
next
case Call thus ?case by (auto intro: terminates.intros dest: subsetq-guardsD)
next
case (DynCom C')
have  $termi: \Gamma \vdash DynCom\ C' \downarrow Normal\ s$  by fact
hence  $termi\text{-}C': \Gamma \vdash C' s \downarrow Normal\ s$ 
by cases
have  $noFault: \Gamma \langle DynCom\ C', Normal\ s \rangle \Rightarrow \notin Fault\ 'UNIV$  by fact
hence  $noFault\text{-}C': \Gamma \langle C' s, Normal\ s \rangle \Rightarrow \notin Fault\ 'UNIV$ 
by (auto intro: exec.intros simp add: final-notin-def)
have  $c \subseteq_g DynCom\ C'$  by fact

```

```

from subseteq-guards-DynCom [OF this] obtain  $C$  where
   $c = \text{DynCom } C$  and
   $C-C': \forall s. C \ s \subseteq_g \ C' \ s$ 
  by blast
from DynCom.hyps termi-C' C-C' [rule-format] noFault-C'
have  $\Gamma \vdash C \ s \downarrow \text{Normal } s$ 
  by fast
with  $c$  show ?case
  by (auto intro: terminates.intros)
next
case (Guard f' g' c')
have noFault:  $\Gamma \vdash \langle \text{Guard } f' \ g' \ c', \text{Normal } s \rangle \Rightarrow \notin \text{Fault ' UNIV}$  by fact
have termi:  $\Gamma \vdash \text{Guard } f' \ g' \ c' \downarrow \text{Normal } s$  by fact
have  $c \subseteq_g \ \text{Guard } f' \ g' \ c'$  by fact
hence c-cases:  $(c \subseteq_g \ c') \vee (\exists c''. c = \text{Guard } f' \ g' \ c'' \wedge (c'' \subseteq_g \ c'))$ 
  by (rule subseteq-guards-Guard)
thus ?case
proof (cases s ∈ g')
  case True
  note s-in-g' = this
  with noFault have noFault-c':  $\Gamma \vdash \langle c', \text{Normal } s \rangle \Rightarrow \notin \text{Fault ' UNIV}$ 
    by (auto simp add: final-notin-def intro: exec.intros)
  from termi s-in-g' have termi-c':  $\Gamma \vdash c' \downarrow \text{Normal } s$ 
    by cases auto
  from c-cases show ?thesis
proof
  assume  $c \subseteq_g \ c'$ 
  from termi-c' this noFault-c'
  show  $\Gamma \vdash c \downarrow \text{Normal } s$ 
    by (rule Guard.hyps)
next
  assume  $\exists c''. c = \text{Guard } f' \ g' \ c'' \wedge (c'' \subseteq_g \ c')$ 
  then obtain  $c''$  where
     $c: c = \text{Guard } f' \ g' \ c''$  and  $c''-c': c'' \subseteq_g \ c'$ 
    by blast
  from termi-c' c''-c' noFault-c'
  have  $\Gamma \vdash c'' \downarrow \text{Normal } s$ 
    by (rule Guard.hyps)
  with s-in-g' c
  show ?thesis
    by (auto intro: terminates.intros)
qed
next
case False
with noFault have False
  by (auto intro: exec.intros simp add: final-notin-def)
thus ?thesis ..
qed
next

```

```

case Throw thus ?case by (auto intro: terminates.intros dest: subseteq-guardsD)
next
case (Catch c1' c2')
have termi:  $\Gamma \vdash \text{Catch } c1' \ c2' \downarrow \text{Normal } s$  by fact
then obtain
  termi-c1':  $\Gamma \vdash c1' \downarrow \text{Normal } s$  and
  termi-c2':  $\forall s'. \Gamma \vdash \langle c1', \text{Normal } s \rangle \Rightarrow \text{Abrupt } s' \longrightarrow \Gamma \vdash c2' \downarrow \text{Normal } s'$ 
  by (auto elim: terminates-Normal-elim-cases)
have noFault:  $\Gamma \vdash \langle \text{Catch } c1' \ c2', \text{Normal } s \rangle \Rightarrow \notin \text{Fault ' UNIV}$  by fact
hence noFault-c1':  $\Gamma \vdash \langle c1', \text{Normal } s \rangle \Rightarrow \notin \text{Fault ' UNIV}$ 
  by (fastforce intro: exec.intros simp add: final-notin-def)
have  $c \subseteq_g \text{Catch } c1' \ c2'$  by fact
from subseteq-guards-Catch [OF this] obtain  $c1 \ c2$  where
   $c: c = \text{Catch } c1 \ c2$  and
   $c1-c1'$ :  $c1 \subseteq_g c1'$  and
   $c2-c2'$ :  $c2 \subseteq_g c2'$ 
  by blast
from termi-c1' c1-c1' noFault-c1'
have  $\Gamma \vdash c1 \downarrow \text{Normal } s$ 
  by (rule Catch.hyps)
moreover
{
  fix  $t$ 
  assume exec-c1:  $\Gamma \vdash \langle c1, \text{Normal } s \rangle \Rightarrow \text{Abrupt } t$ 
  have  $\Gamma \vdash c2 \downarrow \text{Normal } t$ 
  proof –
    from exec-to-exec-subseteq-guards [OF c1-c1' exec-c1] obtain  $t'$  where
      exec-c1':  $\Gamma \vdash \langle c1', \text{Normal } s \rangle \Rightarrow t'$  and
      t'-noFault:  $\neg \text{isFault } t' \longrightarrow t' = \text{Abrupt } t$ 
      by blast
    show ?thesis
    proof (cases isFault t')
      case True
      with exec-c1' noFault-c1'
      have False
      by (fastforce elim: isFaultE dest: Fault-end simp add: final-notin-def)
      thus ?thesis ..
    next
    case False
    with t'-noFault have  $t': t' = \text{Abrupt } t$  by simp
    with termi-c2' exec-c1'
    have termi-c2':  $\Gamma \vdash c2' \downarrow \text{Normal } t$ 
      by auto
    with noFault exec-c1' t'
    have  $\Gamma \vdash \langle c2', \text{Normal } t \rangle \Rightarrow \notin \text{Fault ' UNIV}$ 
      by (auto intro: exec.intros simp add: final-notin-def)
    from termi-c2' c2-c2' this
    show  $\Gamma \vdash c2 \downarrow \text{Normal } t$ 
      by (rule Catch.hyps)
  }

```

```

      qed
    qed
  }
  ultimately show ?case using c by (auto intro: terminates.intros)
qed

theorem terminates-fewer-guards:
  shows  $\llbracket \Gamma \vdash c' \downarrow s; c \subseteq_g c'; \Gamma \vdash \langle c', s \rangle \Rightarrow \notin \text{Fault} \text{ ' UNIV} \rrbracket$ 
     $\implies \Gamma \vdash c \downarrow s$ 
  by (cases s) (auto intro: terminates-fewer-guards-Normal)

lemma terminates-noFault-strip-guards:
  assumes termi:  $\Gamma \vdash c \downarrow \text{Normal } s$ 
  shows  $\llbracket \Gamma \vdash \langle c, \text{Normal } s \rangle \Rightarrow \notin \text{Fault} \text{ ' F} \rrbracket \implies \Gamma \vdash \text{strip-guards } F \ c \downarrow \text{Normal } s$ 
using termi
proof (induct)
  case Skip thus ?case by (auto intro: terminates.intros)
next
  case Basic thus ?case by (auto intro: terminates.intros)
next
  case Spec thus ?case by (auto intro: terminates.intros)
next
  case (Guard s g c f)
  have s-in-g:  $s \in g$  by fact
  have  $\Gamma \vdash c \downarrow \text{Normal } s$  by fact
  have  $\Gamma \vdash \langle \text{Guard } f \ g \ c, \text{Normal } s \rangle \Rightarrow \notin \text{Fault} \text{ ' F}$  by fact
  with s-in-g have  $\Gamma \vdash \langle c, \text{Normal } s \rangle \Rightarrow \notin \text{Fault} \text{ ' F}$ 
    by (fastforce simp add: final-notin-def intro: exec.intros)
  with Guard.hyps have  $\Gamma \vdash \text{strip-guards } F \ c \downarrow \text{Normal } s$  by simp
  with s-in-g show ?case
    by (auto intro: terminates.intros)
next
  case GuardFault thus ?case
    by (auto intro: terminates.intros exec.intros simp add: final-notin-def )
next
  case Fault thus ?case by (auto intro: terminates.intros)
next
  case (Seq c1 s c2)
  have noFault-Seq:  $\Gamma \vdash \langle \text{Seq } c1 \ c2, \text{Normal } s \rangle \Rightarrow \notin \text{Fault} \text{ ' F}$  by fact
  hence noFault-c1:  $\Gamma \vdash \langle c1, \text{Normal } s \rangle \Rightarrow \notin \text{Fault} \text{ ' F}$ 
    by (auto simp add: final-notin-def intro: exec.intros)
  with Seq.hyps have  $\Gamma \vdash \text{strip-guards } F \ c1 \downarrow \text{Normal } s$  by simp
  moreover
  {
    fix s'
    assume exec-strip-guards-c1:  $\Gamma \vdash \langle \text{strip-guards } F \ c1, \text{Normal } s \rangle \Rightarrow s'$ 
    have  $\Gamma \vdash \text{strip-guards } F \ c2 \downarrow s'$ 
    proof (cases isFault s')
      case True

```

```

    thus ?thesis by (auto elim: isFaultE intro: terminates.intros)
  next
  case False
  with exec-strip-guards-to-exec [OF exec-strip-guards-c1] noFault-c1
  have *:  $\Gamma \vdash \langle c1, Normal\ s \rangle \Rightarrow s'$ 
    by (auto simp add: final-notin-def elim!: isFaultE)
  with noFault-Seq have  $\Gamma \vdash \langle c2, s' \rangle \Rightarrow \notin Fault\ 'F$ 
    by (auto simp add: final-notin-def intro: exec.intros)
  with * show ?thesis
    using Seq.hyps by simp
  qed
}
ultimately show ?case
  by (auto intro: terminates.intros)
next
case CondTrue thus ?case
  by (fastforce intro: terminates.intros exec.intros simp add: final-notin-def )
next
case CondFalse thus ?case
  by (fastforce intro: terminates.intros exec.intros simp add: final-notin-def )
next
case (WhileTrue s b c)
have s-in-b:  $s \in b$  by fact
have noFault-while:  $\Gamma \vdash \langle While\ b\ c, Normal\ s \rangle \Rightarrow \notin Fault\ 'F$  by fact
with s-in-b have noFault-c:  $\Gamma \vdash \langle c, Normal\ s \rangle \Rightarrow \notin Fault\ 'F$ 
  by (auto simp add: final-notin-def intro: exec.intros)
with WhileTrue.hyps have  $\Gamma \vdash strip\ guards\ F\ c \downarrow Normal\ s$  by simp
moreover
{
  fix s'
  assume exec-strip-guards-c:  $\Gamma \vdash \langle strip\ guards\ F\ c, Normal\ s \rangle \Rightarrow s'$ 
  have  $\Gamma \vdash strip\ guards\ F\ (While\ b\ c) \downarrow s'$ 
  proof (cases isFault s')
    case True
    thus ?thesis by (auto elim: isFaultE intro: terminates.intros)
  next
  case False
  with exec-strip-guards-to-exec [OF exec-strip-guards-c] noFault-c
  have *:  $\Gamma \vdash \langle c, Normal\ s \rangle \Rightarrow s'$ 
    by (auto simp add: final-notin-def elim!: isFaultE)
  with s-in-b noFault-while have  $\Gamma \vdash \langle While\ b\ c, s' \rangle \Rightarrow \notin Fault\ 'F$ 
    by (auto simp add: final-notin-def intro: exec.intros)
  with * show ?thesis
    using WhileTrue.hyps by simp
  qed
}
ultimately show ?case
  using WhileTrue.hyps by (auto intro: terminates.intros)
next

```

```

  case WhileFalse thus ?case by (auto intro: terminates.intros)
next
  case Call thus ?case by (auto intro: terminates.intros)
next
  case CallUndefined thus ?case by (auto intro: terminates.intros)
next
  case Stuck thus ?case by (auto intro: terminates.intros)
next
  case DynCom thus ?case
    by (auto intro: terminates.intros exec.intros simp add: final-notin-def )
next
  case Throw thus ?case by (auto intro: terminates.intros)
next
  case Abrupt thus ?case by (auto intro: terminates.intros)
next
  case ( Catch c1 s c2 )
  have noFault-Catch:  $\Gamma \vdash \langle \text{Catch } c1 \ c2, \text{Normal } s \rangle \Rightarrow \notin \text{Fault } ' F$  by fact
  hence noFault-c1:  $\Gamma \vdash \langle c1, \text{Normal } s \rangle \Rightarrow \notin \text{Fault } ' F$ 
    by (fastforce simp add: final-notin-def intro: exec.intros)
  with Catch.hyps have  $\Gamma \vdash \text{strip-guards } F \ c1 \downarrow \text{Normal } s$  by simp
  moreover
  {
    fix s'
    assume exec-strip-guards-c1:  $\Gamma \vdash \langle \text{strip-guards } F \ c1, \text{Normal } s \rangle \Rightarrow \text{Abrupt } s'$ 
    have  $\Gamma \vdash \text{strip-guards } F \ c2 \downarrow \text{Normal } s'$ 
    proof -
      from exec-strip-guards-to-exec [OF exec-strip-guards-c1] noFault-c1
      have *:  $\Gamma \vdash \langle c1, \text{Normal } s \rangle \Rightarrow \text{Abrupt } s'$ 
        by (auto simp add: final-notin-def elim!: isFaultE)
      with noFault-Catch have  $\Gamma \vdash \langle c2, \text{Normal } s' \rangle \Rightarrow \notin \text{Fault } ' F$ 
        by (auto simp add: final-notin-def intro: exec.intros)
      with * show ?thesis
        using Catch.hyps by simp
    qed
  }
  ultimately show ?case
    using Catch.hyps by (auto intro: terminates.intros)
qed

```

## 7.8 Lemmas about *strip-guards*

lemma *terminates-noFault-strip*:

```

  assumes termi:  $\Gamma \vdash c \downarrow \text{Normal } s$ 
  shows  $\llbracket \Gamma \vdash \langle c, \text{Normal } s \rangle \Rightarrow \notin \text{Fault } ' F \rrbracket \implies \text{strip } F \ \Gamma \vdash c \downarrow \text{Normal } s$ 
using termi
proof (induct)
  case Skip thus ?case by (auto intro: terminates.intros)
next
  case Basic thus ?case by (auto intro: terminates.intros)

```



```

next
  case Spec thus ?case by (auto intro: terminates.intros)
next
  case (Guard s g c f)
  have s-in-g:  $s \in g$  by fact
  have  $\Gamma \langle \text{Guard } f \ g \ c, \text{Normal } s \rangle \Rightarrow \notin \text{Fault } ' F$  by fact
  with s-in-g have  $\Gamma \langle c, \text{Normal } s \rangle \Rightarrow \notin \text{Fault } ' F$ 
    by (fastforce simp add: final-notin-def intro: exec.intros)
  then have strip F  $\Gamma \vdash c \downarrow \text{Normal } s$  by (simp add: Guard.hyps)
  with s-in-g show ?case
    by (auto intro: terminates.intros simp del: strip-simp)
next
  case GuardFault thus ?case
    by (auto intro: terminates.intros exec.intros simp add: final-notin-def )
next
  case Fault thus ?case by (auto intro: terminates.intros)
next
  case (Seq c1 s c2)
  have noFault-Seq:  $\Gamma \langle \text{Seq } c1 \ c2, \text{Normal } s \rangle \Rightarrow \notin \text{Fault } ' F$  by fact
  hence noFault-c1:  $\Gamma \langle c1, \text{Normal } s \rangle \Rightarrow \notin \text{Fault } ' F$ 
    by (auto simp add: final-notin-def intro: exec.intros)
  then have strip F  $\Gamma \vdash c1 \downarrow \text{Normal } s$  by (simp add: Seq.hyps)
  moreover
  {
    fix s'
    assume exec-strip-c1: strip F  $\Gamma \langle c1, \text{Normal } s \rangle \Rightarrow s'$ 
    have strip F  $\Gamma \vdash c2 \downarrow s'$ 
    proof (cases isFault s')
      case True
      thus ?thesis by (auto elim: isFaultE intro: terminates.intros)
    next
      case False
      with exec-strip-to-exec [OF exec-strip-c1] noFault-c1
      have *:  $\Gamma \langle c1, \text{Normal } s \rangle \Rightarrow s'$ 
        by (auto simp add: final-notin-def elim!: isFaultE)
      with noFault-Seq have  $\Gamma \langle c2, s' \rangle \Rightarrow \notin \text{Fault } ' F$ 
        by (auto simp add: final-notin-def intro: exec.intros)
      with * show ?thesis
        using Seq.hyps by (simp del: strip-simp)
    qed
  }
  ultimately show ?case
    by (fastforce intro: terminates.intros)
next
  case CondTrue thus ?case
    by (fastforce intro: terminates.intros exec.intros simp add: final-notin-def )
next
  case CondFalse thus ?case
    by (fastforce intro: terminates.intros exec.intros simp add: final-notin-def )

```

```

next
  case (WhileTrue s b c)
  have s-in-b: s ∈ b by fact
  have noFault-while:  $\Gamma \vdash \langle \text{While } b \ c, \text{Normal } s \rangle \Rightarrow \notin \text{Fault } ' F$  by fact
  with s-in-b have noFault-c:  $\Gamma \vdash \langle c, \text{Normal } s \rangle \Rightarrow \notin \text{Fault } ' F$ 
    by (auto simp add: final-notin-def intro: exec.intros)
  then have strip F  $\Gamma \vdash c \downarrow \text{Normal } s$  by (simp add: WhileTrue.hyps)
  moreover
  {
    fix s'
    assume exec-strip-c: strip F  $\Gamma \vdash \langle c, \text{Normal } s \rangle \Rightarrow s'$ 
    have strip F  $\Gamma \vdash \text{While } b \ c \downarrow s'$ 
    proof (cases isFault s')
      case True
      thus ?thesis by (auto elim: isFaultE intro: terminates.intros)
    next
      case False
      with exec-strip-to-exec [OF exec-strip-c] noFault-c
      have *:  $\Gamma \vdash \langle c, \text{Normal } s \rangle \Rightarrow s'$ 
        by (auto simp add: final-notin-def elim!: isFaultE)
      with s-in-b noFault-while have  $\Gamma \vdash \langle \text{While } b \ c, s' \rangle \Rightarrow \notin \text{Fault } ' F$ 
        by (auto simp add: final-notin-def intro: exec.intros)
      with * show ?thesis
        using WhileTrue.hyps by (simp del: strip-simp)
    qed
  }
  ultimately show ?case
    using WhileTrue.hyps by (auto intro: terminates.intros simp del: strip-simp)
next
  case WhileFalse thus ?case by (auto intro: terminates.intros)
next
  case (Call p bdy s)
  have bdy:  $\Gamma \ p = \text{Some } \text{bdy}$  by fact
  have  $\Gamma \vdash \langle \text{Call } p, \text{Normal } s \rangle \Rightarrow \notin \text{Fault } ' F$  by fact
  with bdy have bdy-noFault:  $\Gamma \vdash \langle \text{bdy}, \text{Normal } s \rangle \Rightarrow \notin \text{Fault } ' F$ 
    by (auto intro: exec.intros simp add: final-notin-def)
  then have strip-bdy-noFault: strip F  $\Gamma \vdash \langle \text{bdy}, \text{Normal } s \rangle \Rightarrow \notin \text{Fault } ' F$ 
    by (auto simp add: final-notin-def dest!: exec-strip-to-exec elim!: isFaultE)

  from bdy-noFault have strip F  $\Gamma \vdash \text{bdy} \downarrow \text{Normal } s$  by (simp add: Call.hyps)
  from terminates-noFault-strip-guards [OF this strip-bdy-noFault]
  have strip F  $\Gamma \vdash \text{strip-guards } F \ \text{bdy} \downarrow \text{Normal } s$ .
  with bdy show ?case
    by (fastforce intro: terminates.Call)
next
  case CallUndefined thus ?case by (auto intro: terminates.intros)
next
  case Stuck thus ?case by (auto intro: terminates.intros)
next

```

```

case DynCom thus ?case
  by (auto intro: terminates.intros exec.intros simp add: final-notin-def )
next
case Throw thus ?case by (auto intro: terminates.intros)
next
case Abrupt thus ?case by (auto intro: terminates.intros)
next
case (Catch c1 s c2)
have noFault-Catch:  $\Gamma \vdash \langle \text{Catch } c1 \ c2, \text{Normal } s \rangle \Rightarrow \notin \text{Fault } ' F$  by fact
hence noFault-c1:  $\Gamma \vdash \langle c1, \text{Normal } s \rangle \Rightarrow \notin \text{Fault } ' F$ 
  by (fastforce simp add: final-notin-def intro: exec.intros)
then have strip F  $\Gamma \vdash c1 \downarrow \text{Normal } s$  by (simp add: Catch.hyps)
moreover
{
  fix s'
  assume exec-strip-c1: strip F  $\Gamma \vdash \langle c1, \text{Normal } s \rangle \Rightarrow \text{Abrupt } s'$ 
  have strip F  $\Gamma \vdash c2 \downarrow \text{Normal } s'$ 
  proof –
    from exec-strip-to-exec [OF exec-strip-c1] noFault-c1
    have *:  $\Gamma \vdash \langle c1, \text{Normal } s \rangle \Rightarrow \text{Abrupt } s'$ 
      by (auto simp add: final-notin-def elim!: isFaultE)
    with * noFault-Catch have  $\Gamma \vdash \langle c2, \text{Normal } s' \rangle \Rightarrow \notin \text{Fault } ' F$ 
      by (auto simp add: final-notin-def intro: exec.intros)
    with * show ?thesis
      using Catch.hyps by (simp del: strip-simp)
  qed
}
ultimately show ?case
  using Catch.hyps by (auto intro: terminates.intros simp del: strip-simp)
qed

```

## 7.9 Miscellaneous

**lemma** *terminates-while-lemma*:

**assumes** *termi*:  $\Gamma \vdash w \downarrow fk$

**shows**  $\bigwedge k \ b \ c. \llbracket fk = \text{Normal } (f \ k); w = \text{While } b \ c; \rrbracket$

$\forall i. \Gamma \vdash \langle c, \text{Normal } (f \ i) \rangle \Rightarrow \text{Normal } (f \ (\text{Suc } i))$   
 $\implies \exists i. f \ i \notin b$

**using** *termi*

**proof** (*induct*)

**case** *WhileTrue* **thus** ?*case* **by** *blast*

**next**

**case** *WhileFalse* **thus** ?*case* **by** *blast*

**qed** *simp-all*

**lemma** *terminates-while*:

$\llbracket \Gamma \vdash (\text{While } b \ c) \downarrow \text{Normal } (f \ k); \rrbracket$

$\forall i. \Gamma \vdash \langle c, \text{Normal } (f \ i) \rangle \Rightarrow \text{Normal } (f \ (\text{Suc } i))$

$\implies \exists i. f \ i \notin b$

by (blast intro: terminates-while-lemma)

**lemma** *wf-terminates-while*:

*wf*  $\{(t,s). \Gamma \vdash (\text{While } b \ c) \downarrow \text{Normal } s \wedge s \in b \wedge$   
 $\Gamma \vdash \langle c, \text{Normal } s \rangle \Rightarrow \text{Normal } t\}$

**apply** (*subst wf-iff-no-infinite-down-chain*)

**apply** (*rule notI*)

**apply** *clarsimp*

**apply** (*insert terminates-while*)

**apply** *blast*

**done**

**lemma** *terminates-restrict-to-terminates*:

**assumes** *terminates-res*:  $\Gamma \upharpoonright_M \vdash c \downarrow s$

**assumes** *not-Stuck*:  $\Gamma \upharpoonright_M \vdash \langle c, s \rangle \Rightarrow \notin \{\text{Stuck}\}$

**shows**  $\Gamma \vdash c \downarrow s$

**using** *terminates-res not-Stuck*

**proof** (*induct*)

**case** *Skip* **show** *?case* **by** (*rule terminates.Skip*)

**next**

**case** *Basic* **show** *?case* **by** (*rule terminates.Basic*)

**next**

**case** *Spec* **show** *?case* **by** (*rule terminates.Spec*)

**next**

**case** *Guard* **thus** *?case*

**by** (*auto intro: terminates.Guard dest: notStuck-GuardD*)

**next**

**case** *GuardFault* **thus** *?case* **by** (*auto intro: terminates.GuardFault*)

**next**

**case** *Fault* **show** *?case* **by** (*rule terminates.Fault*)

**next**

**case** (*Seq c1 s c2*)

**have** *not-Stuck*:  $\Gamma \upharpoonright_M \vdash \langle \text{Seq } c1 \ c2, \text{Normal } s \rangle \Rightarrow \notin \{\text{Stuck}\}$  **by** *fact*

**hence** *c1-notStuck*:  $\Gamma \upharpoonright_M \vdash \langle c1, \text{Normal } s \rangle \Rightarrow \notin \{\text{Stuck}\}$

**by** (*rule notStuck-SeqD1*)

**show**  $\Gamma \vdash \text{Seq } c1 \ c2 \downarrow \text{Normal } s$

**proof** (*rule terminates.Seq,safe*)

**from** *c1-notStuck*

**show**  $\Gamma \vdash c1 \downarrow \text{Normal } s$

**by** (*rule Seq.hyps*)

**next**

**fix** *s'*

**assume** *exec*:  $\Gamma \vdash \langle c1, \text{Normal } s \rangle \Rightarrow s'$

**show**  $\Gamma \vdash c2 \downarrow s'$

**proof** –

**from** *exec-to-exec-restrict* [*OF exec*] **obtain** *t'* **where**

*exec-res*:  $\Gamma \upharpoonright_M \vdash \langle c1, \text{Normal } s \rangle \Rightarrow t'$  **and**

*t'-notStuck*:  $t' \neq \text{Stuck} \longrightarrow t' = s'$

**by** *blast*

```

show ?thesis
proof (cases t'=Stuck)
  case True
  with c1-notStuck exec-res have False
  by (auto simp add: final-notin-def)
  thus ?thesis ..
next
  case False
  with t'-notStuck have t': t'=s' by simp
  with not-Stuck exec-res
  have  $\Gamma \mid_M \vdash \langle c2, s' \rangle \Rightarrow \notin \{Stuck\}$ 
  by (auto dest: notStuck-SeqD2)
  with exec-res t' Seq.hyps
  show ?thesis
  by auto
qed
qed
qed
next
  case CondTrue thus ?case
  by (auto intro: terminates.CondTrue dest: notStuck-CondTrueD)
next
  case CondFalse thus ?case
  by (auto intro: terminates.CondFalse dest: notStuck-CondFalseD)
next
  case (WhileTrue s b c)
  have s:  $s \in b$  by fact
  have not-Stuck:  $\Gamma \mid_M \vdash \langle While\ b\ c, Normal\ s \rangle \Rightarrow \notin \{Stuck\}$  by fact
  with WhileTrue have c-notStuck:  $\Gamma \mid_M \vdash \langle c, Normal\ s \rangle \Rightarrow \notin \{Stuck\}$ 
  by (iprover intro: notStuck-WhileTrueD1)
  show ?case
  proof (rule terminates.WhileTrue [OF s], safe)
    from c-notStuck
    show  $\Gamma \vdash c \downarrow Normal\ s$ 
    by (rule WhileTrue.hyps)
  next
    fix s'
    assume exec:  $\Gamma \vdash \langle c, Normal\ s \rangle \Rightarrow s'$ 
    show  $\Gamma \vdash While\ b\ c \downarrow s'$ 
    proof -
      from exec-to-exec-restrict [OF exec] obtain t' where
        exec-res:  $\Gamma \mid_M \vdash \langle c, Normal\ s \rangle \Rightarrow t'$  and
        t'-notStuck:  $t' \neq Stuck \longrightarrow t' = s'$ 
      by blast
    show ?thesis
  proof (cases t'=Stuck)
    case True
    with c-notStuck exec-res have False
    by (auto simp add: final-notin-def)

```

```

      thus ?thesis ..
    next
      case False
      with t'-notStuck have t': t'=s' by simp
      with not-Stuck exec-res s
      have  $\Gamma \mid_M \vdash \langle \text{While } b \ c, s' \rangle \Rightarrow \notin \{ \text{Stuck} \}$ 
        by (auto dest: notStuck-WhileTrueD2)
      with exec-res t' WhileTrue.hyps
      show ?thesis
        by auto
    qed
  qed
next
  case WhileFalse then show ?case by (iprover intro: terminates.WhileFalse)
next
  case Call thus ?case
    by (auto intro: terminates.Call dest: notStuck-CallD restrict-SomeD)
next
  case CallUndefined
  thus ?case
    by (auto dest: notStuck-CallDefinedD)
next
  case Stuck show ?case by (rule terminates.Stuck)
next
  case DynCom
  thus ?case
    by (auto intro: terminates.DynCom dest: notStuck-DynComD)
next
  case Throw show ?case by (rule terminates.Throw)
next
  case Abrupt show ?case by (rule terminates.Abrupt)
next
  case (Catch c1 s c2)
  have not-Stuck:  $\Gamma \mid_M \vdash \langle \text{Catch } c1 \ c2, \text{Normal } s \rangle \Rightarrow \notin \{ \text{Stuck} \}$  by fact
  hence c1-notStuck:  $\Gamma \mid_M \vdash \langle c1, \text{Normal } s \rangle \Rightarrow \notin \{ \text{Stuck} \}$ 
    by (rule notStuck-CatchD1)
  show  $\Gamma \vdash \text{Catch } c1 \ c2 \downarrow \text{Normal } s$ 
  proof (rule terminates.Catch,safe)
    from c1-notStuck
    show  $\Gamma \vdash c1 \downarrow \text{Normal } s$ 
      by (rule Catch.hyps)
  next
  fix s'
  assume exec:  $\Gamma \vdash \langle c1, \text{Normal } s \rangle \Rightarrow \text{Abrupt } s'$ 
  show  $\Gamma \vdash c2 \downarrow \text{Normal } s'$ 
  proof -
    from exec-to-exec-restrict [OF exec] obtain t' where
      exec-res:  $\Gamma \mid_M \vdash \langle c1, \text{Normal } s \rangle \Rightarrow t'$  and

```

```

    t'-notStuck:  $t' \neq \text{Stuck} \longrightarrow t' = \text{Abrupt } s'$ 
  by blast
show ?thesis
proof (cases  $t' = \text{Stuck}$ )
  case True
  with c1-notStuck exec-res have False
    by (auto simp add: final-notin-def)
  thus ?thesis ..
next
  case False
  with t'-notStuck have t': t' = Abrupt s' by simp
  with not-Stuck exec-res
  have  $\Gamma \mid_M^+ \langle c2, \text{Normal } s' \rangle \Rightarrow \notin \{\text{Stuck}\}$ 
    by (auto dest: notStuck-CatchD2)
  with exec-res t' Catch.hyps
  show ?thesis
    by auto
qed
qed
qed
qed
end

```

## 8 Small-Step Semantics and Infinite Computations

**theory** *SmallStep* **imports** *Termination*  
**begin**

The *redex* of a statement is the substatement, which is actually altered by the next step in the small-step semantics.

**primrec** *redex*::  $(s', p', f) \text{ com} \Rightarrow (s', p', f) \text{ com}$   
**where**

```

redex Skip = Skip |
redex (Basic f) = (Basic f) |
redex (Spec r) = (Spec r) |
redex (Seq c1 c2) = redex c1 |
redex (Cond b c1 c2) = (Cond b c1 c2) |
redex (While b c) = (While b c) |
redex (Call p) = (Call p) |
redex (DynCom d) = (DynCom d) |
redex (Guard f b c) = (Guard f b c) |
redex (Throw) = Throw |
redex (Catch c1 c2) = redex c1

```

### 8.1 Small-Step Computation: $\Gamma \vdash (c, s) \rightarrow (c', s')$

**type-synonym**  $(s', p', f) \text{ config} = (s', p', f) \text{ com} \times (s', f) \text{ xstate}$

**inductive** *step*::  $(s', p', f) \text{ body}, (s', p', f) \text{ config}, (s', p', f) \text{ config} \Rightarrow \text{bool}$

( $\vdash (- \rightarrow / -)$ ) [81,81,81] 100)

**for**  $\Gamma::(s,p,f)$  *body*  
**where**

*Basic*:  $\Gamma \vdash (\text{Basic } f, \text{Normal } s) \rightarrow (\text{Skip}, \text{Normal } (f s))$

| *Spec*:  $(s,t) \in r \implies \Gamma \vdash (\text{Spec } r, \text{Normal } s) \rightarrow (\text{Skip}, \text{Normal } t)$   
| *SpecStuck*:  $\forall t. (s,t) \notin r \implies \Gamma \vdash (\text{Spec } r, \text{Normal } s) \rightarrow (\text{Skip}, \text{Stuck})$

| *Guard*:  $s \in g \implies \Gamma \vdash (\text{Guard } f g c, \text{Normal } s) \rightarrow (c, \text{Normal } s)$

| *GuardFault*:  $s \notin g \implies \Gamma \vdash (\text{Guard } f g c, \text{Normal } s) \rightarrow (\text{Skip}, \text{Fault } f)$

| *Seq*:  $\Gamma \vdash (c_1, s) \rightarrow (c_1', s')$

$\implies$   
 $\Gamma \vdash (\text{Seq } c_1 c_2, s) \rightarrow (\text{Seq } c_1' c_2, s')$

| *SeqSkip*:  $\Gamma \vdash (\text{Seq Skip } c_2, s) \rightarrow (c_2, s)$

| *SeqThrow*:  $\Gamma \vdash (\text{Seq Throw } c_2, \text{Normal } s) \rightarrow (\text{Throw}, \text{Normal } s)$

| *CondTrue*:  $s \in b \implies \Gamma \vdash (\text{Cond } b c_1 c_2, \text{Normal } s) \rightarrow (c_1, \text{Normal } s)$

| *CondFalse*:  $s \notin b \implies \Gamma \vdash (\text{Cond } b c_1 c_2, \text{Normal } s) \rightarrow (c_2, \text{Normal } s)$

| *WhileTrue*:  $\llbracket s \in b \rrbracket$

$\implies$   
 $\Gamma \vdash (\text{While } b c, \text{Normal } s) \rightarrow (\text{Seq } c (\text{While } b c), \text{Normal } s)$

| *WhileFalse*:  $\llbracket s \notin b \rrbracket$

$\implies$   
 $\Gamma \vdash (\text{While } b c, \text{Normal } s) \rightarrow (\text{Skip}, \text{Normal } s)$

| *Call*:  $\Gamma p = \text{Some } bdy \implies$

$\Gamma \vdash (\text{Call } p, \text{Normal } s) \rightarrow (bdy, \text{Normal } s)$

| *CallUndefined*:  $\Gamma p = \text{None} \implies$

$\Gamma \vdash (\text{Call } p, \text{Normal } s) \rightarrow (\text{Skip}, \text{Stuck})$

| *DynCom*:  $\Gamma \vdash (\text{DynCom } c, \text{Normal } s) \rightarrow (c s, \text{Normal } s)$

| *Catch*:  $\llbracket \Gamma \vdash (c_1, s) \rightarrow (c_1', s') \rrbracket$

$\implies$   
 $\Gamma \vdash (\text{Catch } c_1 c_2, s) \rightarrow (\text{Catch } c_1' c_2, s')$

| *CatchThrow*:  $\Gamma \vdash (\text{Catch Throw } c_2, \text{Normal } s) \rightarrow (c_2, \text{Normal } s)$

| *CatchSkip*:  $\Gamma \vdash (\text{Catch Skip } c_2, s) \rightarrow (\text{Skip}, s)$

| *FaultProp*:  $\llbracket c \neq \text{Skip}; \text{redex } c = c \rrbracket \implies \Gamma \vdash (c, \text{Fault } f) \rightarrow (\text{Skip}, \text{Fault } f)$

| *StuckProp*:  $\llbracket c \neq \text{Skip}; \text{redex } c = c \rrbracket \implies \Gamma \vdash (c, \text{Stuck}) \rightarrow (\text{Skip}, \text{Stuck})$

| *AbruptProp*:  $\llbracket c \neq \text{Skip}; \text{redex } c = c \rrbracket \implies \Gamma \vdash (c, \text{Abrupt } f) \rightarrow (\text{Skip}, \text{Abrupt } f)$



**lemmas** *step-induct* = *step.induct* [*of* - (*c,s*) (*c',s'*), *split-format* (*complete*), *case-names* *Basic Spec SpecStuck Guard GuardFault Seq SeqSkip SeqThrow CondTrue CondFalse WhileTrue WhileFalse Call CallUndefined DynCom Catch CatchThrow CatchSkip FaultProp StuckProp AbruptProp*, *induct set*]

**inductive-cases** *step-elim-cases* [*cases set*]:

$\Gamma \vdash (\text{Skip}, s) \rightarrow u$   
 $\Gamma \vdash (\text{Guard } f \ g \ c, s) \rightarrow u$   
 $\Gamma \vdash (\text{Basic } f, s) \rightarrow u$   
 $\Gamma \vdash (\text{Spec } r, s) \rightarrow u$   
 $\Gamma \vdash (\text{Seq } c1 \ c2, s) \rightarrow u$   
 $\Gamma \vdash (\text{Cond } b \ c1 \ c2, s) \rightarrow u$   
 $\Gamma \vdash (\text{While } b \ c, s) \rightarrow u$   
 $\Gamma \vdash (\text{Call } p, s) \rightarrow u$   
 $\Gamma \vdash (\text{DynCom } c, s) \rightarrow u$   
 $\Gamma \vdash (\text{Throw}, s) \rightarrow u$   
 $\Gamma \vdash (\text{Catch } c1 \ c2, s) \rightarrow u$

**inductive-cases** *step-Normal-elim-cases* [*cases set*]:

$\Gamma \vdash (\text{Skip}, \text{Normal } s) \rightarrow u$   
 $\Gamma \vdash (\text{Guard } f \ g \ c, \text{Normal } s) \rightarrow u$   
 $\Gamma \vdash (\text{Basic } f, \text{Normal } s) \rightarrow u$   
 $\Gamma \vdash (\text{Spec } r, \text{Normal } s) \rightarrow u$   
 $\Gamma \vdash (\text{Seq } c1 \ c2, \text{Normal } s) \rightarrow u$   
 $\Gamma \vdash (\text{Cond } b \ c1 \ c2, \text{Normal } s) \rightarrow u$   
 $\Gamma \vdash (\text{While } b \ c, \text{Normal } s) \rightarrow u$   
 $\Gamma \vdash (\text{Call } p, \text{Normal } s) \rightarrow u$   
 $\Gamma \vdash (\text{DynCom } c, \text{Normal } s) \rightarrow u$   
 $\Gamma \vdash (\text{Throw}, \text{Normal } s) \rightarrow u$   
 $\Gamma \vdash (\text{Catch } c1 \ c2, \text{Normal } s) \rightarrow u$

The final configuration is either of the form (*Skip*, -) for normal termination, or (*Throw*, *Normal s*) in case the program was started in a *Normal* state and terminated abruptly. The *Abrupt* state is not used to model abrupt termination, in contrast to the big-step semantics. Only if the program starts in an *Abrupt* states it ends in the same *Abrupt* state.

**definition** *final*:: (*'s, 'p, 'f*) *config*  $\Rightarrow$  *bool* **where**  
*final cfg* = (*fst cfg*=*Skip*  $\vee$  (*fst cfg*=*Throw*  $\wedge$  ( $\exists s. \text{snd } \text{cfg} = \text{Normal } s$ )))

**abbreviation**

*step-rtrancl* :: [(*'s, 'p, 'f*) *body*, (*'s, 'p, 'f*) *config*, (*'s, 'p, 'f*) *config*]  $\Rightarrow$  *bool*  
 $(\vdash (- \rightarrow^* / -))$  [81,81,81] 100)

**where**

$\Gamma \vdash \text{cf0} \rightarrow^* \text{cf1} \equiv (\text{CONST } \text{step } \Gamma)^{**} \text{cf0 } \text{cf1}$

**abbreviation**

$$\text{step-trancl} :: [(s',p',f) \text{ body},(s',p',f) \text{ config},(s',p',f) \text{ config}] \Rightarrow \text{bool}$$

$$(\langle + (- \rightarrow^+ / -) \rangle [81,81,81] 100)$$
**where**

$$\Gamma \vdash \text{cf0} \rightarrow^+ \text{cf1} \equiv (\text{CONST step } \Gamma)^{++} \text{cf0 cf1}$$
**8.2 Structural Properties of Small Step Computations**

**lemma** *redex-not-Seq*:  $\text{redex } c = \text{Seq } c1 \ c2 \Longrightarrow P$

**apply** *induct c*

**apply** *auto*

**done**

**lemma** *no-step-final*:

**assumes** *step*:  $\Gamma \vdash (c,s) \rightarrow (c',s')$

**shows** *final*  $(c,s) \Longrightarrow P$

**using** *step*

**by** *induct (auto simp add: final-def)*

**lemma** *no-step-final'*:

**assumes** *step*:  $\Gamma \vdash \text{cfg} \rightarrow \text{cfg}'$

**shows** *final*  $\text{cfg} \Longrightarrow P$

**using** *step*

**by** *(cases cfg, cases cfg') (auto intro: no-step-final)*

**lemma** *step-Abrupt*:

**assumes** *step*:  $\Gamma \vdash (c, s) \rightarrow (c', s')$

**shows**  $\bigwedge x. s = \text{Abrupt } x \Longrightarrow s' = \text{Abrupt } x$

**using** *step*

**by** *(induct) auto*

**lemma** *step-Fault*:

**assumes** *step*:  $\Gamma \vdash (c, s) \rightarrow (c', s')$

**shows**  $\bigwedge f. s = \text{Fault } f \Longrightarrow s' = \text{Fault } f$

**using** *step*

**by** *(induct) auto*

**lemma** *step-Stuck*:

**assumes** *step*:  $\Gamma \vdash (c, s) \rightarrow (c', s')$

**shows**  $\bigwedge f. s = \text{Stuck} \Longrightarrow s' = \text{Stuck}$

**using** *step*

**by** *(induct) auto*

**lemma** *SeqSteps*:

**assumes** *steps*:  $\Gamma \vdash \text{cfg}_1 \rightarrow^* \text{cfg}_2$

**shows**  $\bigwedge c_1 \ s \ c_1' \ s'. \llbracket \text{cfg}_1 = (c_1, s); \text{cfg}_2 = (c_1', s') \rrbracket$

$\Longrightarrow \Gamma \vdash (\text{Seq } c_1 \ c_2, s) \rightarrow^* (\text{Seq } c_1' \ c_2, s')$

**using** *steps*

**proof** *(induct rule: converse-rtranclp-induct [case-names Refl Trans])*

**case** *Refl*  
**thus** *?case*  
    **by** *simp*  
**next**  
**case** (*Trans*  $cfg_1$   $cfg''$ )  
**have** *step*:  $\Gamma \vdash cfg_1 \rightarrow cfg''$  **by** *fact*  
**have** *steps*:  $\Gamma \vdash cfg'' \rightarrow^* cfg_2$  **by** *fact*  
**have**  $cfg_1: cfg_1 = (c_1, s)$  **and**  $cfg_2: cfg_2 = (c_1', s')$  **by** *fact*+  
**obtain**  $c_1'' s''$  **where**  $cfg'': cfg''=(c_1'',s'')$   
    **by** (*cases*  $cfg''$ ) *auto*  
**from** *step*  $cfg_1$   $cfg''$   
**have**  $\Gamma \vdash (c_1, s) \rightarrow (c_1'', s'')$   
    **by** *simp*  
**hence**  $\Gamma \vdash (Seq\ c_1\ c_2, s) \rightarrow (Seq\ c_1''\ c_2, s'')$   
    **by** (*rule* *step.Seq*)  
**also from** *Trans.hyps* (3) [*OF*  $cfg''$   $cfg_2$ ]  
**have**  $\Gamma \vdash (Seq\ c_1''\ c_2, s'') \rightarrow^* (Seq\ c_1'\ c_2, s')$  .  
**finally show** *?case* .  
**qed**

**lemma** *CatchSteps*:

**assumes** *steps*:  $\Gamma \vdash cfg_1 \rightarrow^* cfg_2$   
**shows**  $\bigwedge c_1\ s\ c_1'\ s'. \llbracket cfg_1 = (c_1, s); cfg_2 = (c_1', s') \rrbracket$   
     $\implies \Gamma \vdash (Catch\ c_1\ c_2, s) \rightarrow^* (Catch\ c_1'\ c_2, s')$   
**using** *steps*  
**proof** (*induct* *rule*: *converse-rtranclp-induct* [*case-names* *Refl Trans*])  
    **case** *Refl*  
    **thus** *?case*  
    **by** *simp*  
**next**  
    **case** (*Trans*  $cfg_1$   $cfg''$ )  
    **have** *step*:  $\Gamma \vdash cfg_1 \rightarrow cfg''$  **by** *fact*  
    **have** *steps*:  $\Gamma \vdash cfg'' \rightarrow^* cfg_2$  **by** *fact*  
    **have**  $cfg_1: cfg_1 = (c_1, s)$  **and**  $cfg_2: cfg_2 = (c_1', s')$  **by** *fact*+  
    **obtain**  $c_1'' s''$  **where**  $cfg'': cfg''=(c_1'',s'')$   
    **by** (*cases*  $cfg''$ ) *auto*  
    **from** *step*  $cfg_1$   $cfg''$   
    **have**  $s: \Gamma \vdash (c_1, s) \rightarrow (c_1'', s'')$   
    **by** *simp*  
    **hence**  $\Gamma \vdash (Catch\ c_1\ c_2, s) \rightarrow (Catch\ c_1''\ c_2, s'')$   
    **by** (*rule* *step.Catch*)  
    **also from** *Trans.hyps* (3) [*OF*  $cfg''$   $cfg_2$ ]  
    **have**  $\Gamma \vdash (Catch\ c_1''\ c_2, s'') \rightarrow^* (Catch\ c_1'\ c_2, s')$  .  
    **finally show** *?case* .  
**qed**

**lemma** *steps-Fault*:  $\Gamma \vdash (c, Fault\ f) \rightarrow^* (Skip, Fault\ f)$

**proof** (*induct*  $c$ )

**case** (*Seq*  $c_1$   $c_2$ )  
**have** *steps-c1*:  $\Gamma \vdash (c_1, \text{Fault } f) \rightarrow^* (\text{Skip}, \text{Fault } f)$  **by fact**  
**have** *steps-c2*:  $\Gamma \vdash (c_2, \text{Fault } f) \rightarrow^* (\text{Skip}, \text{Fault } f)$  **by fact**  
**from** *SeqSteps* [*OF steps-c1 refl refl*]  
**have**  $\Gamma \vdash (\text{Seq } c_1 \ c_2, \text{Fault } f) \rightarrow^* (\text{Seq } \text{Skip } c_2, \text{Fault } f)$ .  
**also**  
**have**  $\Gamma \vdash (\text{Seq } \text{Skip } c_2, \text{Fault } f) \rightarrow (c_2, \text{Fault } f)$  **by** (*rule SeqSkip*)  
**also note** *steps-c2*  
**finally show** ?*case* **by simp**  
**next**  
**case** (*Catch*  $c_1$   $c_2$ )  
**have** *steps-c1*:  $\Gamma \vdash (c_1, \text{Fault } f) \rightarrow^* (\text{Skip}, \text{Fault } f)$  **by fact**  
**from** *CatchSteps* [*OF steps-c1 refl refl*]  
**have**  $\Gamma \vdash (\text{Catch } c_1 \ c_2, \text{Fault } f) \rightarrow^* (\text{Catch } \text{Skip } c_2, \text{Fault } f)$ .  
**also**  
**have**  $\Gamma \vdash (\text{Catch } \text{Skip } c_2, \text{Fault } f) \rightarrow (\text{Skip}, \text{Fault } f)$  **by** (*rule CatchSkip*)  
**finally show** ?*case* **by simp**  
**qed** (*fastforce intro: step.intros*)+

**lemma** *steps-Stuck*:  $\Gamma \vdash (c, \text{Stuck}) \rightarrow^* (\text{Skip}, \text{Stuck})$   
**proof** (*induct c*)  
**case** (*Seq*  $c_1$   $c_2$ )  
**have** *steps-c1*:  $\Gamma \vdash (c_1, \text{Stuck}) \rightarrow^* (\text{Skip}, \text{Stuck})$  **by fact**  
**have** *steps-c2*:  $\Gamma \vdash (c_2, \text{Stuck}) \rightarrow^* (\text{Skip}, \text{Stuck})$  **by fact**  
**from** *SeqSteps* [*OF steps-c1 refl refl*]  
**have**  $\Gamma \vdash (\text{Seq } c_1 \ c_2, \text{Stuck}) \rightarrow^* (\text{Seq } \text{Skip } c_2, \text{Stuck})$ .  
**also**  
**have**  $\Gamma \vdash (\text{Seq } \text{Skip } c_2, \text{Stuck}) \rightarrow (c_2, \text{Stuck})$  **by** (*rule SeqSkip*)  
**also note** *steps-c2*  
**finally show** ?*case* **by simp**

**next**  
**case** (*Catch*  $c_1$   $c_2$ )  
**have** *steps-c1*:  $\Gamma \vdash (c_1, \text{Stuck}) \rightarrow^* (\text{Skip}, \text{Stuck})$  **by fact**  
**from** *CatchSteps* [*OF steps-c1 refl refl*]  
**have**  $\Gamma \vdash (\text{Catch } c_1 \ c_2, \text{Stuck}) \rightarrow^* (\text{Catch } \text{Skip } c_2, \text{Stuck})$  .  
**also**  
**have**  $\Gamma \vdash (\text{Catch } \text{Skip } c_2, \text{Stuck}) \rightarrow (\text{Skip}, \text{Stuck})$  **by** (*rule CatchSkip*)  
**finally show** ?*case* **by simp**  
**qed** (*fastforce intro: step.intros*)+

**lemma** *steps-Abrupt*:  $\Gamma \vdash (c, \text{Abrupt } s) \rightarrow^* (\text{Skip}, \text{Abrupt } s)$   
**proof** (*induct c*)  
**case** (*Seq*  $c_1$   $c_2$ )  
**have** *steps-c1*:  $\Gamma \vdash (c_1, \text{Abrupt } s) \rightarrow^* (\text{Skip}, \text{Abrupt } s)$  **by fact**  
**have** *steps-c2*:  $\Gamma \vdash (c_2, \text{Abrupt } s) \rightarrow^* (\text{Skip}, \text{Abrupt } s)$  **by fact**  
**from** *SeqSteps* [*OF steps-c1 refl refl*]  
**have**  $\Gamma \vdash (\text{Seq } c_1 \ c_2, \text{Abrupt } s) \rightarrow^* (\text{Seq } \text{Skip } c_2, \text{Abrupt } s)$ .  
**also**  
**have**  $\Gamma \vdash (\text{Seq } \text{Skip } c_2, \text{Abrupt } s) \rightarrow (c_2, \text{Abrupt } s)$  **by** (*rule SeqSkip*)

**also note** *steps-c2*  
**finally show** *?case by simp*  
**next**  
**case** (*Catch c1 c2*)  
**have** *steps-c1*:  $\Gamma \vdash (c_1, \text{Abrupt } s) \rightarrow^* (\text{Skip}, \text{Abrupt } s)$  **by fact**  
**from** *CatchSteps* [*OF steps-c1 refl refl*]  
**have**  $\Gamma \vdash (\text{Catch } c_1 \ c_2, \text{Abrupt } s) \rightarrow^* (\text{Catch } \text{Skip } c_2, \text{Abrupt } s)$ .  
**also**  
**have**  $\Gamma \vdash (\text{Catch } \text{Skip } c_2, \text{Abrupt } s) \rightarrow (\text{Skip}, \text{Abrupt } s)$  **by** (*rule CatchSkip*)  
**finally show** *?case by simp*  
**qed** (*fastforce intro: step.intros*)+

**lemma** *step-Fault-prop*:  
**assumes** *step*:  $\Gamma \vdash (c, s) \rightarrow (c', s')$   
**shows**  $\bigwedge f. s = \text{Fault } f \implies s' = \text{Fault } f$   
**using** *step*  
**by** (*induct*) *auto*

**lemma** *step-Abrupt-prop*:  
**assumes** *step*:  $\Gamma \vdash (c, s) \rightarrow (c', s')$   
**shows**  $\bigwedge x. s = \text{Abrupt } x \implies s' = \text{Abrupt } x$   
**using** *step*  
**by** (*induct*) *auto*

**lemma** *step-Stuck-prop*:  
**assumes** *step*:  $\Gamma \vdash (c, s) \rightarrow (c', s')$   
**shows**  $s = \text{Stuck} \implies s' = \text{Stuck}$   
**using** *step*  
**by** (*induct*) *auto*

**lemma** *steps-Fault-prop*:  
**assumes** *step*:  $\Gamma \vdash (c, s) \rightarrow^* (c', s')$   
**shows**  $s = \text{Fault } f \implies s' = \text{Fault } f$   
**using** *step*  
**proof** (*induct rule: converse-rtranclp-induct2* [*case-names Refl Trans*])  
**case** *Refl* **thus** *?case by simp*  
**next**  
**case** (*Trans c s c'' s''*)  
**thus** *?case*  
**by** (*auto intro: step-Fault-prop*)  
**qed**

**lemma** *steps-Abrupt-prop*:  
**assumes** *step*:  $\Gamma \vdash (c, s) \rightarrow^* (c', s')$   
**shows**  $s = \text{Abrupt } t \implies s' = \text{Abrupt } t$   
**using** *step*  
**proof** (*induct rule: converse-rtranclp-induct2* [*case-names Refl Trans*])  
**case** *Refl* **thus** *?case by simp*  
**next**

```

case (Trans c s c'' s'')
thus ?case
  by (auto intro: step-Abrupt-prop)
qed

```

```

lemma steps-Stuck-prop:
  assumes step:  $\Gamma \vdash \langle c, s \rangle \rightarrow^* (c', s')$ 
  shows  $s = \text{Stuck} \implies s' = \text{Stuck}$ 
using step
proof (induct rule: converse-rtranclp-induct2 [case-names Refl Trans])
  case Refl thus ?case by simp
next
  case (Trans c s c'' s'')
  thus ?case
    by (auto intro: step-Stuck-prop)
qed

```

### 8.3 Equivalence between Small-Step and Big-Step Semantics

```

theorem exec-impl-steps:
  assumes exec:  $\Gamma \vdash \langle c, s \rangle \Rightarrow t$ 
  shows  $\exists c' t'. \Gamma \vdash \langle c, s \rangle \rightarrow^* (c', t') \wedge$ 
    (case t of
      Abrupt  $x \Rightarrow$  if  $s = t$  then  $c' = \text{Skip} \wedge t' = t$  else  $c' = \text{Throw} \wedge t' = \text{Normal } x$ 
      |  $- \Rightarrow c' = \text{Skip} \wedge t' = t$ )
using exec
proof (induct)
  case Skip thus ?case
    by simp
next
  case Guard thus ?case by (blast intro: step.Guard rtranclp-trans)
next
  case GuardFault thus ?case by (fastforce intro: step.GuardFault rtranclp-trans)
next
  case FaultProp show ?case by (fastforce intro: steps-Fault)
next
  case Basic thus ?case by (fastforce intro: step.Basic rtranclp-trans)
next
  case Spec thus ?case by (fastforce intro: step.Spec rtranclp-trans)
next
  case SpecStuck thus ?case by (fastforce intro: step.SpecStuck rtranclp-trans)
next
  case (Seq  $c_1 s s' c_2 t$ )
  have exec-c1:  $\Gamma \vdash \langle c_1, \text{Normal } s \rangle \Rightarrow s'$  by fact
  have exec-c2:  $\Gamma \vdash \langle c_2, s' \rangle \Rightarrow t$  by fact
  show ?case
  proof (cases  $\exists x. s' = \text{Abrupt } x$ )
    case False
    from False Seq.hyps (2)

```

**have**  $\Gamma \vdash (c_1, \text{Normal } s) \rightarrow^* (\text{Skip}, s')$   
**by** (*cases s'*) *auto*  
**hence**  $\text{seq-c}_1: \Gamma \vdash (\text{Seq } c_1 \ c_2, \text{Normal } s) \rightarrow^* (\text{Seq } \text{Skip } \ c_2, s')$   
**by** (*rule SeqSteps*) *auto*  
**from** *Seq.hyps* (4) **obtain**  $c' \ t'$  **where**  
 $\text{steps-c}_2: \Gamma \vdash (c_2, s') \rightarrow^* (c', t')$  **and**  
 $t: (\text{case } t \text{ of}$   
 $\quad \text{Abrupt } x \Rightarrow \text{if } s' = t \text{ then } c' = \text{Skip} \wedge t' = t$   
 $\quad \quad \text{else } c' = \text{Throw} \wedge t' = \text{Normal } x$   
 $\quad | - \Rightarrow c' = \text{Skip} \wedge t' = t)$   
**by** *auto*  
**note**  $\text{seq-c}_1$   
**also have**  $\Gamma \vdash (\text{Seq } \text{Skip } \ c_2, s') \rightarrow (c_2, s')$  **by** (*rule step.SeqSkip*)  
**also note**  $\text{steps-c}_2$   
**finally have**  $\Gamma \vdash (\text{Seq } c_1 \ c_2, \text{Normal } s) \rightarrow^* (c', t')$ .  
**with**  $t \text{ False}$  **show** *?thesis*  
**by** (*cases t*) *auto*  
**next**  
**case** *True*  
**then obtain**  $x$  **where**  $s': s' = \text{Abrupt } x$   
**by** *blast*  
**from**  $s' \text{ Seq.hyps}$  (2)  
**have**  $\Gamma \vdash (c_1, \text{Normal } s) \rightarrow^* (\text{Throw}, \text{Normal } x)$   
**by** *auto*  
**hence**  $\text{seq-c}_1: \Gamma \vdash (\text{Seq } c_1 \ c_2, \text{Normal } s) \rightarrow^* (\text{Seq } \text{Throw } \ c_2, \text{Normal } x)$   
**by** (*rule SeqSteps*) *auto*  
**also have**  $\Gamma \vdash (\text{Seq } \text{Throw } \ c_2, \text{Normal } x) \rightarrow (\text{Throw}, \text{Normal } x)$   
**by** (*rule SeqThrow*)  
**finally have**  $\Gamma \vdash (\text{Seq } c_1 \ c_2, \text{Normal } s) \rightarrow^* (\text{Throw}, \text{Normal } x)$ .  
**moreover**  
**from**  $\text{exec-c}_2 \ s'$  **have**  $t = \text{Abrupt } x$   
**by** (*auto intro: Abrupt-end*)  
**ultimately show** *?thesis*  
**by** *auto*  
**qed**  
**next**  
**case** *CondTrue* **thus** *?case* **by** (*blast intro: step.CondTrue rtranclp-trans*)  
**next**  
**case** *CondFalse* **thus** *?case* **by** (*blast intro: step.CondFalse rtranclp-trans*)  
**next**  
**case** (*WhileTrue s b c s' t*)  
**have**  $\text{exec-c}: \Gamma \vdash \langle c, \text{Normal } s \rangle \Rightarrow s'$  **by** *fact*  
**have**  $\text{exec-w}: \Gamma \vdash \langle \text{While } b \ c, s' \rangle \Rightarrow t$  **by** *fact*  
**have**  $b: s \in b$  **by** *fact*  
**hence**  $\text{step}: \Gamma \vdash (\text{While } b \ c, \text{Normal } s) \rightarrow (\text{Seq } c \ (\text{While } b \ c), \text{Normal } s)$   
**by** (*rule step.WhileTrue*)  
**show** *?case*  
**proof** (*cases*  $\exists x. s' = \text{Abrupt } x$ )  
**case** *False*

**from** *False WhileTrue.hyps* (3)  
**have**  $\Gamma \vdash (c, \text{Normal } s) \rightarrow^* (\text{Skip}, s')$   
**by** (*cases s'*) *auto*  
**hence** *seq-c*:  $\Gamma \vdash (\text{Seq } c \ (\text{While } b \ c), \text{Normal } s) \rightarrow^* (\text{Seq } \text{Skip} \ (\text{While } b \ c), s')$   
**by** (*rule SeqSteps*) *auto*  
**from** *WhileTrue.hyps* (5) **obtain**  $c' \ t'$  **where**  
*steps-c2*:  $\Gamma \vdash (\text{While } b \ c, s') \rightarrow^* (c', t')$  **and**  
*t*: (*case t of*  
    *Abrupt*  $x \Rightarrow$  *if*  $s' = t$  *then*  $c' = \text{Skip} \wedge t' = t$   
    *else*  $c' = \text{Throw} \wedge t' = \text{Normal } x$   
     $| - \Rightarrow c' = \text{Skip} \wedge t' = t$ )  
**by** *auto*  
**note** *step* **also note** *seq-c*  
**also have**  $\Gamma \vdash (\text{Seq } \text{Skip} \ (\text{While } b \ c), s') \rightarrow (\text{While } b \ c, s')$   
**by** (*rule step.SeqSkip*)  
**also note** *steps-c2*  
**finally have**  $\Gamma \vdash (\text{While } b \ c, \text{Normal } s) \rightarrow^* (c', t')$ .  
**with** *t False* **show** *?thesis*  
**by** (*cases t*) *auto*  
**next**  
**case** *True*  
**then obtain**  $x$  **where**  $s' = \text{Abrupt } x$   
**by** *blast*  
**note** *step*  
**also**  
**from**  $s'$  *WhileTrue.hyps* (3)  
**have**  $\Gamma \vdash (c, \text{Normal } s) \rightarrow^* (\text{Throw}, \text{Normal } x)$   
**by** *auto*  
**hence**  
*seq-c*:  $\Gamma \vdash (\text{Seq } c \ (\text{While } b \ c), \text{Normal } s) \rightarrow^* (\text{Seq } \text{Throw} \ (\text{While } b \ c), \text{Normal } x)$   
**by** (*rule SeqSteps*) *auto*  
**also have**  $\Gamma \vdash (\text{Seq } \text{Throw} \ (\text{While } b \ c), \text{Normal } x) \rightarrow (\text{Throw}, \text{Normal } x)$   
**by** (*rule SeqThrow*)  
**finally have**  $\Gamma \vdash (\text{While } b \ c, \text{Normal } s) \rightarrow^* (\text{Throw}, \text{Normal } x)$ .  
**moreover**  
**from** *exec-w s'* **have**  $t = \text{Abrupt } x$   
**by** (*auto intro: Abrupt-end*)  
**ultimately show** *?thesis*  
**by** *auto*  
**qed**  
**next**  
**case** *WhileFalse* **thus** *?case by (fastforce intro: step.WhileFalse rtrancl-trans)*  
**next**  
**case** *Call* **thus** *?case by (blast intro: step.Call rtranclp-trans)*  
**next**  
**case** *CallUndefined* **thus** *?case by (fastforce intro: step.CallUndefined rtranclp-trans)*  
**next**



```

  case StuckProp thus ?case by (fastforce intro: steps-Stuck)
next
  case DynCom thus ?case by (blast intro: step.DynCom rtranclp-trans)
next
  case Throw thus ?case by simp
next
  case AbruptProp thus ?case by (fastforce intro: steps-Abrupt)
next
  case (CatchMatch  $c_1$   $s$   $s'$   $c_2$   $t$ )
  from CatchMatch.hyps (2)
  have  $\Gamma \vdash (c_1, \text{Normal } s) \rightarrow^* (\text{Throw}, \text{Normal } s')$ 
    by simp
  hence  $\Gamma \vdash (\text{Catch } c_1 \ c_2, \text{Normal } s) \rightarrow^* (\text{Catch } \text{Throw } c_2, \text{Normal } s')$ 
    by (rule CatchSteps) auto
  also have  $\Gamma \vdash (\text{Catch } \text{Throw } c_2, \text{Normal } s') \rightarrow (c_2, \text{Normal } s')$ 
    by (rule step.CatchThrow)
  also
  from CatchMatch.hyps (4) obtain  $c' \ t'$  where
    steps-c2:  $\Gamma \vdash (c_2, \text{Normal } s') \rightarrow^* (c', t')$  and
     $t$ : (case  $t$  of
      Abrupt  $x \Rightarrow$  if  $\text{Normal } s' = t$  then  $c' = \text{Skip} \wedge t' = t$ 
      else  $c' = \text{Throw} \wedge t' = \text{Normal } x$ 
      |  $- \Rightarrow c' = \text{Skip} \wedge t' = t$ )
    by auto
  note steps-c2
  finally show ?case
    using  $t$ 
    by (auto split: xstate.splits)
next
  case (CatchMiss  $c_1$   $s$   $t$   $c_2$ )
  have  $t$ :  $\neg \text{isAbr } t$  by fact
  with CatchMiss.hyps (2)
  have  $\Gamma \vdash (c_1, \text{Normal } s) \rightarrow^* (\text{Skip}, t)$ 
    by (cases  $t$ ) auto
  hence  $\Gamma \vdash (\text{Catch } c_1 \ c_2, \text{Normal } s) \rightarrow^* (\text{Catch } \text{Skip } c_2, t)$ 
    by (rule CatchSteps) auto
  also
  have  $\Gamma \vdash (\text{Catch } \text{Skip } c_2, t) \rightarrow (\text{Skip}, t)$ 
    by (rule step.CatchSkip)
  finally show ?case
    using  $t$ 
    by (fastforce split: xstate.splits)
qed

```

**corollary** *exec-impl-steps-Normal*:  
 assumes *exec*:  $\Gamma \vdash \langle c, s \rangle \Rightarrow \text{Normal } t$   
 shows  $\Gamma \vdash \langle c, s \rangle \rightarrow^* (\text{Skip}, \text{Normal } t)$   
 using *exec-impl-steps* [*OF exec*]  
 by *auto*

**corollary** *exec-impl-steps-Normal-Abrupt*:  
**assumes**  $exec: \Gamma \vdash \langle c, Normal\ s \rangle \Rightarrow Abrupt\ t$   
**shows**  $\Gamma \vdash \langle c, Normal\ s \rangle \rightarrow^* (Throw, Normal\ t)$   
**using** *exec-impl-steps* [*OF exec*]  
**by** *auto*

**corollary** *exec-impl-steps-Abrupt-Abrupt*:  
**assumes**  $exec: \Gamma \vdash \langle c, Abrupt\ t \rangle \Rightarrow Abrupt\ t$   
**shows**  $\Gamma \vdash \langle c, Abrupt\ t \rangle \rightarrow^* (Skip, Abrupt\ t)$   
**using** *exec-impl-steps* [*OF exec*]  
**by** *auto*

**corollary** *exec-impl-steps-Fault*:  
**assumes**  $exec: \Gamma \vdash \langle c, s \rangle \Rightarrow Fault\ f$   
**shows**  $\Gamma \vdash \langle c, s \rangle \rightarrow^* (Skip, Fault\ f)$   
**using** *exec-impl-steps* [*OF exec*]  
**by** *auto*

**corollary** *exec-impl-steps-Stuck*:  
**assumes**  $exec: \Gamma \vdash \langle c, s \rangle \Rightarrow Stuck$   
**shows**  $\Gamma \vdash \langle c, s \rangle \rightarrow^* (Skip, Stuck)$   
**using** *exec-impl-steps* [*OF exec*]  
**by** *auto*

**lemma** *step-Abrupt-end*:  
**assumes**  $step: \Gamma \vdash \langle c_1, s \rangle \rightarrow \langle c_1', s' \rangle$   
**shows**  $s' = Abrupt\ x \implies s = Abrupt\ x$   
**using** *step*  
**by** *induct auto*

**lemma** *step-Stuck-end*:  
**assumes**  $step: \Gamma \vdash \langle c_1, s \rangle \rightarrow \langle c_1', s' \rangle$   
**shows**  $s' = Stuck \implies$   
 $s = Stuck \vee$   
 $(\exists r\ x. redex\ c_1 = Spec\ r \wedge s = Normal\ x \wedge (\forall t. \langle x, t \rangle \notin r)) \vee$   
 $(\exists p\ x. redex\ c_1 = Call\ p \wedge s = Normal\ x \wedge \Gamma\ p = None)$   
**using** *step*  
**by** *induct auto*

**lemma** *step-Fault-end*:  
**assumes**  $step: \Gamma \vdash \langle c_1, s \rangle \rightarrow \langle c_1', s' \rangle$   
**shows**  $s' = Fault\ f \implies$   
 $s = Fault\ f \vee$   
 $(\exists g\ c\ x. redex\ c_1 = Guard\ f\ g\ c \wedge s = Normal\ x \wedge x \notin g)$   
**using** *step*  
**by** *induct auto*

```

lemma exec-redex-Stuck:
 $\Gamma \vdash \langle \text{redex } c, s \rangle \Rightarrow \text{Stuck} \implies \Gamma \vdash \langle c, s \rangle \Rightarrow \text{Stuck}$ 
proof (induct c)
  case Seq
  thus ?case
    by (cases s) (auto intro: exec.intros elim:exec-elim-cases)
next
  case Catch
  thus ?case
    by (cases s) (auto intro: exec.intros elim:exec-elim-cases)
qed simp-all

lemma exec-redex-Fault:
 $\Gamma \vdash \langle \text{redex } c, s \rangle \Rightarrow \text{Fault } f \implies \Gamma \vdash \langle c, s \rangle \Rightarrow \text{Fault } f$ 
proof (induct c)
  case Seq
  thus ?case
    by (cases s) (auto intro: exec.intros elim:exec-elim-cases)
next
  case Catch
  thus ?case
    by (cases s) (auto intro: exec.intros elim:exec-elim-cases)
qed simp-all

lemma step-extend:
  assumes step:  $\Gamma \vdash (c, s) \rightarrow (c', s')$ 
  shows  $\bigwedge t. \Gamma \vdash \langle c', s' \rangle \Rightarrow t \implies \Gamma \vdash \langle c, s \rangle \Rightarrow t$ 
using step
proof (induct)
  case Basic thus ?case
    by (fastforce intro: exec.intros elim: exec-Normal-elim-cases)
next
  case Spec thus ?case
    by (fastforce intro: exec.intros elim: exec-Normal-elim-cases)
next
  case SpecStuck thus ?case
    by (fastforce intro: exec.intros elim: exec-Normal-elim-cases)
next
  case Guard thus ?case
    by (fastforce intro: exec.intros elim: exec-Normal-elim-cases)
next
  case GuardFault thus ?case
    by (fastforce intro: exec.intros elim: exec-Normal-elim-cases)
next
  case (Seq c1 s c1' s' c2)
  have step:  $\Gamma \vdash (c_1, s) \rightarrow (c_1', s')$  by fact
  have exec':  $\Gamma \vdash \langle \text{Seq } c_1' c_2, s' \rangle \Rightarrow t$  by fact
  show ?case
  proof (cases s)

```

```

case (Normal  $x$ )
note  $s$ -Normal = this
show ?thesis
proof (cases  $s'$ )
  case (Normal  $x'$ )
  from  $exec'$  [simplified Normal] obtain  $s''$  where
     $exec-c_1'$ :  $\Gamma \vdash \langle c_1', Normal\ x' \rangle \Rightarrow s''$  and
     $exec-c_2$ :  $\Gamma \vdash \langle c_2, s'' \rangle \Rightarrow t$ 
  by cases
  from Seq.hyps (2) Normal  $exec-c_1'$   $s$ -Normal
  have  $\Gamma \vdash \langle c_1, Normal\ x \rangle \Rightarrow s''$ 
  by simp
  from  $exec$ .Seq [OF this  $exec-c_2$ ]  $s$ -Normal
  show ?thesis by simp
next
  case (Abrupt  $x'$ )
  with  $exec'$  have  $t = Abrupt\ x'$ 
  by (auto intro: Abrupt-end)
  moreover
  from step Abrupt
  have  $s = Abrupt\ x'$ 
  by (auto intro: step-Abrupt-end)
  ultimately
  show ?thesis
  by (auto intro: exec.intros)
next
  case (Fault  $f$ )
  from step-Fault-end [OF step this]  $s$ -Normal
  obtain  $g\ c$  where
     $redex-c_1$ :  $redex\ c_1 = Guard\ f\ g\ c$  and
     $fail$ :  $x \notin g$ 
  by auto
  hence  $\Gamma \vdash \langle redex\ c_1, Normal\ x \rangle \Rightarrow Fault\ f$ 
  by (auto intro: exec.intros)
  from  $exec$ -redex-Fault [OF this]
  have  $\Gamma \vdash \langle c_1, Normal\ x \rangle \Rightarrow Fault\ f$ .
  moreover from Fault  $exec'$  have  $t = Fault\ f$ 
  by (auto intro: Fault-end)
  ultimately
  show ?thesis
  using  $s$ -Normal
  by (auto intro: exec.intros)
next
  case Stuck
  from step-Stuck-end [OF step this]  $s$ -Normal
  have  $(\exists r. redex\ c_1 = Spec\ r \wedge (\forall t. (x, t) \notin r)) \vee$ 
     $(\exists p. redex\ c_1 = Call\ p \wedge \Gamma\ p = None)$ 
  by auto
  moreover

```

```

{
  fix r
  assume redex c1 = Spec r and (∀ t. (x, t) ∉ r)
  hence Γ ⊢ ⟨redex c1, Normal x⟩ ⇒ Stuck
    by (auto intro: exec.intros)
  from exec-redex-Stuck [OF this]
  have Γ ⊢ ⟨c1, Normal x⟩ ⇒ Stuck.
  moreover from Stuck exec' have t=Stuck
    by (auto intro: Stuck-end)
  ultimately
  have ?thesis
    using s-Normal
    by (auto intro: exec.intros)
}
moreover
{
  fix p
  assume redex c1 = Call p and Γ p = None
  hence Γ ⊢ ⟨redex c1, Normal x⟩ ⇒ Stuck
    by (auto intro: exec.intros)
  from exec-redex-Stuck [OF this]
  have Γ ⊢ ⟨c1, Normal x⟩ ⇒ Stuck.
  moreover from Stuck exec' have t=Stuck
    by (auto intro: Stuck-end)
  ultimately
  have ?thesis
    using s-Normal
    by (auto intro: exec.intros)
}
ultimately show ?thesis
  by auto
qed
next
case (Abrupt x)
from step-Abrupt [OF step this]
have s'=Abrupt x.
with exec'
have t=Abrupt x
  by (auto intro: Abrupt-end)
with Abrupt
show ?thesis
  by (auto intro: exec.intros)
next
case (Fault f)
from step-Fault [OF step this]
have s'=Fault f.
with exec'
have t=Fault f
  by (auto intro: Fault-end)

```

```

    with Fault
    show ?thesis
      by (auto intro: exec.intros)
  next
    case Stuck
    from step-Stuck [OF step this]
    have s'=Stuck.
    with exec'
    have t=Stuck
      by (auto intro: Stuck-end)
    with Stuck
    show ?thesis
      by (auto intro: exec.intros)
  qed
next
  case (SeqSkip c2 s t) thus ?case
    by (cases s) (fastforce intro: exec.intros elim: exec-elim-cases)+
next
  case (SeqThrow c2 s t) thus ?case
    by (fastforce intro: exec.intros elim: exec-elim-cases)+
next
  case CondTrue thus ?case
    by (fastforce intro: exec.intros elim: exec-Normal-elim-cases)
next
  case CondFalse thus ?case
    by (fastforce intro: exec.intros elim: exec-Normal-elim-cases)
next
  case WhileTrue thus ?case
    by (fastforce intro: exec.intros elim: exec-Normal-elim-cases)
next
  case WhileFalse thus ?case
    by (fastforce intro: exec.intros elim: exec-Normal-elim-cases)
next
  case Call thus ?case
    by (fastforce intro: exec.intros elim: exec-Normal-elim-cases)
next
  case CallUndefined thus ?case
    by (fastforce intro: exec.intros elim: exec-Normal-elim-cases)
next
  case DynCom thus ?case
    by (fastforce intro: exec.intros elim: exec-Normal-elim-cases)
next
  case (Catch c1 s c1' s' c2 t)
  have step:  $\Gamma \vdash (c_1, s) \rightarrow (c_1', s')$  by fact
  have exec':  $\Gamma \vdash \langle \text{Catch } c_1' c_2, s' \rangle \Rightarrow t$  by fact
  show ?case
  proof (cases s)
    case (Normal x)
    note s-Normal = this

```

```

show ?thesis
proof (cases s')
  case (Normal x')
    from exec' [simplified Normal]
    show ?thesis
    proof (cases)
      fix s''
        assume exec-c1':  $\Gamma \vdash \langle c_1', \text{Normal } x' \rangle \Rightarrow \text{Abrupt } s''$ 
        assume exec-c2:  $\Gamma \vdash \langle c_2, \text{Normal } s'' \rangle \Rightarrow t$ 
        from Catch.hyps (2) Normal exec-c1' s-Normal
        have  $\Gamma \vdash \langle c_1, \text{Normal } x \rangle \Rightarrow \text{Abrupt } s''$ 
          by simp
        from exec.CatchMatch [OF this exec-c2] s-Normal
        show ?thesis by simp
      next
        assume exec-c1':  $\Gamma \vdash \langle c_1', \text{Normal } x' \rangle \Rightarrow t$ 
        assume t:  $\neg \text{isAbr } t$ 
        from Catch.hyps (2) Normal exec-c1' s-Normal
        have  $\Gamma \vdash \langle c_1, \text{Normal } x \rangle \Rightarrow t$ 
          by simp
        from exec.CatchMiss [OF this t] s-Normal
        show ?thesis by simp
    qed
  next
    case (Abrupt x')
    with exec' have  $t = \text{Abrupt } x'$ 
      by (auto intro: Abrupt-end)
    moreover
      from step Abrupt
      have  $s = \text{Abrupt } x'$ 
        by (auto intro: step-Abrupt-end)
    ultimately
      show ?thesis
        by (auto intro: exec.intros)
  next
    case (Fault f)
    from step-Fault-end [OF step this] s-Normal
    obtain g c where
      redex-c1: redex c1 = Guard f g c and
      fail:  $x \notin g$ 
    by auto
    hence  $\Gamma \vdash \langle \text{redex } c_1, \text{Normal } x \rangle \Rightarrow \text{Fault } f$ 
      by (auto intro: exec.intros)
    from exec-redex-Fault [OF this]
    have  $\Gamma \vdash \langle c_1, \text{Normal } x \rangle \Rightarrow \text{Fault } f$ .
    moreover from Fault exec' have  $t = \text{Fault } f$ 
      by (auto intro: Fault-end)
    ultimately
      show ?thesis

```

```

    using s-Normal
    by (auto intro: exec.intros)
next
case Stuck
from step-Stuck-end [OF step this] s-Normal
have  $(\exists r. \text{redex } c_1 = \text{Spec } r \wedge (\forall t. (x, t) \notin r)) \vee$ 
     $(\exists p. \text{redex } c_1 = \text{Call } p \wedge \Gamma p = \text{None})$ 
    by auto
moreover
{
  fix r
  assume  $\text{redex } c_1 = \text{Spec } r$  and  $(\forall t. (x, t) \notin r)$ 
  hence  $\Gamma \vdash \langle \text{redex } c_1, \text{Normal } x \rangle \Rightarrow \text{Stuck}$ 
    by (auto intro: exec.intros)
  from exec-redex-Stuck [OF this]
  have  $\Gamma \vdash \langle c_1, \text{Normal } x \rangle \Rightarrow \text{Stuck}$ .
  moreover from Stuck exec' have  $t = \text{Stuck}$ 
    by (auto intro: Stuck-end)
  ultimately
  have ?thesis
    using s-Normal
    by (auto intro: exec.intros)
}
moreover
{
  fix p
  assume  $\text{redex } c_1 = \text{Call } p$  and  $\Gamma p = \text{None}$ 
  hence  $\Gamma \vdash \langle \text{redex } c_1, \text{Normal } x \rangle \Rightarrow \text{Stuck}$ 
    by (auto intro: exec.intros)
  from exec-redex-Stuck [OF this]
  have  $\Gamma \vdash \langle c_1, \text{Normal } x \rangle \Rightarrow \text{Stuck}$ .
  moreover from Stuck exec' have  $t = \text{Stuck}$ 
    by (auto intro: Stuck-end)
  ultimately
  have ?thesis
    using s-Normal
    by (auto intro: exec.intros)
}
ultimately show ?thesis
  by auto
qed
next
case (Abrupt x)
from step-Abrupt [OF step this]
have  $s' = \text{Abrupt } x$ .
with exec'
have  $t = \text{Abrupt } x$ 
  by (auto intro: Abrupt-end)
with Abrupt

```



```

    show ?thesis
      by (auto intro: exec.intros)
  next
    case (Fault f)
    from step-Fault [OF step this]
    have s'=Fault f.
    with exec'
    have t=Fault f
      by (auto intro: Fault-end)
    with Fault
    show ?thesis
      by (auto intro: exec.intros)
  next
    case Stuck
    from step-Stuck [OF step this]
    have s'=Stuck.
    with exec'
    have t=Stuck
      by (auto intro: Stuck-end)
    with Stuck
    show ?thesis
      by (auto intro: exec.intros)
  qed
next
  case CatchThrow thus ?case
    by (fastforce intro: exec.intros elim: exec-Normal-elim-cases)
next
  case CatchSkip thus ?case
    by (fastforce intro: exec.intros elim: exec-elim-cases)
next
  case FaultProp thus ?case
    by (fastforce intro: exec.intros elim: exec-elim-cases)
next
  case StuckProp thus ?case
    by (fastforce intro: exec.intros elim: exec-elim-cases)
next
  case AbruptProp thus ?case
    by (fastforce intro: exec.intros elim: exec-elim-cases)
qed

theorem steps-Skip-impl-exec:
  assumes steps:  $\Gamma \vdash (c,s) \rightarrow^* (Skip,t)$ 
  shows  $\Gamma \vdash \langle c,s \rangle \Rightarrow t$ 
using steps
proof (induct rule: converse-rtranclp-induct2 [case-names Refl Trans])
  case Refl thus ?case
    by (cases t) (auto intro: exec.intros)
next
  case (Trans c s c' s')

```

**have**  $\Gamma \vdash (c, s) \rightarrow (c', s')$  **and**  $\Gamma \vdash \langle c', s' \rangle \Rightarrow t$  **by** *fact+*  
**thus** *?case*  
**by** (*rule step-extend*)  
**qed**

**theorem** *steps-Throw-impl-exec*:  
**assumes** *steps*:  $\Gamma \vdash (c, s) \rightarrow^* (\text{Throw}, \text{Normal } t)$   
**shows**  $\Gamma \vdash \langle c, s \rangle \Rightarrow \text{Abrupt } t$   
**using** *steps*  
**proof** (*induct rule: converse-rtranclp-induct2 [case-names Refl Trans]*)  
**case** *Refl* **thus** *?case*  
**by** (*auto intro: exec.intros*)  
**next**  
**case** (*Trans c s c' s'*)  
**have**  $\Gamma \vdash (c, s) \rightarrow (c', s')$  **and**  $\Gamma \vdash \langle c', s' \rangle \Rightarrow \text{Abrupt } t$  **by** *fact+*  
**thus** *?case*  
**by** (*rule step-extend*)  
**qed**

#### 8.4 Infinite Computations: $\Gamma \vdash (c, s) \rightarrow \dots(\infty)$

**definition** *inf*::  $(s, p, f) \text{ body} \Rightarrow (s, p, f) \text{ config} \Rightarrow \text{bool}$   
 $(\langle \vdash - \rightarrow \dots(\infty) \rangle [60, 80] 100)$  **where**  
 $\Gamma \vdash \text{cfg} \rightarrow \dots(\infty) \equiv (\exists f. f (0::\text{nat}) = \text{cfg} \wedge (\forall i. \Gamma \vdash f i \rightarrow f (i+1)))$

**lemma** *not-infI*:  $\llbracket \bigwedge f. \llbracket f 0 = \text{cfg}; \bigwedge i. \Gamma \vdash f i \rightarrow f (\text{Suc } i) \rrbracket \implies \text{False} \rrbracket$   
 $\implies \neg \Gamma \vdash \text{cfg} \rightarrow \dots(\infty)$   
**by** (*auto simp add: inf-def*)

#### 8.5 Equivalence between Termination and the Absence of Infinite Computations

**lemma** *step-preserves-termination*:  
**assumes** *step*:  $\Gamma \vdash (c, s) \rightarrow (c', s')$   
**shows**  $\Gamma \vdash c \downarrow s \implies \Gamma \vdash c' \downarrow s'$   
**using** *step*  
**proof** (*induct*)  
**case** *Basic* **thus** *?case* **by** (*fastforce intro: terminates.intros*)  
**next**  
**case** *Spec* **thus** *?case* **by** (*fastforce intro: terminates.intros*)  
**next**  
**case** *SpecStuck* **thus** *?case* **by** (*fastforce intro: terminates.intros*)  
**next**  
**case** *Guard* **thus** *?case*  
**by** (*fastforce intro: terminates.intros elim: terminates-Normal-elim-cases*)  
**next**  
**case** *GuardFault* **thus** *?case* **by** (*fastforce intro: terminates.intros*)  
**next**  
**case** (*Seq c<sub>1</sub> s c<sub>1</sub>' s' c<sub>2</sub>*) **thus** *?case*

```

    apply (cases s)
    apply (cases s')
    apply (fastforce intro: terminates.intros step-extend
           elim: terminates-Normal-elim-cases)
    apply (fastforce intro: terminates.intros dest: step-Abrupt-prop
           step-Fault-prop step-Stuck-prop)+
    done
next
case (SeqSkip c2 s)
thus ?case
    apply (cases s)
    apply (fastforce intro: terminates.intros exec.intros
           elim: terminates-Normal-elim-cases )+
    done
next
case (SeqThrow c2 s)
thus ?case
    by (fastforce intro: terminates.intros exec.intros
        elim: terminates-Normal-elim-cases )
next
case CondTrue
thus ?case
    by (fastforce intro: terminates.intros exec.intros
        elim: terminates-Normal-elim-cases )
next
case CondFalse
thus ?case
    by (fastforce intro: terminates.intros
        elim: terminates-Normal-elim-cases )
next
case WhileTrue
thus ?case
    by (fastforce intro: terminates.intros
        elim: terminates-Normal-elim-cases )
next
case WhileFalse
thus ?case
    by (fastforce intro: terminates.intros
        elim: terminates-Normal-elim-cases )
next
case Call
thus ?case
    by (fastforce intro: terminates.intros
        elim: terminates-Normal-elim-cases )
next
case CallUndefined
thus ?case
    by (fastforce intro: terminates.intros
        elim: terminates-Normal-elim-cases )

```

```

next
  case DynCom
  thus ?case
  by (fastforce intro: terminates.intros
      elim: terminates-Normal-elim-cases )
next
  case (Catch  $c_1$   $s$   $c_1'$   $s'$   $c_2$ ) thus ?case
  apply (cases  $s$ )
  apply (cases  $s'$ )
  apply (fastforce intro: terminates.intros step-extend
      elim: terminates-Normal-elim-cases)
  apply (fastforce intro: terminates.intros dest: step-Abrupt-prop
      step-Fault-prop step-Stuck-prop)+
  done
next
  case CatchThrow
  thus ?case
  by (fastforce intro: terminates.intros exec.intros
      elim: terminates-Normal-elim-cases )
next
  case (CatchSkip  $c_2$   $s$ )
  thus ?case
  by (cases  $s$ ) (fastforce intro: terminates.intros)+
next
  case FaultProp thus ?case by (fastforce intro: terminates.intros)
next
  case StuckProp thus ?case by (fastforce intro: terminates.intros)
next
  case AbruptProp thus ?case by (fastforce intro: terminates.intros)
qed

lemma steps-preserves-termination:
  assumes steps:  $\Gamma \vdash (c, s) \rightarrow^* (c', s')$ 
  shows  $\Gamma \vdash c \downarrow s \implies \Gamma \vdash c' \downarrow s'$ 
using steps
proof (induct rule: rtranclp-induct2 [consumes 1, case-names Refl Trans])
  case Refl thus ?case .
next
  case Trans
  thus ?case
  by (blast dest: step-preserves-termination)
qed

ML <
  ML-Thms.bind-thm (tranclp-induct2, Split-Rule.split-rule @ {context}
    (Rule-Insts.read-instantiate @ {context}
      [((( $a$ , 0), Position.none), (aa, ab)), ((( $b$ , 0), Position.none), (ba, bb))]) []
    @ {thm tranclp-induct}));
>

```

**lemma** *steps-preserves-termination'*:  
**assumes** *steps*:  $\Gamma \vdash (c, s) \rightarrow^+ (c', s')$   
**shows**  $\Gamma \vdash c \downarrow s \implies \Gamma \vdash c' \downarrow s'$   
**using** *steps*  
**proof** (*induct rule: tranclp-induct2* [*consumes 1, case-names Step Trans*])  
**case** *Step* **thus** ?*case* **by** (*blast intro: step-preserves-termination*)  
**next**  
**case** *Trans*  
**thus** ?*case*  
**by** (*blast dest: step-preserves-termination*)  
**qed**

**definition** *head-com*::  $(s', p', f) \text{ com} \Rightarrow (s', p', f) \text{ com}$   
**where**  
*head-com* *c* =  
(case *c* of  
Seq *c*<sub>1</sub> *c*<sub>2</sub>  $\Rightarrow c_1$   
| *Catch* *c*<sub>1</sub> *c*<sub>2</sub>  $\Rightarrow c_1$   
| -  $\Rightarrow c$ )

**definition** *head*::  $(s', p', f) \text{ config} \Rightarrow (s', p', f) \text{ config}$   
**where** *head* *cfg* = (*head-com* (*fst* *cfg*), *snd* *cfg*)

**lemma** *le-Suc-cases*:  $\llbracket \bigwedge i. \llbracket i < k \rrbracket \implies P \ i; P \ k \rrbracket \implies \forall i < (Suc \ k). P \ i$   
**apply** *clarify*  
**apply** (*case-tac i=k*)  
**apply** *auto*  
**done**

**lemma** *redex-Seq-False*:  $\bigwedge c' \ c''. (\text{redex } c = \text{Seq } c'' \ c') = \text{False}$   
**by** (*induct c*) *auto*

**lemma** *redex-Catch-False*:  $\bigwedge c' \ c''. (\text{redex } c = \text{Catch } c'' \ c') = \text{False}$   
**by** (*induct c*) *auto*

**lemma** *infinite-computation-extract-head-Seq*:  
**assumes** *inf-comp*:  $\forall i::\text{nat}. \Gamma \vdash f \ i \rightarrow f \ (i+1)$   
**assumes** *f-0*:  $f \ 0 = (\text{Seq } c_1 \ c_2, s)$   
**assumes** *not-fin*:  $\forall i < k. \neg \text{final } (\text{head } (f \ i))$   
**shows**  $\forall i < k. (\exists c' \ s'. f \ (i+1) = (\text{Seq } c' \ c_2, s')) \wedge$   
 $\Gamma \vdash \text{head } (f \ i) \rightarrow \text{head } (f \ (i+1))$   
(is  $\forall i < k. ?P \ i$ )  
**using** *not-fin*  
**proof** (*induct k*)

```

case 0
show ?case by simp
next
case (Suc k)
have not-fin-Suc:
   $\forall i < \text{Suc } k. \neg \text{final } (\text{head } (f \ i))$  by fact
from this[rule-format] have not-fin-k:
   $\forall i < k. \neg \text{final } (\text{head } (f \ i))$ 
apply clarify
apply (subgoal-tac i < Suc k)
apply blast
apply simp
done

from Suc.hyps [OF this]
have hyp:  $\forall i < k. (\exists c' s'. f \ (i + 1) = (\text{Seq } c' \ c_2, \ s')) \wedge$ 
   $\Gamma \vdash \text{head } (f \ i) \rightarrow \text{head } (f \ (i + 1)).$ 
show ?case
proof (rule le-Suc-cases)
  fix i
  assume i < k
  then show ?P i
    by (rule hyp [rule-format])
next
show ?P k
proof –
  from hyp [rule-format, of k - 1] f-0
  obtain c' fs' L' s' where f-k: f k = (Seq c' c2, s')
    by (cases k) auto
  from inf-comp [rule-format, of k] f-k
  have  $\Gamma \vdash (\text{Seq } c' \ c_2, \ s') \rightarrow f \ (k + 1)$ 
    by simp
  moreover
  from not-fin-Suc [rule-format, of k] f-k
  have  $\neg \text{final } (c', s')$ 
    by (simp add: final-def head-def head-com-def)
  ultimately
  obtain c'' s'' where
     $\Gamma \vdash (c', s') \rightarrow (c'', s'')$  and
     $f \ (k + 1) = (\text{Seq } c'' \ c_2, \ s'')$ 
    by cases (auto simp add: redex-Seq-False final-def)
  with f-k
  show ?thesis
    by (simp add: head-def head-com-def)
qed
qed
qed

```

**lemma** infinite-computation-extract-head-Catch:

```

assumes inf-comp:  $\forall i::nat. \Gamma \vdash f\ i \rightarrow f\ (i+1)$ 
assumes f-0:  $f\ 0 = (Catch\ c_1\ c_2, s)$ 
assumes not-fin:  $\forall i < k. \neg final\ (head\ (f\ i))$ 
shows  $\forall i < k. (\exists c'\ s'. f\ (i + 1) = (Catch\ c'\ c_2, s')) \wedge$ 
 $\Gamma \vdash head\ (f\ i) \rightarrow head\ (f\ (i+1))$ 
  (is  $\forall i < k. ?P\ i$ )
using not-fin
proof (induct k)
  case 0
  show ?case by simp
next
  case (Suc k)
  have not-fin-Suc:
     $\forall i < Suc\ k. \neg final\ (head\ (f\ i))$  by fact
  from this[rule-format] have not-fin-k:
     $\forall i < k. \neg final\ (head\ (f\ i))$ 
  apply clarify
  apply (subgoal-tac i < Suc k)
  apply blast
  apply simp
  done

from Suc.hyps [OF this]
have hyp:  $\forall i < k. (\exists c'\ s'. f\ (i + 1) = (Catch\ c'\ c_2, s')) \wedge$ 
 $\Gamma \vdash head\ (f\ i) \rightarrow head\ (f\ (i + 1))$ .
show ?case
proof (rule le-Suc-cases)
  fix i
  assume  $i < k$ 
  then show ?P i
    by (rule hyp [rule-format])
next
show ?P k
proof –
  from hyp [rule-format, of k - 1] f-0
  obtain  $c'\ fs'\ L'\ s'$  where  $f-k: f\ k = (Catch\ c'\ c_2, s')$ 
    by (cases k) auto
  from inf-comp [rule-format, of k] f-k
  have  $\Gamma \vdash (Catch\ c'\ c_2, s') \rightarrow f\ (k + 1)$ 
    by simp
  moreover
  from not-fin-Suc [rule-format, of k] f-k
  have  $\neg final\ (c', s')$ 
    by (simp add: final-def head-def head-com-def)
  ultimately
  obtain  $c''\ s''$  where
     $\Gamma \vdash (c', s') \rightarrow (c'', s'')$  and
     $f\ (k + 1) = (Catch\ c''\ c_2, s'')$ 
  by cases (auto simp add: redex-Catch-False final-def)+

```

```

    with f-k
    show ?thesis
      by (simp add: head-def head-com-def)
  qed
qed
qed

```

**lemma** *no-inf-Throw*:  $\neg \Gamma \vdash (\text{Throw}, s) \rightarrow \dots(\infty)$

```

proof
  assume  $\Gamma \vdash (\text{Throw}, s) \rightarrow \dots(\infty)$ 
  then obtain f where
    step [rule-format]:  $\forall i::\text{nat}. \Gamma \vdash f\ i \rightarrow f\ (i+1)$  and
    f-0:  $f\ 0 = (\text{Throw}, s)$ 
    by (auto simp add: inf-def)
  from step [of 0, simplified f-0] step [of 1]
  show False
    by cases (auto elim: step-elim-cases)
qed

```

**lemma** *split-inf-Seq*:

```

  assumes inf-comp:  $\Gamma \vdash (\text{Seq}\ c_1\ c_2, s) \rightarrow \dots(\infty)$ 
  shows  $\Gamma \vdash (c_1, s) \rightarrow \dots(\infty) \vee$ 
    ( $\exists s'. \Gamma \vdash (c_1, s) \rightarrow^* (\text{Skip}, s') \wedge \Gamma \vdash (c_2, s') \rightarrow \dots(\infty)$ )
proof -

```

from *inf-comp* obtain f where

```

  step:  $\forall i::\text{nat}. \Gamma \vdash f\ i \rightarrow f\ (i+1)$  and
  f-0:  $f\ 0 = (\text{Seq}\ c_1\ c_2, s)$ 
  by (auto simp add: inf-def)

```

from f-0 have head-f-0:  $\text{head}\ (f\ 0) = (c_1, s)$

```

  by (simp add: head-def head-com-def)

```

show ?thesis

**proof** (cases  $\exists i. \text{final}\ (\text{head}\ (f\ i))$ )

case True

define k where  $k = (\text{LEAST}\ i. \text{final}\ (\text{head}\ (f\ i)))$

have less-k:  $\forall i < k. \neg \text{final}\ (\text{head}\ (f\ i))$

```

  apply (intro allI impI)

```

```

  apply (unfold k-def)

```

```

  apply (drule not-less-Least)

```

```

  apply auto

```

done

from *infinite-computation-extract-head-Seq* [OF step f-0 this]

obtain step-head:  $\forall i < k. \Gamma \vdash \text{head}\ (f\ i) \rightarrow \text{head}\ (f\ (i + 1))$  and

```

  conf:  $\forall i < k. (\exists c'\ s'. f\ (i + 1) = (\text{Seq}\ c'\ c_2, s'))$ 

```

by blast

from True

have final-f-k:  $\text{final}\ (\text{head}\ (f\ k))$

```

  apply -

```

```

  apply (erule exE)

```

```

  apply (drule LeastI)

```



```

    apply (simp add: k-def)
  done
moreover
from f-0 conf [rule-format, of k - 1]
obtain c' s' where f-k: f k = (Seq c' c2, s')
  by (cases k) auto
moreover
from step-head have steps-head:  $\Gamma \vdash \text{head } (f\ 0) \rightarrow^* \text{head } (f\ k)$ 
proof (induct k)
  case 0 thus ?case by simp
next
case (Suc m)
have step:  $\forall i < \text{Suc } m. \Gamma \vdash \text{head } (f\ i) \rightarrow \text{head } (f\ (i + 1))$  by fact
hence  $\forall i < m. \Gamma \vdash \text{head } (f\ i) \rightarrow \text{head } (f\ (i + 1))$ 
  by auto
hence  $\Gamma \vdash \text{head } (f\ 0) \rightarrow^* \text{head } (f\ m)$ 
  by (rule Suc.hyps)
also from step [rule-format, of m]
have  $\Gamma \vdash \text{head } (f\ m) \rightarrow \text{head } (f\ (m + 1))$  by simp
finally show ?case by simp
qed
{
  assume f-k: f k = (Seq Skip c2, s')
  with steps-head
  have  $\Gamma \vdash (c_1, s) \rightarrow^* (\text{Skip}, s')$ 
    using head-f-0
    by (simp add: head-def head-com-def)
  moreover
  from step [rule-format, of k] f-k
  obtain  $\Gamma \vdash (\text{Seq Skip } c_2, s') \rightarrow (c_2, s')$  and
    f-Suc-k: f (k + 1) = (c2, s')
    by (fastforce elim: step.cases intro: step.intros)
  define g where g i = f (i + (k + 1)) for i
  from f-Suc-k
  have g-0: g 0 = (c2, s')
    by (simp add: g-def)
  from step
  have  $\forall i. \Gamma \vdash g\ i \rightarrow g\ (i + 1)$ 
    by (simp add: g-def)
  with g-0 have  $\Gamma \vdash (c_2, s') \rightarrow \dots(\infty)$ 
    by (auto simp add: inf-def)
  ultimately
  have ?thesis
    by auto
}
moreover
{
  fix x
  assume s': s' = Normal x and f-k: f k = (Seq Throw c2, s')

```

```

from step [rule-format, of k] f-k s'
obtain  $\Gamma \vdash (\text{Seq Throw } c_2, s') \rightarrow (\text{Throw}, s')$  and
  f-Suc-k:  $f (k + 1) = (\text{Throw}, s')$ 
  by (fastforce elim: step-elim-cases intro: step.intros)
define g where  $g\ i = f (i + (k + 1))$  for i
from f-Suc-k
have g-0:  $g\ 0 = (\text{Throw}, s')$ 
  by (simp add: g-def)
from step
have  $\forall i. \Gamma \vdash g\ i \rightarrow g (i + 1)$ 
  by (simp add: g-def)
with g-0 have  $\Gamma \vdash (\text{Throw}, s') \rightarrow \dots(\infty)$ 
  by (auto simp add: inf-def)
with no-inf-Throw
have ?thesis
  by auto
}
ultimately
show ?thesis
  by (auto simp add: final-def head-def head-com-def)
next
case False
then have not-fin:  $\forall i. \neg \text{final} (\text{head} (f\ i))$ 
  by blast
have  $\forall i. \Gamma \vdash \text{head} (f\ i) \rightarrow \text{head} (f (i + 1))$ 
proof
  fix k
  from not-fin
  have  $\forall i < (\text{Suc } k). \neg \text{final} (\text{head} (f\ i))$ 
    by simp

  from infinite-computation-extract-head-Seq [OF step f-0 this]
  show  $\Gamma \vdash \text{head} (f\ k) \rightarrow \text{head} (f (k + 1))$  by simp
qed
with head-f-0 have  $\Gamma \vdash (c_1, s) \rightarrow \dots(\infty)$ 
  by (auto simp add: inf-def)
thus ?thesis
  by simp
qed
qed

lemma split-inf-Catch:
assumes inf-comp:  $\Gamma \vdash (\text{Catch } c_1\ c_2, s) \rightarrow \dots(\infty)$ 
shows  $\Gamma \vdash (c_1, s) \rightarrow \dots(\infty) \vee$ 
   $(\exists s'. \Gamma \vdash (c_1, s) \rightarrow^* (\text{Throw}, \text{Normal } s') \wedge \Gamma \vdash (c_2, \text{Normal } s') \rightarrow \dots(\infty))$ 
proof –
from inf-comp obtain f where
  step:  $\forall i :: \text{nat}. \Gamma \vdash f\ i \rightarrow f (i + 1)$  and
  f-0:  $f\ 0 = (\text{Catch } c_1\ c_2, s)$ 

```

```

  by (auto simp add: inf-def)
from f-0 have head-f-0: head (f 0) = (c1,s)
  by (simp add: head-def head-com-def)
show ?thesis
proof (cases  $\exists i. \text{final } (\text{head } (f i))$ )
  case True
  define k where k = (LEAST i. final (head (f i)))
  have less-k:  $\forall i < k. \neg \text{final } (\text{head } (f i))$ 
    apply (intro allI impI)
    apply (unfold k-def)
    apply (drule not-less-Least)
    apply auto
  done
from infinite-computation-extract-head-Catch [OF step f-0 this]
obtain step-head:  $\forall i < k. \Gamma \vdash \text{head } (f i) \rightarrow \text{head } (f (i + 1))$  and
  conf:  $\forall i < k. (\exists c' s'. f (i + 1) = (\text{Catch } c' c_2, s'))$ 
  by blast
from True
have final-f-k: final (head (f k))
  apply -
  apply (erule exE)
  apply (drule LeastI)
  apply (simp add: k-def)
  done
moreover
from f-0 conf [rule-format, of k - 1]
obtain c' s' where f-k: f k = (Catch c' c2, s')
  by (cases k) auto
moreover
from step-head have steps-head:  $\Gamma \vdash \text{head } (f 0) \rightarrow^* \text{head } (f k)$ 
proof (induct k)
  case 0 thus ?case by simp
next
  case (Suc m)
  have step:  $\forall i < \text{Suc } m. \Gamma \vdash \text{head } (f i) \rightarrow \text{head } (f (i + 1))$  by fact
  hence  $\forall i < m. \Gamma \vdash \text{head } (f i) \rightarrow \text{head } (f (i + 1))$ 
    by auto
  hence  $\Gamma \vdash \text{head } (f 0) \rightarrow^* \text{head } (f m)$ 
    by (rule Suc.hyps)
  also from step [rule-format, of m]
  have  $\Gamma \vdash \text{head } (f m) \rightarrow \text{head } (f (m + 1))$  by simp
  finally show ?case by simp
qed
{
  assume f-k: f k = (Catch Skip c2, s')
  with steps-head
  have  $\Gamma \vdash (c_1, s) \rightarrow^* (\text{Skip}, s')$ 
    using head-f-0
    by (simp add: head-def head-com-def)

```

```

moreover
from step [rule-format, of k] f-k
obtain  $\Gamma \vdash (\text{Catch Skip } c_2, s') \rightarrow (\text{Skip}, s')$  and
  f-Suc-k:  $f (k + 1) = (\text{Skip}, s')$ 
  by (fastforce elim: step.cases intro: step.intros)
from step [rule-format, of k+1, simplified f-Suc-k]
have ?thesis
  by (rule no-step-final') (auto simp add: final-def)
}
moreover
{
  fix x
  assume s': s'=Normal x and f-k:  $f k = (\text{Catch Throw } c_2, s')$ 
  with steps-head
  have  $\Gamma \vdash (c_1, s) \rightarrow^* (\text{Throw}, s')$ 
    using head-f-0
    by (simp add: head-def head-com-def)
  moreover
  from step [rule-format, of k] f-k s'
  obtain  $\Gamma \vdash (\text{Catch Throw } c_2, s') \rightarrow (c_2, s')$  and
    f-Suc-k:  $f (k + 1) = (c_2, s')$ 
    by (fastforce elim: step-elim-cases intro: step.intros)
  define g where  $g i = f (i + (k + 1))$  for i
  from f-Suc-k
  have g-0:  $g 0 = (c_2, s')$ 
    by (simp add: g-def)
  from step
  have  $\forall i. \Gamma \vdash g i \rightarrow g (i + 1)$ 
    by (simp add: g-def)
  with g-0 have  $\Gamma \vdash (c_2, s') \rightarrow \dots(\infty)$ 
    by (auto simp add: inf-def)
  ultimately
  have ?thesis
    using s'
    by auto
}
ultimately
show ?thesis
  by (auto simp add: final-def head-def head-com-def)
next
case False
then have not-fin:  $\forall i. \neg \text{final } (\text{head } (f i))$ 
  by blast
have  $\forall i. \Gamma \vdash \text{head } (f i) \rightarrow \text{head } (f (i + 1))$ 
proof
  fix k
  from not-fin
  have  $\forall i < (\text{Suc } k). \neg \text{final } (\text{head } (f i))$ 
    by simp

```

```

    from infinite-computation-extract-head-Catch [OF step f-0 this ]
    show  $\Gamma \vdash \text{head } (f\ k) \rightarrow \text{head } (f\ (k + 1))$  by simp
  qed
  with head-f-0 have  $\Gamma \vdash (c_1, s) \rightarrow \dots(\infty)$ 
  by (auto simp add: inf-def)
  thus ?thesis
  by simp
qed
qed

lemma Skip-no-step:  $\Gamma \vdash (\text{Skip}, s) \rightarrow \text{cfg} \implies P$ 
  apply (erule no-step-final^)
  apply (simp add: final-def)
  done

lemma not-inf-Stuck:  $\neg \Gamma \vdash (c, \text{Stuck}) \rightarrow \dots(\infty)$ 
proof (induct c)
  case Skip
  show ?case
  proof (rule not-infI)
    fix f
    assume f-step:  $\bigwedge i. \Gamma \vdash f\ i \rightarrow f\ (\text{Suc } i)$ 
    assume f-0:  $f\ 0 = (\text{Skip}, \text{Stuck})$ 
    from f-step [of 0] f-0
    show False
    by (auto elim: Skip-no-step)
  qed
next
  case (Basic g)
  show ?case
  proof (rule not-infI)
    fix f
    assume f-step:  $\bigwedge i. \Gamma \vdash f\ i \rightarrow f\ (\text{Suc } i)$ 
    assume f-0:  $f\ 0 = (\text{Basic } g, \text{Stuck})$ 
    from f-step [of 0] f-0 f-step [of 1]
    show False
    by (fastforce elim: Skip-no-step step-elim-cases)
  qed
next
  case (Spec r)
  show ?case
  proof (rule not-infI)
    fix f
    assume f-step:  $\bigwedge i. \Gamma \vdash f\ i \rightarrow f\ (\text{Suc } i)$ 
    assume f-0:  $f\ 0 = (\text{Spec } r, \text{Stuck})$ 
    from f-step [of 0] f-0 f-step [of 1]
    show False
    by (fastforce elim: Skip-no-step step-elim-cases)
  qed

```

```

qed
next
case (Seq c1 c2)
show ?case
proof
  assume  $\Gamma \vdash (\text{Seq } c_1 \ c_2, \text{Stuck}) \rightarrow \dots(\infty)$ 
  from split-inf-Seq [OF this] Seq.hyps
  show False
  by (auto dest: steps-Stuck-prop)
qed
next
case (Cond b c1 c2)
show ?case
proof (rule not-infI)
  fix f
  assume f-step:  $\bigwedge i. \Gamma \vdash f \ i \rightarrow f \ (\text{Suc } i)$ 
  assume f-0:  $f \ 0 = (\text{Cond } b \ c_1 \ c_2, \text{Stuck})$ 
  from f-step [of 0] f-0 f-step [of 1]
  show False
  by (fastforce elim: Skip-no-step step-elim-cases)
qed
next
case (While b c)
show ?case
proof (rule not-infI)
  fix f
  assume f-step:  $\bigwedge i. \Gamma \vdash f \ i \rightarrow f \ (\text{Suc } i)$ 
  assume f-0:  $f \ 0 = (\text{While } b \ c, \text{Stuck})$ 
  from f-step [of 0] f-0 f-step [of 1]
  show False
  by (fastforce elim: Skip-no-step step-elim-cases)
qed
next
case (Call p)
show ?case
proof (rule not-infI)
  fix f
  assume f-step:  $\bigwedge i. \Gamma \vdash f \ i \rightarrow f \ (\text{Suc } i)$ 
  assume f-0:  $f \ 0 = (\text{Call } p, \text{Stuck})$ 
  from f-step [of 0] f-0 f-step [of 1]
  show False
  by (fastforce elim: Skip-no-step step-elim-cases)
qed
next
case (DynCom d)
show ?case
proof (rule not-infI)
  fix f
  assume f-step:  $\bigwedge i. \Gamma \vdash f \ i \rightarrow f \ (\text{Suc } i)$ 

```

```

    assume f-0: f 0 = (DynCom d, Stuck)
    from f-step [of 0] f-0 f-step [of 1]
    show False
      by (fastforce elim: Skip-no-step step-elim-cases)
  qed
next
case (Guard m g c)
show ?case
proof (rule not-infI)
  fix f
  assume f-step:  $\bigwedge i. \Gamma \vdash f i \rightarrow f (Suc i)$ 
  assume f-0: f 0 = (Guard m g c, Stuck)
  from f-step [of 0] f-0 f-step [of 1]
  show False
    by (fastforce elim: Skip-no-step step-elim-cases)
  qed
next
case Throw
show ?case
proof (rule not-infI)
  fix f
  assume f-step:  $\bigwedge i. \Gamma \vdash f i \rightarrow f (Suc i)$ 
  assume f-0: f 0 = (Throw, Stuck)
  from f-step [of 0] f-0 f-step [of 1]
  show False
    by (fastforce elim: Skip-no-step step-elim-cases)
  qed
next
case (Catch c1 c2)
show ?case
proof
  assume  $\Gamma \vdash (Catch c_1 c_2, Stuck) \rightarrow \dots(\infty)$ 
  from split-inf-Catch [OF this] Catch.hyps
  show False
    by (auto dest: steps-Stuck-prop)
  qed
qed

lemma not-inf-Fault:  $\neg \Gamma \vdash (c, Fault x) \rightarrow \dots(\infty)$ 
proof (induct c)
  case Skip
  show ?case
  proof (rule not-infI)
    fix f
    assume f-step:  $\bigwedge i. \Gamma \vdash f i \rightarrow f (Suc i)$ 
    assume f-0: f 0 = (Skip, Fault x)
    from f-step [of 0] f-0
    show False
      by (auto elim: Skip-no-step)
  
```

```

qed
next
  case (Basic g)
  thus ?case
  proof (rule not-infI)
    fix f
    assume f-step:  $\bigwedge i. \Gamma \vdash f\ i \rightarrow f\ (Suc\ i)$ 
    assume f-0:  $f\ 0 = (Basic\ g, Fault\ x)$ 
    from f-step [of 0] f-0 f-step [of 1]
    show False
    by (fastforce elim: Skip-no-step step-elim-cases)
  qed
next
  case (Spec r)
  thus ?case
  proof (rule not-infI)
    fix f
    assume f-step:  $\bigwedge i. \Gamma \vdash f\ i \rightarrow f\ (Suc\ i)$ 
    assume f-0:  $f\ 0 = (Spec\ r, Fault\ x)$ 
    from f-step [of 0] f-0 f-step [of 1]
    show False
    by (fastforce elim: Skip-no-step step-elim-cases)
  qed
next
  case (Seq c1 c2)
  show ?case
  proof
    assume  $\Gamma \vdash (Seq\ c_1\ c_2, Fault\ x) \rightarrow \dots(\infty)$ 
    from split-inf-Seq [OF this] Seq.hyps
    show False
    by (auto dest: steps-Fault-prop)
  qed
next
  case (Cond b c1 c2)
  show ?case
  proof (rule not-infI)
    fix f
    assume f-step:  $\bigwedge i. \Gamma \vdash f\ i \rightarrow f\ (Suc\ i)$ 
    assume f-0:  $f\ 0 = (Cond\ b\ c_1\ c_2, Fault\ x)$ 
    from f-step [of 0] f-0 f-step [of 1]
    show False
    by (fastforce elim: Skip-no-step step-elim-cases)
  qed
next
  case (While b c)
  show ?case
  proof (rule not-infI)
    fix f
    assume f-step:  $\bigwedge i. \Gamma \vdash f\ i \rightarrow f\ (Suc\ i)$ 

```



```

    assume f-0: f 0 = (While b c, Fault x)
    from f-step [of 0] f-0 f-step [of 1]
    show False
      by (fastforce elim: Skip-no-step step-elim-cases)
  qed
next
case (Call p)
show ?case
proof (rule not-infI)
  fix f
  assume f-step:  $\bigwedge i. \Gamma \vdash f i \rightarrow f (Suc i)$ 
  assume f-0: f 0 = (Call p, Fault x)
  from f-step [of 0] f-0 f-step [of 1]
  show False
    by (fastforce elim: Skip-no-step step-elim-cases)
  qed
next
case (DynCom d)
show ?case
proof (rule not-infI)
  fix f
  assume f-step:  $\bigwedge i. \Gamma \vdash f i \rightarrow f (Suc i)$ 
  assume f-0: f 0 = (DynCom d, Fault x)
  from f-step [of 0] f-0 f-step [of 1]
  show False
    by (fastforce elim: Skip-no-step step-elim-cases)
  qed
next
case (Guard m g c)
show ?case
proof (rule not-infI)
  fix f
  assume f-step:  $\bigwedge i. \Gamma \vdash f i \rightarrow f (Suc i)$ 
  assume f-0: f 0 = (Guard m g c, Fault x)
  from f-step [of 0] f-0 f-step [of 1]
  show False
    by (fastforce elim: Skip-no-step step-elim-cases)
  qed
next
case Throw
show ?case
proof (rule not-infI)
  fix f
  assume f-step:  $\bigwedge i. \Gamma \vdash f i \rightarrow f (Suc i)$ 
  assume f-0: f 0 = (Throw, Fault x)
  from f-step [of 0] f-0 f-step [of 1]
  show False
    by (fastforce elim: Skip-no-step step-elim-cases)
  qed
qed

```

```

next
  case (Catch c1 c2)
  show ?case
  proof
    assume  $\Gamma \vdash (\text{Catch } c_1 \ c_2, \text{Fault } x) \rightarrow \dots(\infty)$ 
    from split-inf-Catch [OF this] Catch.hyps
    show False
    by (auto dest: steps-Fault-prop)
  qed
qed

lemma not-inf-Abrupt:  $\neg \Gamma \vdash (c, \text{Abrupt } s) \rightarrow \dots(\infty)$ 
proof (induct c)
  case Skip
  show ?case
  proof (rule not-infI)
    fix f
    assume f-step:  $\bigwedge i. \Gamma \vdash f \ i \rightarrow f \ (\text{Suc } i)$ 
    assume f-0:  $f \ 0 = (\text{Skip}, \text{Abrupt } s)$ 
    from f-step [of 0] f-0
    show False
    by (auto elim: Skip-no-step)
  qed
next
  case (Basic g)
  thus ?case
  proof (rule not-infI)
    fix f
    assume f-step:  $\bigwedge i. \Gamma \vdash f \ i \rightarrow f \ (\text{Suc } i)$ 
    assume f-0:  $f \ 0 = (\text{Basic } g, \text{Abrupt } s)$ 
    from f-step [of 0] f-0 f-step [of 1]
    show False
    by (fastforce elim: Skip-no-step step-elim-cases)
  qed
next
  case (Spec r)
  thus ?case
  proof (rule not-infI)
    fix f
    assume f-step:  $\bigwedge i. \Gamma \vdash f \ i \rightarrow f \ (\text{Suc } i)$ 
    assume f-0:  $f \ 0 = (\text{Spec } r, \text{Abrupt } s)$ 
    from f-step [of 0] f-0 f-step [of 1]
    show False
    by (fastforce elim: Skip-no-step step-elim-cases)
  qed
next
  case (Seq c1 c2)
  show ?case
  proof

```

```

    assume  $\Gamma \vdash (\text{Seq } c_1 \ c_2, \text{Abrupt } s) \rightarrow \dots(\infty)$ 
    from split-inf-Seq [OF this] Seq.hyps
    show False
      by (auto dest: steps-Abrupt-prop)
  qed
next
  case (Cond  $b \ c_1 \ c_2$ )
  show ?case
  proof (rule not-infI)
    fix  $f$ 
    assume  $f\text{-step}: \bigwedge i. \Gamma \vdash f \ i \rightarrow f \ (\text{Suc } i)$ 
    assume  $f\text{-0}: f \ 0 = (\text{Cond } b \ c_1 \ c_2, \text{Abrupt } s)$ 
    from  $f\text{-step}$  [of 0]  $f\text{-0}$   $f\text{-step}$  [of 1]
    show False
      by (fastforce elim: Skip-no-step step-elim-cases)
  qed
next
  case (While  $b \ c$ )
  show ?case
  proof (rule not-infI)
    fix  $f$ 
    assume  $f\text{-step}: \bigwedge i. \Gamma \vdash f \ i \rightarrow f \ (\text{Suc } i)$ 
    assume  $f\text{-0}: f \ 0 = (\text{While } b \ c, \text{Abrupt } s)$ 
    from  $f\text{-step}$  [of 0]  $f\text{-0}$   $f\text{-step}$  [of 1]
    show False
      by (fastforce elim: Skip-no-step step-elim-cases)
  qed
next
  case (Call  $p$ )
  show ?case
  proof (rule not-infI)
    fix  $f$ 
    assume  $f\text{-step}: \bigwedge i. \Gamma \vdash f \ i \rightarrow f \ (\text{Suc } i)$ 
    assume  $f\text{-0}: f \ 0 = (\text{Call } p, \text{Abrupt } s)$ 
    from  $f\text{-step}$  [of 0]  $f\text{-0}$   $f\text{-step}$  [of 1]
    show False
      by (fastforce elim: Skip-no-step step-elim-cases)
  qed
next
  case (DynCom  $d$ )
  show ?case
  proof (rule not-infI)
    fix  $f$ 
    assume  $f\text{-step}: \bigwedge i. \Gamma \vdash f \ i \rightarrow f \ (\text{Suc } i)$ 
    assume  $f\text{-0}: f \ 0 = (\text{DynCom } d, \text{Abrupt } s)$ 
    from  $f\text{-step}$  [of 0]  $f\text{-0}$   $f\text{-step}$  [of 1]
    show False
      by (fastforce elim: Skip-no-step step-elim-cases)
  qed

```

```

next
  case (Guard m g c)
  show ?case
  proof (rule not-infI)
    fix f
    assume f-step:  $\bigwedge i. \Gamma \vdash f i \rightarrow f (Suc i)$ 
    assume f-0:  $f 0 = (Guard\ m\ g\ c, Abrupt\ s)$ 
    from f-step [of 0] f-0 f-step [of 1]
    show False
    by (fastforce elim: Skip-no-step step-elim-cases)
  qed
next
  case Throw
  show ?case
  proof (rule not-infI)
    fix f
    assume f-step:  $\bigwedge i. \Gamma \vdash f i \rightarrow f (Suc i)$ 
    assume f-0:  $f 0 = (Throw, Abrupt\ s)$ 
    from f-step [of 0] f-0 f-step [of 1]
    show False
    by (fastforce elim: Skip-no-step step-elim-cases)
  qed
next
  case (Catch c1 c2)
  show ?case
  proof
    assume  $\Gamma \vdash (Catch\ c_1\ c_2, Abrupt\ s) \rightarrow \dots(\infty)$ 
    from split-inf-Catch [OF this] Catch.hyps
    show False
    by (auto dest: steps-Abrupt-prop)
  qed
qed

```

**theorem** *terminates-impl-no-infinite-computation:*

```

  assumes termi:  $\Gamma \vdash c \downarrow s$ 
  shows  $\neg \Gamma \vdash (c, s) \rightarrow \dots(\infty)$ 
using termi
proof (induct)
  case (Skip s) thus ?case
  proof (rule not-infI)
    fix f
    assume f-step:  $\bigwedge i. \Gamma \vdash f i \rightarrow f (Suc i)$ 
    assume f-0:  $f 0 = (Skip, Normal\ s)$ 
    from f-step [of 0] f-0
    show False
    by (auto elim: Skip-no-step)
  qed
next

```

```

case (Basic g s)
thus ?case
proof (rule not-infI)
  fix f
  assume f-step:  $\bigwedge i. \Gamma \vdash f\ i \rightarrow f\ (Suc\ i)$ 
  assume f-0:  $f\ 0 = (Basic\ g, Normal\ s)$ 
  from f-step [of 0] f-0 f-step [of 1]
  show False
    by (fastforce elim: Skip-no-step step-elim-cases)
qed
next
case (Spec r s)
thus ?case
proof (rule not-infI)
  fix f
  assume f-step:  $\bigwedge i. \Gamma \vdash f\ i \rightarrow f\ (Suc\ i)$ 
  assume f-0:  $f\ 0 = (Spec\ r, Normal\ s)$ 
  from f-step [of 0] f-0 f-step [of 1]
  show False
    by (fastforce elim: Skip-no-step step-elim-cases)
qed
next
case (Guard s g c m)
have g:  $s \in g$  by fact
have hyp:  $\neg \Gamma \vdash (c, Normal\ s) \rightarrow \dots(\infty)$  by fact
show ?case
proof (rule not-infI)
  fix f
  assume f-step:  $\bigwedge i. \Gamma \vdash f\ i \rightarrow f\ (Suc\ i)$ 
  assume f-0:  $f\ 0 = (Guard\ m\ g\ c, Normal\ s)$ 
  from f-step [of 0] f-0 g
  have f 1 =  $(c, Normal\ s)$ 
    by (fastforce elim: step-elim-cases)
  with f-step
  have  $\Gamma \vdash (c, Normal\ s) \rightarrow \dots(\infty)$ 
    apply (simp add: inf-def)
    apply (rule-tac x= $\lambda i. f\ (Suc\ i)$  in exI)
    by simp
  with hyp show False ..
qed
next
case (GuardFault s g m c)
have g:  $s \notin g$  by fact
show ?case
proof (rule not-infI)
  fix f
  assume f-step:  $\bigwedge i. \Gamma \vdash f\ i \rightarrow f\ (Suc\ i)$ 
  assume f-0:  $f\ 0 = (Guard\ m\ g\ c, Normal\ s)$ 
  from g f-step [of 0] f-0 f-step [of 1]

```

```

    show False
      by (fastforce elim: Skip-no-step step-elim-cases)
  qed
next
  case (Fault c m)
  thus ?case
    by (rule not-inf-Fault)
next
  case (Seq c1 s c2)
  show ?case
  proof
    assume  $\Gamma \vdash (\text{Seq } c_1 \ c_2, \text{Normal } s) \rightarrow \dots(\infty)$ 
    from split-inf-Seq [OF this] Seq.hyps
    show False
      by (auto intro: steps-Skip-impl-exec)
  qed
next
  case (CondTrue s b c1 c2)
  have  $b: s \in b$  by fact
  have hyp-c1:  $\neg \Gamma \vdash (c_1, \text{Normal } s) \rightarrow \dots(\infty)$  by fact
  show ?case
  proof (rule not-infI)
    fix f
    assume f-step:  $\bigwedge i. \Gamma \vdash f \ i \rightarrow f \ (\text{Suc } i)$ 
    assume f-0:  $f \ 0 = (\text{Cond } b \ c_1 \ c_2, \text{Normal } s)$ 
    from b f-step [of 0] f-0
    have  $f \ 1 = (c_1, \text{Normal } s)$ 
      by (auto elim: step-Normal-elim-cases)
    with f-step
    have  $\Gamma \vdash (c_1, \text{Normal } s) \rightarrow \dots(\infty)$ 
      apply (simp add: inf-def)
      apply (rule-tac x= $\lambda i. f \ (\text{Suc } i)$  in exI)
      by simp
    with hyp-c1 show False by simp
  qed
next
  case (CondFalse s b c2 c1)
  have  $b: s \notin b$  by fact
  have hyp-c2:  $\neg \Gamma \vdash (c_2, \text{Normal } s) \rightarrow \dots(\infty)$  by fact
  show ?case
  proof (rule not-infI)
    fix f
    assume f-step:  $\bigwedge i. \Gamma \vdash f \ i \rightarrow f \ (\text{Suc } i)$ 
    assume f-0:  $f \ 0 = (\text{Cond } b \ c_1 \ c_2, \text{Normal } s)$ 
    from b f-step [of 0] f-0
    have  $f \ 1 = (c_2, \text{Normal } s)$ 
      by (auto elim: step-Normal-elim-cases)
    with f-step
    have  $\Gamma \vdash (c_2, \text{Normal } s) \rightarrow \dots(\infty)$ 

```

**apply** (*simp add: inf-def*)  
**apply** (*rule-tac x= $\lambda i$ . f (Suc i) in exI*)  
**by simp**  
**with hyp-c2 show False by simp**  
**qed**  
**next**  
**case** (*WhileTrue s b c*)  
**have**  $b: s \in b$  **by fact**  
**have** *hyp-c*:  $\neg \Gamma \vdash (c, \text{Normal } s) \rightarrow \dots(\infty)$  **by fact**  
**have** *hyp-w*:  $\forall s'. \Gamma \vdash \langle c, \text{Normal } s \rangle \Rightarrow s' \longrightarrow$   
 $\Gamma \vdash \text{While } b \ c \ \downarrow \ s' \wedge \neg \Gamma \vdash (\text{While } b \ c, \ s') \rightarrow \dots(\infty)$  **by fact**  
**have** *not-inf-Seq*:  $\neg \Gamma \vdash (\text{Seq } c \ (\text{While } b \ c), \ \text{Normal } s) \rightarrow \dots(\infty)$   
**proof**  
**assume**  $\Gamma \vdash (\text{Seq } c \ (\text{While } b \ c), \ \text{Normal } s) \rightarrow \dots(\infty)$   
**from** *split-inf-Seq [OF this] hyp-c hyp-w* **show False**  
**by** (*auto intro: steps-Skip-impl-exec*)  
**qed**  
**show** *?case*  
**proof**  
**assume**  $\Gamma \vdash (\text{While } b \ c, \ \text{Normal } s) \rightarrow \dots(\infty)$   
**then obtain f where**  
*f-step*:  $\bigwedge i. \Gamma \vdash f \ i \rightarrow f \ (\text{Suc } i)$  **and**  
*f-0*:  $f \ 0 = (\text{While } b \ c, \ \text{Normal } s)$   
**by** (*auto simp add: inf-def*)  
**from** *f-step [of 0]* *f-0* **b**  
**have**  $f \ 1 = (\text{Seq } c \ (\text{While } b \ c), \ \text{Normal } s)$   
**by** (*auto elim: step-Normal-elim-cases*)  
**with** *f-step*  
**have**  $\Gamma \vdash (\text{Seq } c \ (\text{While } b \ c), \ \text{Normal } s) \rightarrow \dots(\infty)$   
**apply** (*simp add: inf-def*)  
**apply** (*rule-tac x= $\lambda i$ . f (Suc i) in exI*)  
**by simp**  
**with not-inf-Seq show False by simp**  
**qed**  
**next**  
**case** (*WhileFalse s b c*)  
**have**  $b: s \notin b$  **by fact**  
**show** *?case*  
**proof** (*rule not-infI*)  
**fix**  $f$   
**assume** *f-step*:  $\bigwedge i. \Gamma \vdash f \ i \rightarrow f \ (\text{Suc } i)$   
**assume** *f-0*:  $f \ 0 = (\text{While } b \ c, \ \text{Normal } s)$   
**from** *b f-step [of 0] f-0 f-step [of 1]*  
**show False**  
**by** (*fastforce elim: Skip-no-step step-elim-cases*)  
**qed**  
**next**  
**case** (*Call p bdy s*)  
**have**  $bdy: \Gamma \ p = \text{Some } bdy$  **by fact**

```

have hyp:  $\neg \Gamma \vdash (bdy, Normal\ s) \rightarrow \dots(\infty)$  by fact
show ?case
proof (rule not-infI)
  fix f
  assume f-step:  $\bigwedge i. \Gamma \vdash f\ i \rightarrow f\ (Suc\ i)$ 
  assume f-0:  $f\ 0 = (Call\ p, Normal\ s)$ 
  from bdy f-step [of 0] f-0
  have f 1 = (bdy, Normal s)
    by (auto elim: step-Normal-elim-cases)
  with f-step
  have  $\Gamma \vdash (bdy, Normal\ s) \rightarrow \dots(\infty)$ 
  apply (simp add: inf-def)
  apply (rule-tac x= $\lambda i. f\ (Suc\ i)$  in exI)
  by simp
with hyp show False by simp
qed
next
case (CallUndefined p s)
have no-bdy:  $\Gamma\ p = None$  by fact
show ?case
proof (rule not-infI)
  fix f
  assume f-step:  $\bigwedge i. \Gamma \vdash f\ i \rightarrow f\ (Suc\ i)$ 
  assume f-0:  $f\ 0 = (Call\ p, Normal\ s)$ 
  from no-bdy f-step [of 0] f-0 f-step [of 1]
  show False
  by (fastforce elim: Skip-no-step step-elim-cases)
qed
next
case (Stuck c)
show ?case
  by (rule not-inf-Stuck)
next
case (DynCom c s)
have hyp:  $\neg \Gamma \vdash (c\ s, Normal\ s) \rightarrow \dots(\infty)$  by fact
show ?case
proof (rule not-infI)
  fix f
  assume f-step:  $\bigwedge i. \Gamma \vdash f\ i \rightarrow f\ (Suc\ i)$ 
  assume f-0:  $f\ 0 = (DynCom\ c, Normal\ s)$ 
  from f-step [of 0] f-0
  have  $f\ (Suc\ 0) = (c\ s, Normal\ s)$ 
    by (auto elim: step-elim-cases)
  with f-step have  $\Gamma \vdash (c\ s, Normal\ s) \rightarrow \dots(\infty)$ 
  apply (simp add: inf-def)
  apply (rule-tac x= $\lambda i. f\ (Suc\ i)$  in exI)
  by simp
with hyp
show False by simp

```



```

qed
next
  case (Throw s) thus ?case
  proof (rule not-infI)
    fix f
    assume f-step:  $\bigwedge i. \Gamma \vdash i \rightarrow f \text{ (Suc } i)$ 
    assume f-0:  $f \ 0 = (\text{Throw}, \text{Normal } s)$ 
    from f-step [of 0] f-0
    show False
    by (auto elim: step-elim-cases)
  qed
next
  case (Abrupt c)
  show ?case
  by (rule not-inf-Abrupt)
next
  case (Catch c1 s c2)
  show ?case
  proof
    assume  $\Gamma \vdash (\text{Catch } c_1 \ c_2, \text{Normal } s) \rightarrow \dots(\infty)$ 
    from split-inf-Catch [OF this] Catch.hyps
    show False
    by (auto intro: steps-Throw-impl-exec)
  qed
qed

```

### definition

*termi-call-steps* ::  $(\text{'s}, \text{'p}, \text{'f}) \text{ body} \Rightarrow ((\text{'s} \times \text{'p}) \times (\text{'s} \times \text{'p})) \text{set}$   
**where**  
*termi-call-steps*  $\Gamma =$   
 $\{((t, q), (s, p)). \Gamma \vdash \text{Call } p \downarrow \text{Normal } s \wedge$   
 $(\exists c. \Gamma \vdash (\text{Call } p, \text{Normal } s) \rightarrow^+ (c, \text{Normal } t) \wedge \text{redex } c = \text{Call } q)\}$

**primrec** *subst-redex* ::  $(\text{'s}, \text{'p}, \text{'f}) \text{com} \Rightarrow (\text{'s}, \text{'p}, \text{'f}) \text{com} \Rightarrow (\text{'s}, \text{'p}, \text{'f}) \text{com}$

### where

```

subst-redex Skip c = c |
subst-redex (Basic f) c = c |
subst-redex (Spec r) c = c |
subst-redex (Seq c1 c2) c = Seq (subst-redex c1 c) c2 |
subst-redex (Cond b c1 c2) c = c |
subst-redex (While b c') c = c |
subst-redex (Call p) c = c |
subst-redex (DynCom d) c = c |
subst-redex (Guard f b c') c = c |
subst-redex (Throw) c = c |
subst-redex (Catch c1 c2) c = Catch (subst-redex c1 c) c2

```

**lemma** *subst-redex-redex*:

*subst-redex c (redex c) = c*

**by** (*induct c*) *auto*

**lemma** *redex-subst-redex*: *redex (subst-redex c r) = redex r*

**by** (*induct c*) *auto*

**lemma** *step-redex'*:

**shows**  $\Gamma \vdash (\text{redex } c, s) \rightarrow (r', s') \implies \Gamma \vdash (c, s) \rightarrow (\text{subst-redex } c \ r', s')$

**by** (*induct c*) (*auto intro: step.Seq step.Catch*)

**lemma** *step-redex*:

**shows**  $\Gamma \vdash (r, s) \rightarrow (r', s') \implies \Gamma \vdash (\text{subst-redex } c \ r, s) \rightarrow (\text{subst-redex } c \ r', s')$

**by** (*induct c*) (*auto intro: step.Seq step.Catch*)

**lemma** *steps-redex*:

**assumes** *steps*:  $\Gamma \vdash (r, s) \rightarrow^* (r', s')$

**shows**  $\bigwedge c. \Gamma \vdash (\text{subst-redex } c \ r, s) \rightarrow^* (\text{subst-redex } c \ r', s')$

**using** *steps*

**proof** (*induct rule: converse-rtranclp-induct2 [case-names Refl Trans]*)

**case** *Refl*

**show**  $\Gamma \vdash (\text{subst-redex } c \ r', s') \rightarrow^* (\text{subst-redex } c \ r', s')$

**by** *simp*

**next**

**case** (*Trans r s r'' s''*)

**have**  $\Gamma \vdash (r, s) \rightarrow (r'', s'')$  **by** *fact*

**from** *step-redex [OF this]*

**have**  $\Gamma \vdash (\text{subst-redex } c \ r, s) \rightarrow (\text{subst-redex } c \ r'', s'')$ .

**also**

**have**  $\Gamma \vdash (\text{subst-redex } c \ r'', s'') \rightarrow^* (\text{subst-redex } c \ r', s')$  **by** *fact*

**finally show** *?case* .

**qed**

**ML** <

*ML-Thms.bind-thm (trancl-induct2, Split-Rule.split-rule @ {context}*

*(Rule-Insts.read-instantiate @ {context}*

*[(((a, 0), Position.none), (aa, ab)), (((b, 0), Position.none), (ba, bb))] []*

*@ {thm trancl-induct}));*

>

**lemma** *steps-redex'*:

**assumes** *steps*:  $\Gamma \vdash (r, s) \rightarrow^+ (r', s')$

**shows**  $\bigwedge c. \Gamma \vdash (\text{subst-redex } c \ r, s) \rightarrow^+ (\text{subst-redex } c \ r', s')$

**using** *steps*

**proof** (*induct rule: tranclp-induct2 [consumes 1, case-names Step Trans]*)

**case** (*Step r' s'*)

**have**  $\Gamma \vdash (r, s) \rightarrow (r', s')$  **by** *fact*

**then have**  $\Gamma \vdash (\text{subst-redex } c \ r, s) \rightarrow (\text{subst-redex } c \ r', s')$

by (rule step-redex)  
 then show  $\Gamma \vdash (\text{subst-redex } c \ r, \ s) \rightarrow^+ (\text{subst-redex } c \ r', \ s')$ .  
 next  
 case (Trans  $r' \ s' \ r'' \ s''$ )  
 have  $\Gamma \vdash (\text{subst-redex } c \ r, \ s) \rightarrow^+ (\text{subst-redex } c \ r', \ s')$  by fact  
 also  
 have  $\Gamma \vdash (r', \ s') \rightarrow (r'', \ s'')$  by fact  
 hence  $\Gamma \vdash (\text{subst-redex } c \ r', \ s') \rightarrow (\text{subst-redex } c \ r'', \ s'')$   
 by (rule step-redex)  
 finally show  $\Gamma \vdash (\text{subst-redex } c \ r, \ s) \rightarrow^+ (\text{subst-redex } c \ r'', \ s'')$ .  
 qed

primrec seq:: (nat  $\Rightarrow$  ('s,'p,'f)com)  $\Rightarrow$  'p  $\Rightarrow$  nat  $\Rightarrow$  ('s,'p,'f)com  
 where  
 seq c p 0 = Call p |  
 seq c p (Suc i) = subst-redex (seq c p i) (c i)

lemma renumber':  
 assumes f:  $\forall i. (a, f \ i) \in r^* \wedge (f \ i, f \ (\text{Suc } i)) \in r$   
 assumes a-b:  $(a, b) \in r^*$   
 shows  $b = f \ 0 \implies (\exists f. f \ 0 = a \wedge (\forall i. (f \ i, f \ (\text{Suc } i)) \in r))$   
 using a-b  
 proof (induct rule: converse-rtrancl-induct [consumes 1])  
 assume  $b = f \ 0$   
 with f show  $\exists f. f \ 0 = b \wedge (\forall i. (f \ i, f \ (\text{Suc } i)) \in r)$   
 by blast  
 next  
 fix a z  
 assume a-z:  $(a, z) \in r$  and  $(z, b) \in r^*$   
 assume  $b = f \ 0 \implies \exists f. f \ 0 = z \wedge (\forall i. (f \ i, f \ (\text{Suc } i)) \in r)$   
 $b = f \ 0$   
 then obtain f where  $f \ 0 = z$  and seq:  $\forall i. (f \ i, f \ (\text{Suc } i)) \in r$   
 by iprover  
 {  
 fix i have  $((\lambda i. \text{case } i \ \text{of } 0 \Rightarrow a \mid \text{Suc } i \Rightarrow f \ i) \ i, f \ i) \in r$   
 using seq a-z f0  
 by (cases i) auto  
 }  
 then  
 show  $\exists f. f \ 0 = a \wedge (\forall i. (f \ i, f \ (\text{Suc } i)) \in r)$   
 by - (rule exI [where x= $\lambda i. \text{case } i \ \text{of } 0 \Rightarrow a \mid \text{Suc } i \Rightarrow f \ i$ ], simp)  
 qed

lemma renumber:  
 $\forall i. (a, f \ i) \in r^* \wedge (f \ i, f \ (\text{Suc } i)) \in r$   
 $\implies \exists f. f \ 0 = a \wedge (\forall i. (f \ i, f \ (\text{Suc } i)) \in r)$   
 by (blast dest:renumber')

**lemma** *lem*:  
 $\forall y. r^{++} a y \longrightarrow P a \longrightarrow P y$   
 $\implies ((b,a) \in \{(y,x). P x \wedge r x y\}^+) = ((b,a) \in \{(y,x). P x \wedge r^{++} x y\})$

**apply** (*rule iffI*)  
**apply** *clarify*  
**apply** (*erule trancl-induct*)  
**apply** *blast*  
**apply** (*blast intro:tranclp-trans*)  
**apply** *clarify*  
**apply** (*erule tranclp-induct*)  
**apply** *blast*  
**apply** (*blast intro:trancl-trans*)  
**done**

**corollary** *terminates-impl-no-infinite-trans-computation*:  
**assumes** *terminates*:  $\Gamma \vdash c \downarrow s$   
**shows**  $\neg(\exists f. f 0 = (c,s) \wedge (\forall i. \Gamma \vdash f i \rightarrow^+ f (Suc i)))$

**proof** –  
**have**  $wf(\{(y,x). \Gamma \vdash (c,s) \rightarrow^* x \wedge \Gamma \vdash x \rightarrow y\}^+)$   
**proof** (*rule wf-trancl*)  
**show**  $wf \{(y, x). \Gamma \vdash (c,s) \rightarrow^* x \wedge \Gamma \vdash x \rightarrow y\}$   
**proof** (*simp only: wf-iff-no-infinite-down-chain,clarify,simp*)  
**fix** *f*  
**assume**  $\forall i. \Gamma \vdash (c,s) \rightarrow^* f i \wedge \Gamma \vdash f i \rightarrow f (Suc i)$   
**hence**  $\exists f. f (0::nat) = (c,s) \wedge (\forall i. \Gamma \vdash f i \rightarrow f (Suc i))$   
**by** (*rule renumber [to-pred]*)  
**moreover from** *terminates-impl-no-infinite-computation [OF terminates]*  
**have**  $\neg(\exists f. f (0::nat) = (c, s) \wedge (\forall i. \Gamma \vdash f i \rightarrow f (Suc i)))$   
**by** (*simp add: inf-def*)  
**ultimately show** *False*  
**by** *simp*  
**qed**

**qed**  
**hence**  $\neg(\exists f. \forall i. (f (Suc i), f i) \in \{(y, x). \Gamma \vdash (c, s) \rightarrow^* x \wedge \Gamma \vdash x \rightarrow y\}^+)$   
**by** (*simp add: wf-iff-no-infinite-down-chain*)  
**thus** *?thesis*  
**proof** (*rule contrapos-nn*)  
**assume**  $\exists f. f (0::nat) = (c, s) \wedge (\forall i. \Gamma \vdash f i \rightarrow^+ f (Suc i))$   
**then obtain** *f where*  
 $f0: f 0 = (c, s)$  **and**  
 $seq: \forall i. \Gamma \vdash f i \rightarrow^+ f (Suc i)$   
**by** *iprover*  
**show**  
 $\exists f. \forall i. (f (Suc i), f i) \in \{(y, x). \Gamma \vdash (c, s) \rightarrow^* x \wedge \Gamma \vdash x \rightarrow y\}^+$   
**proof** (*rule exI [where x=f],rule allI*)  
**fix** *i*  
**show**  $(f (Suc i), f i) \in \{(y, x). \Gamma \vdash (c, s) \rightarrow^* x \wedge \Gamma \vdash x \rightarrow y\}^+$   
**proof** –

```

{
  fix i have  $\Gamma \vdash (c, s) \rightarrow^* f i$ 
  proof (induct i)
    case 0 show  $\Gamma \vdash (c, s) \rightarrow^* f 0$ 
      by (simp add: f0)
    next
      case (Suc n)
      have  $\Gamma \vdash (c, s) \rightarrow^* f n$  by fact
      with seq show  $\Gamma \vdash (c, s) \rightarrow^* f (Suc n)$ 
        by (blast intro: tranclp-into-rtranclp rtranclp-trans)
      qed
    }
  hence  $\Gamma \vdash (c, s) \rightarrow^* f i$ 
    by iprover
  with seq have
     $(f (Suc i), f i) \in \{(y, x). \Gamma \vdash (c, s) \rightarrow^* x \wedge \Gamma \vdash x \rightarrow^+ y\}$ 
    by clarsimp
  moreover
  have  $\forall y. \Gamma \vdash f i \rightarrow^+ y \longrightarrow \Gamma \vdash (c, s) \rightarrow^* f i \longrightarrow \Gamma \vdash (c, s) \rightarrow^* y$ 
    by (blast intro: tranclp-into-rtranclp rtranclp-trans)
  ultimately
  show ?thesis
    by (subst lem )
  qed
qed
qed
qed
qed

```

**theorem** *wf-termi-call-steps*: *wf (termi-call-steps  $\Gamma$ )*

**proof** (*simp only: termi-call-steps-def wf-iff-no-infinite-down-chain, clarify, simp*)

**fix** *f*

**assume** *inf*:  $\forall i. (\lambda(t, q) (s, p).$

$\Gamma \vdash \text{Call } p \downarrow \text{Normal } s \wedge$

$(\exists c. \Gamma \vdash (\text{Call } p, \text{Normal } s) \rightarrow^+ (c, \text{Normal } t) \wedge \text{redex } c = \text{Call } q))$

$(f (Suc i)) (f i)$

**define** *s* **where**  $s i = \text{fst } (f i)$  **for**  $i :: \text{nat}$

**define** *p* **where**  $p i = (\text{snd } (f i) :: 'b)$  **for**  $i :: \text{nat}$

**from** *inf*

**have** *inf'*:  $\forall i. \Gamma \vdash \text{Call } (p i) \downarrow \text{Normal } (s i) \wedge$

$(\exists c. \Gamma \vdash (\text{Call } (p i), \text{Normal } (s i)) \rightarrow^+ (c, \text{Normal } (s (i+1)))) \wedge$

$\text{redex } c = \text{Call } (p (i+1))$

**apply** –

**apply** (*rule allI*)

**apply** (*erule-tac x=i in allE*)

**apply** (*auto simp add: s-def p-def*)

**done**

**show** *False*

**proof** –

**from** *inf'*  
**have**  $\exists c. \forall i. \Gamma \vdash \text{Call } (p \ i) \downarrow \text{Normal } (s \ i) \wedge$   
 $\Gamma \vdash (\text{Call } (p \ i), \text{Normal } (s \ i)) \rightarrow^+ (c \ i, \text{Normal } (s \ (i+1))) \wedge$   
 $\text{redex } (c \ i) = \text{Call } (p \ (i+1))$   
**apply** –  
**apply** (*rule choice*)  
**by** *blast*  
**then obtain** *c* **where**  
*termi-c*:  $\forall i. \Gamma \vdash \text{Call } (p \ i) \downarrow \text{Normal } (s \ i)$  **and**  
*steps-c*:  $\forall i. \Gamma \vdash (\text{Call } (p \ i), \text{Normal } (s \ i)) \rightarrow^+ (c \ i, \text{Normal } (s \ (i+1)))$  **and**  
*red-c*:  $\forall i. \text{redex } (c \ i) = \text{Call } (p \ (i+1))$   
**by** *auto*  
**define** *g* **where**  $g \ i = (\text{seq } c \ (p \ 0) \ i, \text{Normal } (s \ i)::('a, 'c) \ xstate)$  **for** *i*  
**from** *red-c* [*rule-format, of 0*]  
**have**  $g \ 0 = (\text{Call } (p \ 0), \text{Normal } (s \ 0))$   
**by** (*simp add: g-def*)  
**moreover**  
{  
**fix** *i*  
**have**  $\text{redex } (\text{seq } c \ (p \ 0) \ i) = \text{Call } (p \ i)$   
**by** (*induct i*) (*auto simp add: redex-subst-redex red-c*)  
**from** *this* [*symmetric*]  
**have**  $\text{subst-redex } (\text{seq } c \ (p \ 0) \ i) (\text{Call } (p \ i)) = (\text{seq } c \ (p \ 0) \ i)$   
**by** (*simp add: subst-redex-redex*)  
} **note** *subst-redex-seq = this*  
**have**  $\forall i. \Gamma \vdash (g \ i) \rightarrow^+ (g \ (i+1))$   
**proof**  
**fix** *i*  
**from** *steps-c* [*rule-format, of i*]  
**have**  $\Gamma \vdash (\text{Call } (p \ i), \text{Normal } (s \ i)) \rightarrow^+ (c \ i, \text{Normal } (s \ (i + 1)))$ .  
**from** *steps-redex'* [*OF this, of (seq c (p 0) i)*]  
**have**  $\Gamma \vdash (\text{subst-redex } (\text{seq } c \ (p \ 0) \ i) (\text{Call } (p \ i)), \text{Normal } (s \ i)) \rightarrow^+$   
 $(\text{subst-redex } (\text{seq } c \ (p \ 0) \ i) (c \ i), \text{Normal } (s \ (i + 1)))$  .  
**hence**  $\Gamma \vdash (\text{seq } c \ (p \ 0) \ i, \text{Normal } (s \ i)) \rightarrow^+$   
 $(\text{seq } c \ (p \ 0) \ (i+1), \text{Normal } (s \ (i + 1)))$   
**by** (*simp add: subst-redex-seq*)  
**thus**  $\Gamma \vdash (g \ i) \rightarrow^+ (g \ (i+1))$   
**by** (*simp add: g-def*)  
**qed**  
**moreover**  
**from** *terminates-impl-no-infinite-trans-computation* [*OF termi-c* [*rule-format, of 0*]]  
**have**  $\neg (\exists f. f \ 0 = (\text{Call } (p \ 0), \text{Normal } (s \ 0)) \wedge (\forall i. \Gamma \vdash f \ i \rightarrow^+ f \ (\text{Suc } i)))$  .  
**ultimately show** *False*  
**by** *auto*  
**qed**  
**qed**

**lemma** *no-infinite-computation-implies-wf*:  
**assumes** *not-inf*:  $\neg \Gamma \vdash (c, s) \rightarrow \dots(\infty)$   
**shows** *wf*  $\{(c2, c1). \Gamma \vdash (c, s) \rightarrow^* c1 \wedge \Gamma \vdash c1 \rightarrow c2\}$   
**proof** (*simp only: wf-iff-no-infinite-down-chain, clarify, simp*)  
**fix** *f*  
**assume**  $\forall i. \Gamma \vdash (c, s) \rightarrow^* f i \wedge \Gamma \vdash f i \rightarrow f (Suc i)$   
**hence**  $\exists f. f 0 = (c, s) \wedge (\forall i. \Gamma \vdash f i \rightarrow f (Suc i))$   
**by** (*rule renumber [to-pred]*)  
**moreover from** *not-inf*  
**have**  $\neg (\exists f. f 0 = (c, s) \wedge (\forall i. \Gamma \vdash f i \rightarrow f (Suc i)))$   
**by** (*simp add: inf-def*)  
**ultimately show** *False*  
**by** *simp*  
**qed**

**lemma** *not-final-Stuck-step*:  $\neg final (c, Stuck) \implies \exists c' s'. \Gamma \vdash (c, Stuck) \rightarrow (c', s')$   
**by** (*induct c*) (*fastforce intro: step.intros simp add: final-def*) $+$

**lemma** *not-final-Abrupt-step*:  
 $\neg final (c, Abrupt s) \implies \exists c' s'. \Gamma \vdash (c, Abrupt s) \rightarrow (c', s')$   
**by** (*induct c*) (*fastforce intro: step.intros simp add: final-def*) $+$

**lemma** *not-final-Fault-step*:  
 $\neg final (c, Fault f) \implies \exists c' s'. \Gamma \vdash (c, Fault f) \rightarrow (c', s')$   
**by** (*induct c*) (*fastforce intro: step.intros simp add: final-def*) $+$

**lemma** *not-final-Normal-step*:  
 $\neg final (c, Normal s) \implies \exists c' s'. \Gamma \vdash (c, Normal s) \rightarrow (c', s')$   
**proof** (*induct c*)  
**case** *Skip* **thus** *?case* **by** (*fastforce intro: step.intros simp add: final-def*)  
**next**  
**case** *Basic* **thus** *?case* **by** (*fastforce intro: step.intros*)  
**next**  
**case** (*Spec r*)  
**thus** *?case*  
**by** (*cases*  $\exists t. (s, t) \in r$ ) (*fastforce intro: step.intros*) $+$   
**next**  
**case** (*Seq c<sub>1</sub> c<sub>2</sub>*)  
**thus** *?case*  
**by** (*cases final (c<sub>1</sub>, Normal s)*) (*fastforce intro: step.intros simp add: final-def*) $+$   
**next**  
**case** (*Cond b c<sub>1</sub> c<sub>2</sub>*)  
**show** *?case*  
**by** (*cases s*  $\in b$ ) (*fastforce intro: step.intros*) $+$   
**next**  
**case** (*While b c*)  
**show** *?case*  
**by** (*cases s*  $\in b$ ) (*fastforce intro: step.intros*) $+$   
**next**

```

    case (Call p)
    show ?case
    by (cases  $\Gamma$  p) (fastforce intro: step.intros)+
next
    case DynCom thus ?case by (fastforce intro: step.intros)
next
    case (Guard f g c)
    show ?case
    by (cases  $s \in g$ ) (fastforce intro: step.intros)+
next
    case Throw
    thus ?case by (fastforce intro: step.intros simp add: final-def)
next
    case (Catch c1 c2)
    thus ?case
    by (cases final (c1, Normal s)) (fastforce intro: step.intros simp add: final-def)+
qed

```

**lemma** *final-termi*:  
 $final (c, s) \implies \Gamma \vdash c \downarrow s$   
 by (cases s) (auto simp add: final-def terminates.intros)

**lemma** *split-computation*:  
 assumes  $steps: \Gamma \vdash (c, s) \rightarrow^* (c_f, s_f)$   
 assumes *not-final*:  $\neg final (c, s)$   
 assumes *final*:  $final (c_f, s_f)$   
 shows  $\exists c' s'. \Gamma \vdash (c, s) \rightarrow (c', s') \wedge \Gamma \vdash (c', s') \rightarrow^* (c_f, s_f)$   
 using *steps not-final final*  
**proof** (*induct rule: converse-rtranclp-induct2 [case-names Refl Trans]*)  
 case *Refl* thus ?case by *simp*  
next  
 case (*Trans* c s c' s')  
 thus ?case by *auto*  
qed

**lemma** *wf-implies-termi-reach-step-case*:  
 assumes *hyp*:  $\bigwedge c' s'. \Gamma \vdash (c, Normal s) \rightarrow (c', s') \implies \Gamma \vdash c' \downarrow s'$   
 shows  $\Gamma \vdash c \downarrow Normal s$   
 using *hyp*  
**proof** (*induct c*)  
 case *Skip* show ?case by (fastforce intro: terminates.intros)  
next  
 case *Basic* show ?case by (fastforce intro: terminates.intros)  
next  
 case (*Spec* r)  
 show ?case  
 by (cases  $\exists t. (s, t) \in r$ ) (fastforce intro: terminates.intros)+  
next



```

case (Seq c1 c2)
have hyp:  $\bigwedge c' s'. \Gamma \vdash (\text{Seq } c_1 \ c_2, \text{Normal } s) \rightarrow (c', s') \implies \Gamma \vdash c' \downarrow s'$  by fact
show ?case
proof (rule terminates.Seq)
  {
    fix c' s'
    assume step-c1:  $\Gamma \vdash (c_1, \text{Normal } s) \rightarrow (c', s')$ 
    have  $\Gamma \vdash c' \downarrow s'$ 
    proof -
      from step-c1
      have  $\Gamma \vdash (\text{Seq } c_1 \ c_2, \text{Normal } s) \rightarrow (\text{Seq } c' \ c_2, s')$ 
        by (rule step.Seq)
      from hyp [OF this]
      have  $\Gamma \vdash \text{Seq } c' \ c_2 \downarrow s'$ .
      thus  $\Gamma \vdash c' \downarrow s'$ 
        by cases auto
    qed
  }
from Seq.hyps (1) [OF this]
show  $\Gamma \vdash c_1 \downarrow \text{Normal } s$ .
next
show  $\forall s'. \Gamma \vdash \langle c_1, \text{Normal } s \rangle \Rightarrow s' \longrightarrow \Gamma \vdash c_2 \downarrow s'$ 
proof (intro allI impI)
  fix s'
  assume exec-c1:  $\Gamma \vdash \langle c_1, \text{Normal } s \rangle \Rightarrow s'$ 
  show  $\Gamma \vdash c_2 \downarrow s'$ 
  proof (cases final (c1, Normal s))
    case True
    hence c1=Skip  $\vee$  c1=Throw
      by (simp add: final-def)
    thus ?thesis
  proof
    assume Skip: c1=Skip
    have  $\Gamma \vdash (\text{Seq } \text{Skip } \ c_2, \text{Normal } s) \rightarrow (c_2, \text{Normal } s)$ 
      by (rule step.SeqSkip)
    from hyp [simplified Skip, OF this]
    have  $\Gamma \vdash c_2 \downarrow \text{Normal } s$  .
    moreover from exec-c1 Skip
    have s'=Normal s
      by (auto elim: exec-Normal-elim-cases)
    ultimately show ?thesis by simp
  next
  assume Throw: c1=Throw
  with exec-c1 have s'=Abrupt s
    by (auto elim: exec-Normal-elim-cases)
  thus ?thesis
    by auto
  qed
next

```

**case** *False*  
**from** *exec-impl-steps* [*OF exec-c1*]  
**obtain**  $c_f t$  **where**  
 $steps-c_1: \Gamma \vdash (c_1, Normal\ s) \rightarrow^* (c_f, t)$  **and**  
 $fin: (case\ s'\ of$   
 $\quad Abrupt\ x \Rightarrow c_f = Throw \wedge t = Normal\ x$   
 $\quad | - \Rightarrow c_f = Skip \wedge t = s')$   
**by** (*fastforce split: xstate.splits*)  
**with** *fin* **have** *final: final* ( $c_f, t$ )  
**by** (*cases s'*) (*auto simp add: final-def*)  
**from** *split-computation* [*OF steps-c1 False this*]  
**obtain**  $c'' s''$  **where**  
 $first: \Gamma \vdash (c_1, Normal\ s) \rightarrow (c'', s'')$  **and**  
 $rest: \Gamma \vdash (c'', s'') \rightarrow^* (c_f, t)$   
**by** *blast*  
**from** *step.Seq* [*OF first*]  
**have**  $\Gamma \vdash (Seq\ c_1\ c_2, Normal\ s) \rightarrow (Seq\ c''\ c_2, s'')$ .  
**from** *hyp* [*OF this*]  
**have** *termi-s'':*  $\Gamma \vdash Seq\ c''\ c_2 \downarrow s''$ .  
**show** *?thesis*  
**proof** (*cases s''*)  
**case** (*Normal x*)  
**from** *termi-s''* [*simplified Normal*]  
**have** *termi-c2:*  $\forall t. \Gamma \vdash \langle c'', Normal\ x \rangle \Rightarrow t \longrightarrow \Gamma \vdash c_2 \downarrow t$   
**by** *cases*  
**show** *?thesis*  
**proof** (*cases*  $\exists x'. s' = Abrupt\ x'$ )  
**case** *False*  
**with** *fin* **obtain**  $c_f = Skip\ t = s'$   
**by** (*cases s'*) *auto*  
**from** *steps-Skip-impl-exec* [*OF rest [simplified this] Normal*]  
**have**  $\Gamma \vdash \langle c'', Normal\ x \rangle \Rightarrow s'$   
**by** *simp*  
**from** *termi-c2* [*rule-format, OF this*]  
**show**  $\Gamma \vdash c_2 \downarrow s'$ .  
**next**  
**case** *True*  
**with** *fin* **obtain**  $x'$  **where**  $s': s' = Abrupt\ x'$  **and**  $c_f = Throw\ t = Normal\ x'$   
**by** *auto*  
**from** *steps-Throw-impl-exec* [*OF rest [simplified this] Normal*]  
**have**  $\Gamma \vdash \langle c'', Normal\ x \rangle \Rightarrow Abrupt\ x'$   
**by** *simp*  
**from** *termi-c2* [*rule-format, OF this*]  $s'$   
**show**  $\Gamma \vdash c_2 \downarrow s'$  **by** *simp*  
**qed**  
**next**  
**case** (*Abrupt x*)  
**from** *steps-Abrupt-prop* [*OF rest this*]  
**have**  $t = Abrupt\ x$  **by** *simp*

```

    with fin have  $s' = \text{Abrupt } x$ 
      by (cases  $s^\wedge$ ) auto
    thus  $\Gamma \vdash c_2 \downarrow s'$ 
      by auto
  next
    case (Fault  $f$ )
    from steps-Fault-prop [OF rest this]
    have  $t = \text{Fault } f$  by simp
    with fin have  $s' = \text{Fault } f$ 
      by (cases  $s^\wedge$ ) auto
    thus  $\Gamma \vdash c_2 \downarrow s'$ 
      by auto
  next
    case Stuck
    from steps-Stuck-prop [OF rest this]
    have  $t = \text{Stuck}$  by simp
    with fin have  $s' = \text{Stuck}$ 
      by (cases  $s^\wedge$ ) auto
    thus  $\Gamma \vdash c_2 \downarrow s'$ 
      by auto
  qed
qed
qed
qed
next
  case (Cond  $b$   $c_1$   $c_2$ )
  have hyp:  $\bigwedge c' s'. \Gamma \vdash (\text{Cond } b \ c_1 \ c_2, \text{Normal } s) \rightarrow (c', s') \implies \Gamma \vdash c' \downarrow s'$  by fact
  show ?case
  proof (cases  $s \in b$ )
    case True
    then have  $\Gamma \vdash (\text{Cond } b \ c_1 \ c_2, \text{Normal } s) \rightarrow (c_1, \text{Normal } s)$ 
      by (rule step.CondTrue)
    from hyp [OF this] have  $\Gamma \vdash c_1 \downarrow \text{Normal } s$ .
    with True show ?thesis
      by (auto intro: terminates.intros)
  next
    case False
    then have  $\Gamma \vdash (\text{Cond } b \ c_1 \ c_2, \text{Normal } s) \rightarrow (c_2, \text{Normal } s)$ 
      by (rule step.CondFalse)
    from hyp [OF this] have  $\Gamma \vdash c_2 \downarrow \text{Normal } s$ .
    with False show ?thesis
      by (auto intro: terminates.intros)
  qed
next
  case (While  $b$   $c$ )
  have hyp:  $\bigwedge c' s'. \Gamma \vdash (\text{While } b \ c, \text{Normal } s) \rightarrow (c', s') \implies \Gamma \vdash c' \downarrow s'$  by fact
  show ?case
  proof (cases  $s \in b$ )
    case True

```

```

then have  $\Gamma \vdash (\text{While } b \ c, \text{Normal } s) \rightarrow (\text{Seq } c \ (\text{While } b \ c), \text{Normal } s)$ 
  by (rule step.WhileTrue)
from hyp [OF this] have  $\Gamma \vdash (\text{Seq } c \ (\text{While } b \ c)) \downarrow \text{Normal } s.$ 
with True show ?thesis
  by (auto elim: terminates-Normal-elim-cases intro: terminates.intros)
next
  case False
  thus ?thesis
    by (auto intro: terminates.intros)
  qed
next
  case (Call p)
  have hyp:  $\bigwedge c' \ s'. \Gamma \vdash (\text{Call } p, \text{Normal } s) \rightarrow (c', s') \implies \Gamma \vdash c' \downarrow s'$  by fact
  show ?case
  proof (cases  $\Gamma \ p$ )
    case None
    thus ?thesis
      by (auto intro: terminates.intros)
    next
    case (Some bdy)
    then have  $\Gamma \vdash (\text{Call } p, \text{Normal } s) \rightarrow (\text{bdy}, \text{Normal } s)$ 
      by (rule step.Call)
    from hyp [OF this] have  $\Gamma \vdash \text{bdy} \downarrow \text{Normal } s.$ 
    with Some show ?thesis
      by (auto intro: terminates.intros)
    qed
  next
  case (DynCom c)
  have hyp:  $\bigwedge c' \ s'. \Gamma \vdash (\text{DynCom } c, \text{Normal } s) \rightarrow (c', s') \implies \Gamma \vdash c' \downarrow s'$  by fact
  have  $\Gamma \vdash (\text{DynCom } c, \text{Normal } s) \rightarrow (c \ s, \text{Normal } s)$ 
    by (rule step.DynCom)
  from hyp [OF this] have  $\Gamma \vdash c \ s \downarrow \text{Normal } s.$ 
  then show ?case
    by (auto intro: terminates.intros)
  next
  case (Guard f g c)
  have hyp:  $\bigwedge c' \ s'. \Gamma \vdash (\text{Guard } f \ g \ c, \text{Normal } s) \rightarrow (c', s') \implies \Gamma \vdash c' \downarrow s'$  by fact
  show ?case
  proof (cases  $s \in g$ )
    case True
    then have  $\Gamma \vdash (\text{Guard } f \ g \ c, \text{Normal } s) \rightarrow (c, \text{Normal } s)$ 
      by (rule step.Guard)
    from hyp [OF this] have  $\Gamma \vdash c \downarrow \text{Normal } s.$ 
    with True show ?thesis
      by (auto intro: terminates.intros)
    next
    case False
    thus ?thesis
      by (auto intro: terminates.intros)

```

```

qed
next
  case Throw show ?case by (auto intro: terminates.intros)
next
  case (Catch  $c_1$   $c_2$ )
  have hyp:  $\bigwedge c' s'. \Gamma \vdash (\text{Catch } c_1 \ c_2, \text{Normal } s) \rightarrow (c', s') \implies \Gamma \vdash c' \downarrow s'$  by fact
  show ?case
  proof (rule terminates.Catch)
    {
      fix  $c' s'$ 
      assume step-c1:  $\Gamma \vdash (c_1, \text{Normal } s) \rightarrow (c', s')$ 
      have  $\Gamma \vdash c' \downarrow s'$ 
      proof –
        from step-c1
        have  $\Gamma \vdash (\text{Catch } c_1 \ c_2, \text{Normal } s) \rightarrow (\text{Catch } c' \ c_2, s')$ 
          by (rule step.Catch)
        from hyp [OF this]
        have  $\Gamma \vdash \text{Catch } c' \ c_2 \downarrow s'$ .
        thus  $\Gamma \vdash c' \downarrow s'$ 
          by cases auto
      qed
    }
  from Catch.hyps (1) [OF this]
  show  $\Gamma \vdash c_1 \downarrow \text{Normal } s$ .
next
  show  $\forall s'. \Gamma \vdash \langle c_1, \text{Normal } s \rangle \implies \text{Abrupt } s' \longrightarrow \Gamma \vdash c_2 \downarrow \text{Normal } s'$ 
  proof (intro allI impI)
    fix  $s'$ 
    assume exec-c1:  $\Gamma \vdash \langle c_1, \text{Normal } s \rangle \implies \text{Abrupt } s'$ 
    show  $\Gamma \vdash c_2 \downarrow \text{Normal } s'$ 
    proof (cases final ( $c_1, \text{Normal } s$ ))
      case True
      with exec-c1
      have Throw:  $c_1 = \text{Throw}$ 
        by (auto simp add: final-def elim: exec-Normal-elim-cases)
      have  $\Gamma \vdash (\text{Catch } \text{Throw } \ c_2, \text{Normal } s) \rightarrow (c_2, \text{Normal } s)$ 
        by (rule step.CatchThrow)
      from hyp [simplified Throw, OF this]
      have  $\Gamma \vdash c_2 \downarrow \text{Normal } s$ .
      moreover from exec-c1 Throw
      have  $s' = s$ 
        by (auto elim: exec-Normal-elim-cases)
      ultimately show ?thesis by simp
    next
    case False
    from exec-impl-steps [OF exec-c1]
    obtain  $c_f \ t$  where
      steps-c1:  $\Gamma \vdash (c_1, \text{Normal } s) \rightarrow^* (\text{Throw}, \text{Normal } s')$ 
      by (fastforce split: xstate.splits)

```

```

from split-computation [OF steps-c1 False]
obtain  $c'' s''$  where
  first:  $\Gamma \vdash (c_1, \text{Normal } s) \rightarrow (c'', s'')$  and
  rest:  $\Gamma \vdash (c'', s'') \rightarrow^* (\text{Throw}, \text{Normal } s')$ 
  by (auto simp add: final-def)
from step.Catch [OF first]
have  $\Gamma \vdash (\text{Catch } c_1 c_2, \text{Normal } s) \rightarrow (\text{Catch } c'' c_2, s'')$ .
from hyp [OF this]
have  $\Gamma \vdash \text{Catch } c'' c_2 \downarrow s''$ .
moreover
from steps-Throw-impl-exec [OF rest]
have  $\Gamma \vdash \langle c'', s'' \rangle \Rightarrow \text{Abrupt } s'$ .
moreover
from rest obtain  $x$  where  $s'' = \text{Normal } x$ 
  by (cases s'')
  (auto dest: steps-Fault-prop steps-Abrupt-prop steps-Stuck-prop)
ultimately show ?thesis
  by (fastforce elim: terminates-elim-cases)
qed
qed
qed
qed

```

```

lemma wf-implies-termi-reach:
assumes wf:  $wf \{(cfg2, cfg1). \Gamma \vdash (c, s) \rightarrow^* cfg1 \wedge \Gamma \vdash cfg1 \rightarrow cfg2\}$ 
shows  $\bigwedge c1 s1. [\Gamma \vdash (c, s) \rightarrow^* cfg1; \text{cfg1} = (c1, s1)] \Longrightarrow \Gamma \vdash c1 \downarrow s1$ 
using wf
proof (induct cfg1, simp)
  fix  $c1 s1$ 
  assume reach:  $\Gamma \vdash (c, s) \rightarrow^* (c1, s1)$ 
  assume hyp-raw:  $\bigwedge y c2 s2. [\Gamma \vdash (c1, s1) \rightarrow (c2, s2); \Gamma \vdash (c, s) \rightarrow^* (c2, s2); y = (c2, s2)] \Longrightarrow \Gamma \vdash c2 \downarrow s2$ 
have hyp:  $\bigwedge c2 s2. \Gamma \vdash (c1, s1) \rightarrow (c2, s2) \Longrightarrow \Gamma \vdash c2 \downarrow s2$ 
  apply –
  apply (rule hyp-raw)
  apply assumption
  using reach
  apply simp
  apply (rule refl)
  done

show  $\Gamma \vdash c1 \downarrow s1$ 
proof (cases s1)
  case (Normal s1')
  with wf-implies-termi-reach-step-case [OF hyp [simplified Normal]]
  show ?thesis
  by auto
qed (auto intro: terminates.intros)

```

qed

**theorem** *no-infinite-computation-impl-terminates*:

assumes *not-inf*:  $\neg \Gamma \vdash (c, s) \rightarrow \dots(\infty)$

shows  $\Gamma \vdash c \downarrow s$

**proof** –

from *no-infinite-computation-implies-wf* [*OF not-inf*]

have *wf*:  $wf \{(c2, c1). \Gamma \vdash (c, s) \rightarrow^* c1 \wedge \Gamma \vdash c1 \rightarrow c2\}$ .

show *?thesis*

by (*rule wf-implies-termi-reach* [*OF wf*]) *auto*

qed

**corollary** *terminates-iff-no-infinite-computation*:

$\Gamma \vdash c \downarrow s = (\neg \Gamma \vdash (c, s) \rightarrow \dots(\infty))$

**apply** (*rule*)

**apply** (*erule terminates-impl-no-infinite-computation*)

**apply** (*erule no-infinite-computation-impl-terminates*)

**done**

## 8.6 Generalised Redexes

For an important lemma for the completeness proof of the Hoare-logic for total correctness we need a generalisation of *redex* that not only yield the redex itself but all the enclosing statements as well.

**primrec** *redexes*::  $(\prime s, \prime p, \prime f)com \Rightarrow (\prime s, \prime p, \prime f)com\ set$

**where**

*redexes* *Skip* =  $\{Skip\}$  |

*redexes* (*Basic f*) =  $\{Basic\ f\}$  |

*redexes* (*Spec r*) =  $\{Spec\ r\}$  |

*redexes* (*Seq c<sub>1</sub> c<sub>2</sub>*) =  $\{Seq\ c_1\ c_2\} \cup redexes\ c_1$  |

*redexes* (*Cond b c<sub>1</sub> c<sub>2</sub>*) =  $\{Cond\ b\ c_1\ c_2\}$  |

*redexes* (*While b c*) =  $\{While\ b\ c\}$  |

*redexes* (*Call p*) =  $\{Call\ p\}$  |

*redexes* (*DynCom d*) =  $\{DynCom\ d\}$  |

*redexes* (*Guard f b c*) =  $\{Guard\ f\ b\ c\}$  |

*redexes* (*Throw*) =  $\{Throw\}$  |

*redexes* (*Catch c<sub>1</sub> c<sub>2</sub>*) =  $\{Catch\ c_1\ c_2\} \cup redexes\ c_1$

**lemma** *root-in-redexes*:  $c \in redexes\ c$

**apply** (*induct c*)

**apply** *auto*

**done**

**lemma** *redex-in-redexes*:  $redex\ c \in redexes\ c$

**apply** (*induct c*)

**apply** *auto*

**done**

**lemma** *redex-redexes*:  $\bigwedge c'. \llbracket c' \in redexes\ c; redex\ c' = c \rrbracket \Longrightarrow redex\ c = c'$

```

apply (induct c)
apply auto
done

lemma step-redexes:
  shows  $\bigwedge r r'. \llbracket \Gamma \vdash (r, s) \rightarrow (r', s') \rrbracket; r \in \text{redexes } c \rrbracket$ 
   $\implies \exists c'. \Gamma \vdash (c, s) \rightarrow (c', s') \wedge r' \in \text{redexes } c'$ 
proof (induct c)
  case Skip thus ?case by (fastforce intro: step.intros elim: step-elim-cases)
next
  case Basic thus ?case by (fastforce intro: step.intros elim: step-elim-cases)
next
  case Spec thus ?case by (fastforce intro: step.intros elim: step-elim-cases)
next
  case (Seq c1 c2)
  have  $r \in \text{redexes } (\text{Seq } c_1 c_2)$  by fact
  hence  $r: r = \text{Seq } c_1 c_2 \vee r \in \text{redexes } c_1$ 
  by simp
  have step-r:  $\Gamma \vdash (r, s) \rightarrow (r', s')$  by fact
  from r show ?case
  proof
    assume  $r = \text{Seq } c_1 c_2$ 
    with step-r
    show ?case
    by (auto simp add: root-in-redexes)
  next
    assume  $r: r \in \text{redexes } c_1$ 
    from Seq.hyps (1) [OF step-r this]
    obtain c' where
      step-c1:  $\Gamma \vdash (c_1, s) \rightarrow (c', s')$  and
       $r': r' \in \text{redexes } c'$ 
    by blast
    from step.Seq [OF step-c1]
    have  $\Gamma \vdash (\text{Seq } c_1 c_2, s) \rightarrow (\text{Seq } c' c_2, s')$ .
    with r'
    show ?case
    by auto
  qed
next
  case Cond
  thus ?case
  by (fastforce intro: step.intros elim: step-elim-cases simp add: root-in-redexes)
next
  case While
  thus ?case
  by (fastforce intro: step.intros elim: step-elim-cases simp add: root-in-redexes)
next
  case Call thus ?case
  by (fastforce intro: step.intros elim: step-elim-cases simp add: root-in-redexes)

```



```

next
  case DynCom thus ?case
    by (fastforce intro: step.intros elim: step-elim-cases simp add: root-in-redexes)
next
  case Guard thus ?case
    by (fastforce intro: step.intros elim: step-elim-cases simp add: root-in-redexes)
next
  case Throw thus ?case
    by (fastforce intro: step.intros elim: step-elim-cases simp add: root-in-redexes)
next
  case (Catch c1 c2)
  have  $r \in \text{redexes } (\text{Catch } c_1 c_2)$  by fact
  hence  $r: r = \text{Catch } c_1 c_2 \vee r \in \text{redexes } c_1$ 
    by simp
  have step-r:  $\Gamma \vdash (r, s) \rightarrow (r', s')$  by fact
  from r show ?case
  proof
    assume  $r = \text{Catch } c_1 c_2$ 
    with step-r
    show ?case
      by (auto simp add: root-in-redexes)
  next
    assume  $r: r \in \text{redexes } c_1$ 
    from Catch.hyps (1) [OF step-r this]
    obtain c' where
      step-c1:  $\Gamma \vdash (c_1, s) \rightarrow (c', s')$  and
       $r': r' \in \text{redexes } c'$ 
      by blast
    from step.Catch [OF step-c1]
    have  $\Gamma \vdash (\text{Catch } c_1 c_2, s) \rightarrow (\text{Catch } c' c_2, s')$ .
    with  $r'$ 
    show ?case
      by auto
  qed
qed

lemma steps-redexes:
  assumes steps:  $\Gamma \vdash (r, s) \rightarrow^* (r', s')$ 
  shows  $\bigwedge c. r \in \text{redexes } c \implies \exists c'. \Gamma \vdash (c, s) \rightarrow^* (c', s') \wedge r' \in \text{redexes } c'$ 
using steps
proof (induct rule: converse-rtranclp-induct2 [case-names Refl Trans])
  case Refl
  then
  show  $\exists c'. \Gamma \vdash (c, s) \rightarrow^* (c', s') \wedge r' \in \text{redexes } c'$ 
    by auto
  next
  case (Trans r s r'' s'')
  have  $\Gamma \vdash (r, s) \rightarrow (r'', s'')$   $r \in \text{redexes } c$  by fact+
  from step-redexes [OF this]

```

**obtain**  $c'$  **where**  
*step*:  $\Gamma \vdash (c, s) \rightarrow (c', s'')$  **and**  
 $r''$ :  $r'' \in \text{redexes } c'$   
**by** *blast*  
**note** *step*  
**also**  
**from** *Trans.hyps* ( $\exists$ ) [*OF*  $r''$ ]  
**obtain**  $c''$  **where**  
*steps*:  $\Gamma \vdash (c', s'') \rightarrow^* (c'', s')$  **and**  
 $r'$ :  $r' \in \text{redexes } c''$   
**by** *blast*  
**note** *steps*  
**finally**  
**show** *?case*  
**using**  $r'$   
**by** *blast*  
**qed**

**lemma** *steps-redexes'*:  
**assumes** *steps*:  $\Gamma \vdash (r, s) \rightarrow^+ (r', s')$   
**shows**  $\bigwedge c. r \in \text{redexes } c \implies \exists c'. \Gamma \vdash (c, s) \rightarrow^+ (c', s') \wedge r' \in \text{redexes } c'$   
**using** *steps*  
**proof** (*induct rule: tranclp-induct2* [*consumes 1, case-names Step Trans*])  
**case** (*Step*  $r' s' c'$ )  
**have**  $\Gamma \vdash (r, s) \rightarrow (r', s') \wedge r' \in \text{redexes } c'$  **by** *fact+*  
**from** *step-redexes* [*OF this*]  
**show** *?case*  
**by** (*blast intro: r-into-trancl*)  
**next**  
**case** (*Trans*  $r' s' r'' s''$ )  
**from** *Trans* **obtain**  $c'$  **where**  
*steps*:  $\Gamma \vdash (c, s) \rightarrow^+ (c', s')$  **and**  
 $r'$ :  $r' \in \text{redexes } c'$   
**by** *blast*  
**note** *steps*  
**moreover**  
**have**  $\Gamma \vdash (r', s') \rightarrow (r'', s'')$  **by** *fact*  
**from** *step-redexes* [*OF this*  $r'$ ] **obtain**  $c''$  **where**  
*step*:  $\Gamma \vdash (c', s') \rightarrow (c'', s'')$  **and**  
 $r''$ :  $r'' \in \text{redexes } c''$   
**by** *blast*  
**note** *step*  
**finally** **show** *?case*  
**using**  $r''$  **by** *blast*  
**qed**

**lemma** *step-redexes-Seq*:

**assumes** *step*:  $\Gamma \vdash (r, s) \rightarrow (r', s')$   
**assumes** *Seq*:  $\text{Seq } r \ c_2 \in \text{redexes } c$   
**shows**  $\exists c'. \Gamma \vdash (c, s) \rightarrow (c', s') \wedge \text{Seq } r' \ c_2 \in \text{redexes } c'$   
**proof** –  
**from** *step.Seq* [OF *step*]  
**have**  $\Gamma \vdash (\text{Seq } r \ c_2, s) \rightarrow (\text{Seq } r' \ c_2, s')$ .  
**from** *step-redexes* [OF *this Seq*]  
**show** *?thesis* .  
**qed**

**lemma** *steps-redexes-Seq*:  
**assumes** *steps*:  $\Gamma \vdash (r, s) \rightarrow^* (r', s')$   
**shows**  $\bigwedge c. \text{Seq } r \ c_2 \in \text{redexes } c \implies \exists c'. \Gamma \vdash (c, s) \rightarrow^* (c', s') \wedge \text{Seq } r' \ c_2 \in \text{redexes } c'$   
**using** *steps*  
**proof** (*induct rule: converse-rtranclp-induct2* [case-names *Refl Trans*])  
**case** *Refl*  
**then show** *?case*  
**by** (*auto*)

**next**  
**case** (*Trans r s r'' s''*)  
**have**  $\Gamma \vdash (r, s) \rightarrow (r'', s'') \text{Seq } r \ c_2 \in \text{redexes } c$  **by** *fact+*  
**from** *step-redexes-Seq* [OF *this*]  
**obtain** *c'* **where**  
*step*:  $\Gamma \vdash (c, s) \rightarrow (c', s'')$  **and**  
*r''*:  $\text{Seq } r'' \ c_2 \in \text{redexes } c'$   
**by** *blast*  
**note** *step*  
**also**  
**from** *Trans.hyps* (3) [OF *r''*]  
**obtain** *c''* **where**  
*steps*:  $\Gamma \vdash (c', s'') \rightarrow^* (c'', s')$  **and**  
*r'*:  $\text{Seq } r' \ c_2 \in \text{redexes } c''$   
**by** *blast*  
**note** *steps*  
**finally**  
**show** *?case*  
**using** *r'*  
**by** *blast*  
**qed**

**lemma** *steps-redexes-Seq'*:  
**assumes** *steps*:  $\Gamma \vdash (r, s) \rightarrow^+ (r', s')$   
**shows**  $\bigwedge c. \text{Seq } r \ c_2 \in \text{redexes } c \implies \exists c'. \Gamma \vdash (c, s) \rightarrow^+ (c', s') \wedge \text{Seq } r' \ c_2 \in \text{redexes } c'$   
**using** *steps*  
**proof** (*induct rule: tranclp-induct2* [consumes 1, case-names *Step Trans*])  
**case** (*Step r' s' c'*)

**have**  $\Gamma \vdash (r, s) \rightarrow (r', s')$  *Seq*  $r \ c_2 \in \text{redexes } c'$  **by** *fact+*  
**from** *step-redexes-Seq* [*OF this*]  
**show** *?case*  
**by** (*blast intro: r-into-trancl*)  
**next**  
**case** (*Trans*  $r' \ s' \ r'' \ s''$ )  
**from** *Trans* **obtain**  $c'$  **where**  
*steps*:  $\Gamma \vdash (c, s) \rightarrow^+ (c', s')$  **and**  
 $r'$ : *Seq*  $r' \ c_2 \in \text{redexes } c'$   
**by** *blast*  
**note** *steps*  
**moreover**  
**have**  $\Gamma \vdash (r', s') \rightarrow (r'', s'')$  **by** *fact*  
**from** *step-redexes-Seq* [*OF this r'*] **obtain**  $c''$  **where**  
*step*:  $\Gamma \vdash (c', s') \rightarrow (c'', s'')$  **and**  
 $r''$ : *Seq*  $r'' \ c_2 \in \text{redexes } c''$   
**by** *blast*  
**note** *step*  
**finally show** *?case*  
**using**  $r''$  **by** *blast*  
**qed**

**lemma** *step-redexes-Catch*:  
**assumes** *step*:  $\Gamma \vdash (r, s) \rightarrow (r', s')$   
**assumes** *Catch*: *Catch*  $r \ c_2 \in \text{redexes } c$   
**shows**  $\exists c'. \Gamma \vdash (c, s) \rightarrow (c', s') \wedge \text{Catch } r' \ c_2 \in \text{redexes } c'$   
**proof** –  
**from** *step.Catch* [*OF step*]  
**have**  $\Gamma \vdash (\text{Catch } r \ c_2, s) \rightarrow (\text{Catch } r' \ c_2, s')$ .  
**from** *step-redexes* [*OF this Catch*]  
**show** *?thesis* .  
**qed**

**lemma** *steps-redexes-Catch*:  
**assumes** *steps*:  $\Gamma \vdash (r, s) \rightarrow^* (r', s')$   
**shows**  $\bigwedge c. \text{Catch } r \ c_2 \in \text{redexes } c \implies \exists c'. \Gamma \vdash (c, s) \rightarrow^* (c', s') \wedge \text{Catch } r' \ c_2 \in \text{redexes } c'$   
**using** *steps*  
**proof** (*induct rule: converse-rtranclp-induct2* [*case-names Refl Trans*])  
**case** *Refl*  
**then show** *?case*  
**by** (*auto*)

**next**  
**case** (*Trans*  $r \ s \ r'' \ s''$ )  
**have**  $\Gamma \vdash (r, s) \rightarrow (r'', s'')$  *Catch*  $r \ c_2 \in \text{redexes } c$  **by** *fact+*  
**from** *step-redexes-Catch* [*OF this*]  
**obtain**  $c'$  **where**  
*step*:  $\Gamma \vdash (c, s) \rightarrow (c', s'')$  **and**

$r''$ : Catch  $r'' c_2 \in \text{redexes } c'$   
 by blast  
 note step  
 also  
 from *Trans.hyps* (3) [OF  $r''$ ]  
 obtain  $c''$  where  
 $\text{steps}: \Gamma \vdash (c', s'') \rightarrow^* (c'', s')$  and  
 $r'$ : Catch  $r' c_2 \in \text{redexes } c''$   
 by blast  
 note steps  
 finally  
 show ?case  
 using  $r'$   
 by blast  
 qed

**lemma** *steps-redexes-Catch'*:  
 assumes  $\text{steps}: \Gamma \vdash (r, s) \rightarrow^+ (r', s')$   
 shows  $\bigwedge c. \text{Catch } r c_2 \in \text{redexes } c$   
 $\implies \exists c'. \Gamma \vdash (c, s) \rightarrow^+ (c', s') \wedge \text{Catch } r' c_2 \in \text{redexes } c'$   
 using *steps*  
**proof** (*induct rule: tranclp-induct2* [consumes 1, case-names *Step Trans*])  
 case (*Step*  $r' s' c'$ )  
 have  $\Gamma \vdash (r, s) \rightarrow (r', s') \text{ Catch } r c_2 \in \text{redexes } c'$  by *fact+*  
 from *step-redexes-Catch* [OF *this*]  
 show ?case  
 by (*blast intro: r-into-trancl*)  
 next  
 case (*Trans*  $r' s' r'' s''$ )  
 from *Trans* obtain  $c'$  where  
 $\text{steps}: \Gamma \vdash (c, s) \rightarrow^+ (c', s')$  and  
 $r'$ : Catch  $r' c_2 \in \text{redexes } c'$   
 by blast  
 note steps  
 moreover  
 have  $\Gamma \vdash (r', s') \rightarrow (r'', s'')$  by *fact*  
 from *step-redexes-Catch* [OF *this*  $r'$ ] obtain  $c''$  where  
 $\text{step}: \Gamma \vdash (c', s') \rightarrow (c'', s'')$  and  
 $r''$ : Catch  $r'' c_2 \in \text{redexes } c''$   
 by blast  
 note step  
 finally show ?case  
 using  $r''$  by blast  
 qed

**lemma** *redexes-subset*:  $\bigwedge c'. c' \in \text{redexes } c \implies \text{redexes } c' \subseteq \text{redexes } c$   
 by (*induct c*) *auto*

**lemma** *redexes-preserves-termination*:

**assumes** *termi*:  $\Gamma \vdash c \downarrow s$   
**shows**  $\bigwedge c'. c' \in \text{redexes } c \implies \Gamma \vdash c' \downarrow s$   
**using** *termi*  
**by** *induct (auto intro: terminates.intros)*

**end**

## 9 Hoare Logic for Total Correctness

**theory** *HoareTotalDef* **imports** *HoarePartialDef Termination* **begin**

### 9.1 Validity of Hoare Tuples: $\Gamma \models_{t/F} P \ c \ Q, A$

**definition**

*validt* ::  $[(\ 's, 'p, 'f) \text{ body}, 'f \text{ set}, 's \text{ assn}, (\ 's, 'p, 'f) \text{ com}, 's \text{ assn}, 's \text{ assn}] \Rightarrow \text{bool}$   
 $(\ \langle -, \models_{t/F} \rangle \ / \ - \ - \ - \rangle \ [61, 60, 1000, 20, 1000, 1000] \ 60)$

**where**

$\Gamma \models_{t/F} P \ c \ Q, A \equiv \Gamma \models_{t/F} P \ c \ Q, A \wedge (\forall s \in \text{Normal } \langle \ 'P, \Gamma \vdash c \downarrow s \rangle)$

**definition**

*cvalidt* ::  
 $[(\ 's, 'p, 'f) \text{ body}, (\ 's, 'p) \text{ quadruple set}, 'f \text{ set},$   
 $'s \text{ assn}, (\ 's, 'p, 'f) \text{ com}, 's \text{ assn}, 's \text{ assn}] \Rightarrow \text{bool}$   
 $(\ \langle -, \models_{t/F} \rangle \ / \ - \ - \ - \rangle \ [61, 60, 60, 1000, 20, 1000, 1000] \ 60)$

**where**

$\Gamma, \Theta \models_{t/F} P \ c \ Q, A \equiv (\forall (P, p, Q, A) \in \Theta. \Gamma \models_{t/F} P \ (\text{Call } p) \ Q, A) \longrightarrow \Gamma \models_{t/F} P \ c \ Q, A$

**notation** (*ASCII*)

*validt*  $(\ \langle -, \models_{t/F} \rangle \ / \ - \ - \ - \rangle \ [61, 60, 1000, 20, 1000, 1000] \ 60)$  **and**  
*cvalidt*  $(\ \langle -, \models_{t/F} \rangle \ / \ - \ - \ - \rangle \ [61, 60, 60, 1000, 20, 1000, 1000] \ 60)$

### 9.2 Properties of Validity

**lemma** *validtI*:

$\llbracket \bigwedge s \ t. \llbracket \Gamma \vdash \langle c, \text{Normal } s \rangle \Rightarrow t; s \in P; t \notin \text{Fault } \langle F \rangle \rrbracket \rrbracket \implies t \in \text{Normal } \langle Q \cup \text{Abrupt } \langle A \rangle$

$\bigwedge s. s \in P \implies \Gamma \vdash c \downarrow (\text{Normal } s) \rrbracket$   
 $\implies \Gamma \models_{t/F} P \ c \ Q, A$

**by** (*auto simp add: validt-def valid-def*)

**lemma** *cvalidtI*:

$\llbracket \bigwedge s \ t. \llbracket \forall (P, p, Q, A) \in \Theta. \Gamma \models_{t/F} P \ (\text{Call } p) \ Q, A; \Gamma \vdash \langle c, \text{Normal } s \rangle \Rightarrow t; s \in P;$   
 $t \notin \text{Fault } \langle F \rangle \rrbracket \rrbracket$   
 $\implies t \in \text{Normal } \langle Q \cup \text{Abrupt } \langle A \rangle$

$\bigwedge s. \llbracket \forall (P,p,Q,A) \in \Theta. \Gamma \models_{t/F} P \text{ (Call } p) \text{ } Q, A; s \in P \rrbracket \implies \Gamma \vdash c \downarrow (\text{Normal } s)$   
 $\implies \Gamma, \Theta \models_{t/F} P \text{ } c \text{ } Q, A$   
**by** (*auto simp add: cvalidt-def validt-def valid-def*)

**lemma** *cvalidt-postD*:

$\llbracket \Gamma, \Theta \models_{t/F} P \text{ } c \text{ } Q, A; \forall (P,p,Q,A) \in \Theta. \Gamma \models_{t/F} P \text{ (Call } p) \text{ } Q, A; \Gamma \vdash \langle c, \text{Normal } s \rangle \implies$   
 $t;$   
 $s \in P; t \notin \text{Fault } 'F$   
 $\implies t \in \text{Normal } 'Q \cup \text{Abrupt } 'A$   
**by** (*simp add: cvalidt-def validt-def valid-def*)

**lemma** *cvalidt-termD*:

$\llbracket \Gamma, \Theta \models_{t/F} P \text{ } c \text{ } Q, A; \forall (P,p,Q,A) \in \Theta. \Gamma \models_{t/F} P \text{ (Call } p) \text{ } Q, A; s \in P \rrbracket$   
 $\implies \Gamma \vdash c \downarrow (\text{Normal } s)$   
**by** (*simp add: cvalidt-def validt-def valid-def*)

**lemma** *validt-augment-Faults*:

**assumes** *valid*:  $\Gamma \models_{t/F} P \text{ } c \text{ } Q, A$   
**assumes**  $F': F \subseteq F'$   
**shows**  $\Gamma \models_{t/F'} P \text{ } c \text{ } Q, A$   
**using** *valid*  $F'$   
**by** (*auto intro: valid-augment-Faults simp add: validt-def*)

### 9.3 The Hoare Rules: $\Gamma, \Theta \vdash_{t/F} P \text{ } c \text{ } Q, A$

**inductive** *hoaret*::  $[(\text{'s}, \text{'p}, \text{'f}) \text{ body}, (\text{'s}, \text{'p}) \text{ quadruple set}, \text{'f set},$   
 $\text{'s assn}, (\text{'s}, \text{'p}, \text{'f}) \text{ com}, \text{'s assn}, \text{'s assn}]$   
 $\implies \text{bool}$   
 $(\langle \exists -, \text{-} \vdash_{t/F} \text{-} \rangle \text{ } (-) \text{ } (-) \text{ } \text{-}, \text{-}) \text{ } [61, 60, 60, 1000, 20, 1000, 1000] 60$   
**for**  $\Gamma::(\text{'s}, \text{'p}, \text{'f}) \text{ body}$

**where**

*Skip*:  $\Gamma, \Theta \vdash_{t/F} Q \text{ } \text{Skip} \text{ } Q, A$

| *Basic*:  $\Gamma, \Theta \vdash_{t/F} \{s. f \text{ } s \in Q\} \text{ (Basic } f) \text{ } Q, A$

| *Spec*:  $\Gamma, \Theta \vdash_{t/F} \{s. (\forall t. (s, t) \in r \longrightarrow t \in Q) \wedge (\exists t. (s, t) \in r)\} \text{ (Spec } r) \text{ } Q, A$

| *Seq*:  $\llbracket \Gamma, \Theta \vdash_{t/F} P \text{ } c_1 \text{ } R, A; \Gamma, \Theta \vdash_{t/F} R \text{ } c_2 \text{ } Q, A \rrbracket$   
 $\implies$   
 $\Gamma, \Theta \vdash_{t/F} P \text{ } \text{Seq } c_1 \text{ } c_2 \text{ } Q, A$

| *Cond*:  $\llbracket \Gamma, \Theta \vdash_{t/F} (P \cap b) \text{ } c_1 \text{ } Q, A; \Gamma, \Theta \vdash_{t/F} (P \cap \text{- } b) \text{ } c_2 \text{ } Q, A \rrbracket$   
 $\implies$   
 $\Gamma, \Theta \vdash_{t/F} P \text{ (Cond } b \text{ } c_1 \text{ } c_2) \text{ } Q, A$

| *While*:  $\llbracket \text{wf } r; \forall \sigma. \Gamma, \Theta \vdash_{t/F} (\{\sigma\} \cap P \cap b) \text{ } c \text{ } (\{t. (t, \sigma) \in r\} \cap P), A \rrbracket$   
 $\implies$

$$\begin{aligned}
& \Gamma, \Theta \vdash_{t/F} P \text{ (While } b \text{ c) } (P \cap - b), A \\
| \text{ Guard: } & \Gamma, \Theta \vdash_{t/F} (g \cap P) \text{ c } Q, A \\
& \implies \\
& \Gamma, \Theta \vdash_{t/F} (g \cap P) \text{ Guard } f \text{ g c } Q, A \\
| \text{ Guarantee: } & \llbracket f \in F; \Gamma, \Theta \vdash_{t/F} (g \cap P) \text{ c } Q, A \rrbracket \\
& \implies \\
& \Gamma, \Theta \vdash_{t/F} P \text{ (Guard } f \text{ g c) } Q, A \\
| \text{ CallRec:} & \\
& \llbracket (P, p, Q, A) \in \text{Specs}; \\
& \text{wf } r; \\
& \text{Specs-wf} = (\lambda p \sigma. (\lambda (P, q, Q, A). (P \cap \{s. ((s, q), (\sigma, p)) \in r\}, q, Q, A)) \text{ 'Specs}); \\
& \forall (P, p, Q, A) \in \text{Specs}. \\
& p \in \text{dom } \Gamma \wedge (\forall \sigma. \Gamma, \Theta \cup \text{Specs-wf } p \sigma \vdash_{t/F} (\{\sigma\} \cap P) \text{ (the } (\Gamma \text{ } p)) \text{ } Q, A) \\
& \rrbracket \\
& \implies \\
& \Gamma, \Theta \vdash_{t/F} P \text{ (Call } p) \text{ } Q, A \\
| \text{ DynCom: } & \forall s \in P. \Gamma, \Theta \vdash_{t/F} P \text{ (c s) } Q, A \\
& \implies \\
& \Gamma, \Theta \vdash_{t/F} P \text{ (DynCom c) } Q, A \\
| \text{ Throw: } & \Gamma, \Theta \vdash_{t/F} A \text{ Throw } Q, A \\
| \text{ Catch: } & \llbracket \Gamma, \Theta \vdash_{t/F} P \text{ c}_1 \text{ } Q, R; \Gamma, \Theta \vdash_{t/F} R \text{ c}_2 \text{ } Q, A \rrbracket \implies \Gamma, \Theta \vdash_{t/F} P \text{ Catch } c_1 \text{ c}_2 \\
& Q, A \\
| \text{ Conseq: } & \forall s \in P. \exists P' Q' A'. \Gamma, \Theta \vdash_{t/F} P' \text{ c } Q', A' \wedge s \in P' \wedge Q' \subseteq Q \wedge A' \subseteq A \\
& \implies \Gamma, \Theta \vdash_{t/F} P \text{ c } Q, A
\end{aligned}$$

$$\begin{aligned}
| \text{ Asm: } & (P, p, Q, A) \in \Theta \\
& \implies \\
& \Gamma, \Theta \vdash_{t/F} P \text{ (Call } p) \text{ } Q, A
\end{aligned}$$

$$\begin{aligned}
| \text{ ExFalso: } & \llbracket \Gamma, \Theta \models_{t/F} P \text{ c } Q, A; \neg \Gamma \models_{t/F} P \text{ c } Q, A \rrbracket \implies \Gamma, \Theta \vdash_{t/F} P \text{ c } Q, A \\
& \text{— This is a hack rule that enables us to derive completeness for an arbitrary context } \Theta, \text{ from completeness for an empty context.}
\end{aligned}$$

Does not work, because of rule ExFalso, the context  $\Theta$  is to blame. A weaker version with empty context can be derived from soundness later on.

**lemma** *hoaret-to-hoarep*:  
**assumes** *hoaret*:  $\Gamma, \Theta \vdash_{t/F} P \text{ } p \text{ } Q, A$   
**shows**  $\Gamma, \Theta \vdash_{t/F} P \text{ } p \text{ } Q, A$



```

using hoaret
proof (induct)
  case Skip thus ?case by (rule hoarep.intros)
next
  case Basic thus ?case by (rule hoarep.intros)
next
  case Seq thus ?case by – (rule hoarep.intros)
next
  case Cond thus ?case by – (rule hoarep.intros)
next
  case (While  $r \Theta F P b c A$ )
  hence  $\forall \sigma. \Gamma, \Theta \vdash_{/F} (\{\sigma\} \cap P \cap b) c (\{t. (t, \sigma) \in r\} \cap P), A$ 
    by iprover
  hence  $\Gamma, \Theta \vdash_{/F} (P \cap b) c P, A$ 
    by (rule HoarePartialDef.conseq) blast
  then show  $\Gamma, \Theta \vdash_{/F} P \text{ While } b c (P \cap - b), A$ 
    by (rule hoarep.While)
next
  case Guard thus ?case by – (rule hoarep.intros)

next
  case DynCom thus ?case by (blast intro: hoarep.DynCom)
next
  case Throw thus ?case by – (rule hoarep.Throw)
next
  case Catch thus ?case by – (rule hoarep.Catch)
next
  case Conseq thus ?case by – (rule hoarep.Conseq,blast)
next
  case Asm thus ?case by (rule HoarePartialDef.Asm)
next
  case (ExFalso  $\Theta F P c Q A$ )
  assume  $\Gamma, \Theta \models_{t/F} P c Q, A$ 
  hence  $\Gamma, \Theta \vdash_{/F} P c Q, A$ 
    oops

lemma hoaret-augment-context:
  assumes deriv:  $\Gamma, \Theta \vdash_{t/F} P p Q, A$ 
  shows  $\bigwedge \Theta'. \Theta \subseteq \Theta' \implies \Gamma, \Theta' \vdash_{t/F} P p Q, A$ 
using deriv
proof (induct)
  case (CallRec  $P p Q A \text{ Specs } r \text{ Specs-wf } \Theta F \Theta'$ )
  have aug:  $\Theta \subseteq \Theta'$  by fact
  then
  have  $h$ :  $\bigwedge \tau p. \Theta \cup \text{Specs-wf } p \tau$ 
     $\subseteq \Theta' \cup \text{Specs-wf } p \tau$ 
    by blast

```

**have**  $\forall (P,p,Q,A) \in \text{Specs}. p \in \text{dom } \Gamma \wedge$   
 $(\forall \tau. \Gamma, \Theta \cup \text{Specs-wf } p \tau \vdash_{t/F} (\{\tau\} \cap P) \text{ (the } (\Gamma p)) Q, A \wedge$   
 $(\forall x. \Theta \cup \text{Specs-wf } p \tau$   
 $\subseteq x \longrightarrow$   
 $\Gamma, x \vdash_{t/F} (\{\tau\} \cap P) \text{ (the } (\Gamma p)) Q, A))$  **by fact**  
**hence**  $\forall (P,p,Q,A) \in \text{Specs}. p \in \text{dom } \Gamma \wedge$   
 $(\forall \tau. \Gamma, \Theta' \cup \text{Specs-wf } p \tau \vdash_{t/F} (\{\tau\} \cap P) \text{ (the } (\Gamma p)) Q, A)$   
**apply** (*clarify*)  
**apply** (*rename-tac P p Q A*)  
**apply** (*drule (1) bspec*)  
**apply** (*clarsimp*)  
**apply** (*erule-tac x= $\tau$  in allE*)  
**apply** (*clarify*)  
**apply** (*erule-tac x= $\Theta'$   $\cup$  Specs-wf p  $\tau$  in allE*)  
**apply** (*insert aug*)  
**apply** (*auto*)  
**done**  
**with CallRec show ?case by - (rule hoaret.CallRec)**  
**next**  
**case DynCom thus ?case by (blast intro: hoaret.DynCom)**  
**next**  
**case (Conseq P  $\Theta$  F c Q A  $\Theta'$ )**  
**from Conseq**  
**have**  $\forall s \in P. (\exists P' Q' A'. (\Gamma, \Theta' \vdash_{t/F} P' c Q', A') \wedge s \in P' \wedge Q' \subseteq Q \wedge A' \subseteq A)$   
**by blast**  
**with Conseq show ?case by - (rule hoaret.Conseq)**  
**next**  
**case (ExFalso  $\Theta$  F P c Q A  $\Theta'$ )**  
**have**  $\Gamma, \Theta \models_{t/F} P c Q, A \neg \Gamma \models_{t/F} P c Q, A \Theta \subseteq \Theta'$  **by fact+**  
**then show ?case**  
**by (fastforce intro: hoaret.ExFalso simp add: cvalidt-def)**  
**qed (blast intro: hoaret.intros)+**

## 9.4 Some Derived Rules

**lemma** *Conseq'*:  $\forall s. s \in P \longrightarrow$   
 $(\exists P' Q' A'.$   
 $(\forall Z. \Gamma, \Theta \vdash_{t/F} (P' Z) c (Q' Z), (A' Z)) \wedge$   
 $(\exists Z. s \in P' Z \wedge (Q' Z \subseteq Q) \wedge (A' Z \subseteq A)))$   
 $\implies$   
 $\Gamma, \Theta \vdash_{t/F} P c Q, A$   
**apply** (*rule Conseq*)  
**apply** (*rule ballI*)  
**apply** (*erule-tac x=s in allE*)  
**apply** (*clarify*)  
**apply** (*rule-tac x= $P' Z$  in exI*)  
**apply** (*rule-tac x= $Q' Z$  in exI*)  
**apply** (*rule-tac x= $A' Z$  in exI*)

**apply** *blast*  
**done**

**lemma** *conseq*:  $\llbracket \forall Z. \Gamma, \Theta \vdash_{t/F} (P' Z) c (Q' Z), (A' Z);$   
 $\forall s. s \in P \longrightarrow (\exists Z. s \in P' Z \wedge (Q' Z \subseteq Q) \wedge (A' Z \subseteq A)) \rrbracket$   
 $\implies$   
 $\Gamma, \Theta \vdash_{t/F} P c Q, A$   
**by** (*rule Conseq*) *blast*

**theorem** *conseqPrePost*:  
 $\Gamma, \Theta \vdash_{t/F} P' c Q', A' \implies P \subseteq P' \implies Q' \subseteq Q \implies A' \subseteq A \implies \Gamma, \Theta \vdash_{t/F} P c Q, A$   
**by** (*rule conseq* [**where**  $?P' = \lambda Z. P'$  **and**  $?Q' = \lambda Z. Q'$ ]) *auto*

**lemma** *conseqPre*:  $\Gamma, \Theta \vdash_{t/F} P' c Q, A \implies P \subseteq P' \implies \Gamma, \Theta \vdash_{t/F} P c Q, A$   
**by** (*rule conseq*) *auto*

**lemma** *conseqPost*:  $\Gamma, \Theta \vdash_{t/F} P c Q', A' \implies Q' \subseteq Q \implies A' \subseteq A \implies \Gamma, \Theta \vdash_{t/F} P c Q, A$   
**by** (*rule conseq*) *auto*

**lemma** *Spec-wf-conv*:  
 $(\lambda(P, q, Q, A). (P \cap \{s. ((s, q), \tau, p) \in r\}, q, Q, A)) \text{ '}$   
 $(\bigcup p \in \text{Procs}. \bigcup Z. \{(P p Z, p, Q p Z, A p Z)\}) =$   
 $(\bigcup q \in \text{Procs}. \bigcup Z. \{(P q Z \cap \{s. ((s, q), \tau, p) \in r\}, q, Q q Z, A q Z)\})$   
**by** (*auto intro!*: *image-eqI*)

**lemma** *CallRec'*:  
 $\llbracket p \in \text{Procs}; \text{Procs} \subseteq \text{dom } \Gamma;$   
 $\text{wf } r;$   
 $\forall p \in \text{Procs}. \forall \tau Z.$   
 $\Gamma, \Theta \cup (\bigcup q \in \text{Procs}. \bigcup Z.$   
 $\{((P q Z) \cap \{s. ((s, q), (\tau, p)) \in r\}, q, Q q Z, (A q Z))\})$   
 $\vdash_{t/F} (\{\tau\} \cap (P p Z)) (\text{the } (\Gamma p)) (Q p Z), (A p Z) \rrbracket$   
 $\implies$   
 $\Gamma, \Theta \vdash_{t/F} (P p Z) (\text{Call } p) (Q p Z), (A p Z)$   
**apply** (*rule CallRec* [**where**  $\text{Specs} = \bigcup p \in \text{Procs}. \bigcup Z. \{((P p Z), p, Q p Z, A p Z)\}$   
**and**  
 $r = r$ ])

**apply** *blast*  
**apply** *assumption*  
**apply** (*rule refl*)  
**apply** (*clarsimp*)  
**apply** (*rename-tac p'*)  
**apply** (*rule conjI*)  
**apply** *blast*  
**apply** (*intro allI*)

```

apply (rename-tac  $Z \tau$ )
apply (drule-tac  $x=p'$  in bspec, assumption)
apply (erule-tac  $x=\tau$  in allE)
apply (erule-tac  $x=Z$  in allE)
apply (fastforce simp add: Spec-wf-conv)
done

end

```

## 10 Properties of Total Correctness Hoare Logic

```

theory HoareTotalProps imports SmallStep HoareTotalDef HoarePartialProps begin

```

### 10.1 Soundness

```

lemma hoaret-sound:
  assumes hoare:  $\Gamma, \Theta \vdash_{t/F} P \text{ c } Q, A$ 
  shows  $\Gamma, \Theta \models_{t/F} P \text{ c } Q, A$ 
using hoare
proof (induct)
  case (Skip  $\Theta F P A$ )
  show  $\Gamma, \Theta \models_{t/F} P \text{ Skip } P, A$ 
  proof (rule cvalidtI)
    fix  $s t$ 
    assume  $\Gamma \vdash \langle \text{Skip}, \text{Normal } s \rangle \Rightarrow t s \in P$ 
    thus  $t \in \text{Normal } ' P \cup \text{Abrupt } ' A$ 
    by cases auto
  next
    fix  $s$  show  $\Gamma \vdash \text{Skip} \downarrow \text{Normal } s$ 
    by (rule terminates.intros)
  qed
next
  case (Basic  $\Theta F f P A$ )
  show  $\Gamma, \Theta \models_{t/F} \{s. f s \in P\} (\text{Basic } f) P, A$ 
  proof (rule cvalidtI)
    fix  $s t$ 
    assume  $\Gamma \vdash \langle \text{Basic } f, \text{Normal } s \rangle \Rightarrow t s \in \{s. f s \in P\}$ 
    thus  $t \in \text{Normal } ' P \cup \text{Abrupt } ' A$ 
    by cases auto
  next
    fix  $s$  show  $\Gamma \vdash \text{Basic } f \downarrow \text{Normal } s$ 
    by (rule terminates.intros)
  qed
next
  case (Spec  $\Theta F r Q A$ )
  show  $\Gamma, \Theta \models_{t/F} \{s. (\forall t. (s, t) \in r \longrightarrow t \in Q) \wedge (\exists t. (s, t) \in r)\} \text{Spec } r Q, A$ 
  proof (rule cvalidtI)

```

```

fix s t
assume  $\Gamma \vdash \langle \text{Spec } r, \text{Normal } s \rangle \Rightarrow t$ 
       $s \in \{s. (\forall t. (s, t) \in r \longrightarrow t \in Q) \wedge (\exists t. (s, t) \in r)\}$ 
thus  $t \in \text{Normal } ' Q \cup \text{Abrupt } ' A$ 
  by cases auto
next
  fix s show  $\Gamma \vdash \text{Spec } r \downarrow \text{Normal } s$ 
    by (rule terminates.intros)
qed
next
case (Seq  $\Theta$  F P c1 R A c2 Q)
have valid-c1:  $\Gamma, \Theta \models_{t/F} P \text{ c1 } R, A$  by fact
have valid-c2:  $\Gamma, \Theta \models_{t/F} R \text{ c2 } Q, A$  by fact
show  $\Gamma, \Theta \models_{t/F} P \text{ Seq } \text{c1 } \text{c2 } Q, A$ 
proof (rule cvalidtI)
  fix s t
  assume ctxt:  $\forall (P, p, Q, A) \in \Theta. \Gamma \models_{t/F} P (\text{Call } p) Q, A$ 
  assume exec:  $\Gamma \vdash \langle \text{Seq } \text{c1 } \text{c2}, \text{Normal } s \rangle \Rightarrow t$ 
  assume P:  $s \in P$ 
  assume t-notin-F:  $t \notin \text{Fault } ' F$ 
  from exec P obtain r where
    exec-c1:  $\Gamma \vdash \langle \text{c1}, \text{Normal } s \rangle \Rightarrow r$  and exec-c2:  $\Gamma \vdash \langle \text{c2}, r \rangle \Rightarrow t$ 
    by cases auto
  with t-notin-F have  $r \notin \text{Fault } ' F$ 
    by (auto dest: Fault-end)
  from valid-c1 ctxt exec-c1 P this
  have r:  $r \in \text{Normal } ' R \cup \text{Abrupt } ' A$ 
    by (rule cvalidt-postD)
  show  $t \in \text{Normal } ' Q \cup \text{Abrupt } ' A$ 
  proof (cases r)
    case (Normal r')
      with exec-c2 r
      show  $t \in \text{Normal } ' Q \cup \text{Abrupt } ' A$ 
        apply -
        apply (rule cvalidt-postD [OF valid-c2 ctxt - - t-notin-F])
        apply auto
        done
    next
    case (Abrupt r')
      with exec-c2 have  $t = \text{Abrupt } r'$ 
        by (auto elim: exec-elim-cases)
      with Abrupt r show ?thesis
        by auto
    next
    case Fault with r show ?thesis by blast
  next
  case Stuck with r show ?thesis by blast
qed
next

```

```

fix s
assume ctxt:  $\forall (P, p, Q, A) \in \Theta. \Gamma \models_{t/F} P \text{ (Call } p) \text{ } Q, A$ 
assume P:  $s \in P$ 
show  $\Gamma \vdash \text{Seq } c1 \ c2 \downarrow \text{Normal } s$ 
proof –
  from valid-c1 ctxt P
  have  $\Gamma \vdash c1 \downarrow \text{Normal } s$ 
    by (rule cvalidt-termD)
  moreover
  {
    fix r assume exec-c1:  $\Gamma \vdash \langle c1, \text{Normal } s \rangle \Rightarrow r$ 
    have  $\Gamma \vdash c2 \downarrow r$ 
    proof (cases r)
      case (Normal r')
        with cvalidt-postD [OF valid-c1 ctxt exec-c1 P]
        have  $r: r \in \text{Normal} \text{ ' } R$ 
          by auto
        with cvalidt-termD [OF valid-c2 ctxt] exec-c1
        show  $\Gamma \vdash c2 \downarrow r$ 
          by auto
        qed auto
      }
    ultimately show ?thesis
      by (iprover intro: terminates.intros)
  }
qed
qed
next
case (Cond  $\Theta \ F \ P \ b \ c1 \ Q \ A \ c2$ )
have valid-c1:  $\Gamma, \Theta \models_{t/F} (P \cap b) \ c1 \ Q, A$  by fact
have valid-c2:  $\Gamma, \Theta \models_{t/F} (P \cap \neg b) \ c2 \ Q, A$  by fact
show  $\Gamma, \Theta \models_{t/F} P \ \text{Cond } b \ c1 \ c2 \ Q, A$ 
proof (rule cvalidtI)
  fix s t
  assume ctxt:  $\forall (P, p, Q, A) \in \Theta. \Gamma \models_{t/F} P \text{ (Call } p) \text{ } Q, A$ 
  assume exec:  $\Gamma \vdash \langle \text{Cond } b \ c1 \ c2, \text{Normal } s \rangle \Rightarrow t$ 
  assume P:  $s \in P$ 
  assume t-notin-F:  $t \notin \text{Fault} \text{ ' } F$ 
  show  $t \in \text{Normal} \text{ ' } Q \cup \text{Abrupt} \text{ ' } A$ 
  proof (cases s  $\in b$ )
    case True
      with exec have  $\Gamma \vdash \langle c1, \text{Normal } s \rangle \Rightarrow t$ 
        by cases auto
      with P True
      show ?thesis
        by – (rule cvalidt-postD [OF valid-c1 ctxt - - t-notin-F], auto)
    next
      case False
      with exec P have  $\Gamma \vdash \langle c2, \text{Normal } s \rangle \Rightarrow t$ 

```

```

    by cases auto
  with P False
  show ?thesis
    by - (rule cvalidt-postD [OF valid-c2 ctxt - - t-notin-F],auto)
qed
next
fix s
assume ctxt:  $\forall (P, p, Q, A) \in \Theta. \Gamma \models_{t/F} P (Call\ p)\ Q, A$ 
assume P:  $s \in P$ 
thus  $\Gamma \vdash Cond\ b\ c1\ c2 \downarrow Normal\ s$ 
  using cvalidt-termD [OF valid-c1 ctxt] cvalidt-termD [OF valid-c2 ctxt]
  by (cases  $s \in b$ ) (auto intro: terminates.intros)
qed
next
case (While r  $\Theta\ F\ P\ b\ c\ A$ )
assume wf: wf r
have valid-c:  $\forall \sigma. \Gamma, \Theta \models_{t/F} (\{\sigma\} \cap P \cap b)\ c\ (\{t. (t, \sigma) \in r\} \cap P), A$ 
  using While.hyps by iprover
show  $\Gamma, \Theta \models_{t/F} P (While\ b\ c)\ (P \cap -\ b), A$ 
proof (rule cvalidtI)
  fix s t
  assume ctxt:  $\forall (P, p, Q, A) \in \Theta. \Gamma \models_{t/F} P (Call\ p)\ Q, A$ 
  assume wprems:  $\Gamma \vdash \langle While\ b\ c, Normal\ s \rangle \Rightarrow t\ s \in P\ t \notin Fault\ 'F$ 
  from wf
  have  $\bigwedge t. [\Gamma \vdash \langle While\ b\ c, Normal\ s \rangle \Rightarrow t; s \in P; t \notin Fault\ 'F]$ 
     $\Rightarrow t \in Normal\ '(P \cap -\ b) \cup Abrupt\ 'A$ 
  proof (induct)
    fix s t
    assume hyp:
       $\bigwedge s' t. [(s', s) \in r; \Gamma \vdash \langle While\ b\ c, Normal\ s' \rangle \Rightarrow t; s' \in P; t \notin Fault\ 'F]$ 
         $\Rightarrow t \in Normal\ '(P \cap -\ b) \cup Abrupt\ 'A$ 
    assume exec:  $\Gamma \vdash \langle While\ b\ c, Normal\ s \rangle \Rightarrow t$ 
    assume P:  $s \in P$ 
    assume t-notin-F:  $t \notin Fault\ 'F$ 
    from exec
    show  $t \in Normal\ '(P \cap -\ b) \cup Abrupt\ 'A$ 
    proof (cases)
      fix s'
      assume b:  $s \in b$ 
      assume exec-c:  $\Gamma \vdash \langle c, Normal\ s \rangle \Rightarrow s'$ 
      assume exec-w:  $\Gamma \vdash \langle While\ b\ c, s' \rangle \Rightarrow t$ 
      from exec-w t-notin-F have  $s' \notin Fault\ 'F$ 
        by (auto dest: Fault-end)
      from exec-c P b valid-c ctxt this
      have  $s': s' \in Normal\ '(\{s'. (s', s) \in r\} \cap P) \cup Abrupt\ 'A$ 
        by (auto simp add: cvalidt-def validt-def valid-def)
      show ?thesis
    proof (cases s')
      case Normal

```

```

    with exec-w s' t-notin-F
    show ?thesis
      by - (rule hyp,auto)
  next
    case Abrupt
    with exec-w have  $t=s'$ 
      by (auto dest: Abrupt-end)
    with Abrupt s' show ?thesis
      by blast
  next
    case Fault
    with exec-w have  $t=s'$ 
      by (auto dest: Fault-end)
    with Fault s' show ?thesis
      by blast
  next
    case Stuck
    with exec-w have  $t=s'$ 
      by (auto dest: Stuck-end)
    with Stuck s' show ?thesis
      by blast
  qed
next
  assume  $s \notin b$   $t=Normal\ s$  with P show ?thesis by simp
  qed
qed
with wprems show  $t \in Normal \text{ ' } (P \cap - b) \cup Abrupt \text{ ' } A$  by blast
next
  fix s
  assume ctxt:  $\forall (P, p, Q, A) \in \Theta. \Gamma \models_{t/F} P (Call\ p)\ Q, A$ 
  assume  $s \in P$ 
  with wf
  show  $\Gamma \vdash While\ b\ c \downarrow Normal\ s$ 
  proof (induct)
    fix s
    assume hyp:  $\bigwedge s'. \llbracket (s',s) \in r; s' \in P \rrbracket$ 
       $\implies \Gamma \vdash While\ b\ c \downarrow Normal\ s'$ 
    assume P:  $s \in P$ 
    show  $\Gamma \vdash While\ b\ c \downarrow Normal\ s$ 
    proof (cases  $s \in b$ )
      case False with P show ?thesis
        by (blast intro: terminates.intros)
    next
      case True
      with valid-c P ctxt
      have  $\Gamma \vdash c \downarrow Normal\ s$ 
        by (simp add: cvalidt-def validt-def)
      moreover
      {

```



```

    fix s'
    assume exec-c:  $\Gamma \vdash \langle c, \text{Normal } s \rangle \Rightarrow s'$ 
    have  $\Gamma \vdash \text{While } b \ c \ \downarrow \ s'$ 
    proof (cases s')
      case (Normal s'')
        with exec-c P True valid-c ctxt
        have s':  $s' \in \text{Normal } \cdot (\{s'. (s', s) \in r\} \cap P)$ 
          by (fastforce simp add: cvalidt-def validt-def valid-def)
        then show ?thesis
          by (blast intro: hyp)
      qed auto
    }
    ultimately
    show ?thesis
      by (blast intro: terminates.intros)
    qed
  qed
next
case (Guard  $\Theta \ F \ g \ P \ c \ Q \ A \ f$ )
have valid-c:  $\Gamma, \Theta \models_{t/F} (g \cap P) \ c \ Q, A$  by fact
show  $\Gamma, \Theta \models_{t/F} (g \cap P) \ \text{Guard } f \ g \ c \ Q, A$ 
proof (rule cvalidtI)
  fix s t
  assume ctxt:  $\forall (P, p, Q, A) \in \Theta. \Gamma \models_{t/F} P \ (\text{Call } p) \ Q, A$ 
  assume exec:  $\Gamma \vdash \langle \text{Guard } f \ g \ c, \text{Normal } s \rangle \Rightarrow t$ 
  assume t-notin-F:  $t \notin \text{Fault } \cdot F$ 
  assume P:s  $\in (g \cap P)$ 
  from exec P have  $\Gamma \vdash \langle c, \text{Normal } s \rangle \Rightarrow t$ 
    by cases auto
  from valid-c ctxt this P t-notin-F
  show  $t \in \text{Normal } \cdot Q \cup \text{Abrupt } \cdot A$ 
    by (rule cvalidt-postD)
next
fix s
assume ctxt:  $\forall (P, p, Q, A) \in \Theta. \Gamma \models_{t/F} P \ (\text{Call } p) \ Q, A$ 
assume P:s  $\in (g \cap P)$ 
thus  $\Gamma \vdash \text{Guard } f \ g \ c \ \downarrow \ \text{Normal } s$ 
  by (auto intro: terminates.intros cvalidt-termD [OF valid-c ctxt])
qed
next
case (Guarantee f F  $\Theta \ g \ P \ c \ Q \ A$ )
have valid-c:  $\Gamma, \Theta \models_{t/F} (g \cap P) \ c \ Q, A$  by fact
have f-F:  $f \in F$  by fact
show  $\Gamma, \Theta \models_{t/F} P \ \text{Guard } f \ g \ c \ Q, A$ 
proof (rule cvalidtI)
  fix s t
  assume ctxt:  $\forall (P, p, Q, A) \in \Theta. \Gamma \models_{t/F} P \ (\text{Call } p) \ Q, A$ 

```

```

assume exec:  $\Gamma \vdash \langle \text{Guard } f \ g \ c, \text{Normal } s \rangle \Rightarrow t$ 
assume t-notin-F:  $t \notin \text{Fault } ' F$ 
assume P:  $s \in P$ 
from exec f-F t-notin-F have g:  $s \in g$ 
  by cases auto
with P have P':  $s \in g \cap P$ 
  by blast
from exec g have  $\Gamma \vdash \langle c, \text{Normal } s \rangle \Rightarrow t$ 
  by cases auto
from valid-c ctxt this P' t-notin-F
show  $t \in \text{Normal } ' Q \cup \text{Abrupt } ' A$ 
  by (rule cvalidt-postD)
next
fix s
assume ctxt:  $\forall (P, p, Q, A) \in \Theta. \Gamma \models_{t/F} P \text{ (Call } p) \ Q, A$ 
assume P:  $s \in P$ 
thus  $\Gamma \vdash \text{Guard } f \ g \ c \ \downarrow \ \text{Normal } s$ 
  by (auto intro: terminates.intros cvalidt-termD [OF valid-c ctxt])
qed
next
case (CallRec P p Q A Specs r Specs-wf  $\Theta$  F)
have p:  $(P, p, Q, A) \in \text{Specs}$  by fact
have wf: wf r by fact
have Specs-wf:
  Specs-wf =  $(\lambda p \ \tau. (\lambda (P, q, Q, A). (P \cap \{s. ((s, q), \tau, p) \in r\}, q, Q, A))) \ ' \ \text{Specs}$  by
fact
from CallRec.hyps
have valid-body:
   $\forall (P, p, Q, A) \in \text{Specs}. p \in \text{dom } \Gamma \wedge$ 
   $(\forall \tau. \Gamma, \Theta \cup \text{Specs-wf } p \ \tau \models_{t/F} (\{\tau\} \cap P) \ \text{the } (\Gamma \ p)) \ Q, A$  by auto
show  $\Gamma, \Theta \models_{t/F} P \text{ (Call } p) \ Q, A$ 
proof –
  {
    fix  $\tau p$ 
assume ctxt:  $\forall (P, p, Q, A) \in \Theta. \Gamma \models_{t/F} P \text{ (Call } p) \ Q, A$ 
from wf
have  $\bigwedge \tau \ p \ P \ Q \ A. \llbracket \tau p = (\tau, p); (P, p, Q, A) \in \text{Specs} \rrbracket \Longrightarrow$ 
 $\Gamma \models_{t/F} (\{\tau\} \cap P) \ \text{the } (\Gamma \ (p)) \ Q, A$ 
proof (induct  $\tau p$  rule: wf-induct [rule-format, consumes 1, case-names WF])
case (WF  $\tau p \ \tau \ p \ P \ Q \ A$ )
have  $\tau p$ :  $\tau p = (\tau, p)$  by fact
have p:  $(P, p, Q, A) \in \text{Specs}$  by fact
    {
      fix  $q \ P' \ Q' \ A'$ 
assume q:  $(P', q, Q', A') \in \text{Specs}$ 
have  $\Gamma \models_{t/F} (P' \cap \{s. ((s, q), \tau, p) \in r\}) \ \text{(Call } q) \ Q', A'$ 
proof (rule validtI)
      fix s t

```

```

assume exec-q:
   $\Gamma \vdash \langle \text{Call } q, \text{Normal } s \rangle \Rightarrow t$ 
assume Pre:  $s \in P' \cap \{s. ((s, q), \tau, p) \in r\}$ 
assume t-notin-F:  $t \notin \text{Fault } F$ 
from Pre  $q \ \tau \ p$ 
have valid-bdy:
   $\Gamma \models_{t/F} (\{s\} \cap P')$  the  $(\Gamma \ q) \ Q', A'$ 
  by  $-$  (rule WF.hyps, auto)
from Pre  $q$ 
have Pre':  $s \in \{s\} \cap P'$ 
  by auto
from exec-q show  $t \in \text{Normal } Q' \cup \text{Abrupt } A'$ 
proof (cases)
  fix bdy
  assume bdy:  $\Gamma \ q = \text{Some } bdy$ 
  assume exec-bdy:  $\Gamma \vdash \langle bdy, \text{Normal } s \rangle \Rightarrow t$ 
  from valid-bdy [simplified bdy option.sel] t-notin-F exec-bdy Pre'
  have  $t \in \text{Normal } Q' \cup \text{Abrupt } A'$ 
  by (auto simp add: validt-def valid-def)
  with Pre  $q$ 
  show ?thesis
  by auto
next
  assume  $\Gamma \ q = \text{None}$ 
  with q valid-body have False by auto
  thus ?thesis ..
qed
next
fix s
assume Pre:  $s \in P' \cap \{s. ((s, q), \tau, p) \in r\}$ 
from Pre  $q \ \tau \ p$ 
have valid-bdy:
   $\Gamma \models_{t/F} (\{s\} \cap P')$  (the  $(\Gamma \ q)$ )  $Q', A'$ 
  by  $-$  (rule WF.hyps, auto)
from Pre  $q$ 
have Pre':  $s \in \{s\} \cap P'$ 
  by auto
from valid-bdy ctxt Pre'
have  $\Gamma \vdash \text{the } (\Gamma \ q) \downarrow \text{Normal } s$ 
  by (auto simp add: validt-def)
with valid-body  $q$ 
show  $\Gamma \vdash \text{Call } q \downarrow \text{Normal } s$ 
  by (fastforce intro: terminates.Call)
qed
}
hence  $\forall (P, p, Q, A) \in \text{Specs-wf } p \ \tau. \Gamma \models_{t/F} P \ \text{Call } p \ Q, A$ 
  by (auto simp add: cvalidt-def Specs-wf)
with ctxt have  $\forall (P, p, Q, A) \in \Theta \cup \text{Specs-wf } p \ \tau. \Gamma \models_{t/F} P \ \text{Call } p \ Q, A$ 
  by auto

```

```

    with  $p$  valid-body
    show  $\Gamma \models_{t/F} (\{\tau\} \cap P)$  (the  $(\Gamma p)$ )  $Q, A$ 
      by (simp add: cvalidt-def) blast
  qed
}
note lem = this
have valid-body':
   $\bigwedge \tau. \forall (P, p, Q, A) \in \Theta. \Gamma \models_{t/F} P$  (Call  $p$ )  $Q, A \implies$ 
   $\forall (P, p, Q, A) \in \text{Specs}. \Gamma \models_{t/F} (\{\tau\} \cap P)$  (the  $(\Gamma p)$ )  $Q, A$ 
  by (auto intro: lem)
show  $\Gamma, \Theta \models_{t/F} P$  (Call  $p$ )  $Q, A$ 
proof (rule cvalidtI)
  fix  $s t$ 
  assume ctxt:  $\forall (P, p, Q, A) \in \Theta. \Gamma \models_{t/F} P$  (Call  $p$ )  $Q, A$ 
  assume exec-call:  $\Gamma \vdash \langle \text{Call } p, \text{Normal } s \rangle \Rightarrow t$ 
  assume  $P: s \in P$ 
  assume t-notin-F:  $t \notin \text{Fault } ' F$ 
  from exec-call show  $t \in \text{Normal } ' Q \cup \text{Abrupt } ' A$ 
  proof (cases)
    fix bdy
    assume bdy:  $\Gamma p = \text{Some bdy}$ 
    assume exec-body:  $\Gamma \vdash \langle \text{bdy}, \text{Normal } s \rangle \Rightarrow t$ 
    from exec-body bdy  $p P$  t-notin-F
      valid-body' [of  $s, OF$  ctxt]
      ctxt
    have  $t \in \text{Normal } ' Q \cup \text{Abrupt } ' A$ 
    apply (simp only: cvalidt-def validt-def valid-def)
    apply (drule (1) bspec)
    apply auto
    done
  with  $p P$ 
  show ?thesis
  by simp
  next
  assume  $\Gamma p = \text{None}$ 
  with  $p$  valid-body have False by auto
  thus ?thesis by simp
  qed
next
  fix  $s$ 
  assume ctxt:  $\forall (P, p, Q, A) \in \Theta. \Gamma \models_{t/F} P$  (Call  $p$ )  $Q, A$ 
  assume  $P: s \in P$ 
  show  $\Gamma \vdash \text{Call } p \downarrow \text{Normal } s$ 
  proof -
    from ctxt  $P p$  valid-body' [of  $s, OF$  ctxt]
    have  $\Gamma \vdash (\text{the } (\Gamma p)) \downarrow \text{Normal } s$ 
    by (auto simp add: cvalidt-def validt-def)
    with valid-body  $p$  show ?thesis
  
```

```

      by (fastforce intro: terminates.Call)
    qed
  qed
  qed
next
case (DynCom P  $\Theta$  F c Q A)
hence valid-c:  $\forall s \in P. \Gamma, \Theta \models_{t/F} P (c s) Q, A$  by simp
show  $\Gamma, \Theta \models_{t/F} P \text{ DynCom } c Q, A$ 
proof (rule cvalidI)
  fix s t
  assume ctxt:  $\forall (P, p, Q, A) \in \Theta. \Gamma \models_{t/F} P (Call p) Q, A$ 
  assume exec:  $\Gamma \vdash \langle \text{DynCom } c, \text{Normal } s \rangle \Rightarrow t$ 
  assume P:  $s \in P$ 
  assume t-notin-F:  $t \notin \text{Fault } ' F$ 
  from exec show  $t \in \text{Normal } ' Q \cup \text{Abrupt } ' A$ 
  proof (cases)
    assume  $\Gamma \vdash \langle c s, \text{Normal } s \rangle \Rightarrow t$ 
    from cvalid-postD [OF valid-c [rule-format, OF P] ctxt this P t-notin-F]
    show ?thesis .
  qed
next
fix s
assume ctxt:  $\forall (P, p, Q, A) \in \Theta. \Gamma \models_{t/F} P (Call p) Q, A$ 
assume P:  $s \in P$ 
show  $\Gamma \vdash \text{DynCom } c \downarrow \text{Normal } s$ 
proof -
  from cvalid-termD [OF valid-c [rule-format, OF P] ctxt P]
  have  $\Gamma \vdash c s \downarrow \text{Normal } s$  .
  thus ?thesis
    by (rule terminates.intros)
  qed
qed
next
case (Throw  $\Theta$  F A Q)
show  $\Gamma, \Theta \models_{t/F} A \text{ Throw } Q, A$ 
proof (rule cvalidI)
  fix s t
  assume  $\Gamma \vdash \langle \text{Throw}, \text{Normal } s \rangle \Rightarrow t$ 
  then show  $t \in \text{Normal } ' Q \cup \text{Abrupt } ' A$ 
    by cases simp
next
fix s
show  $\Gamma \vdash \text{Throw} \downarrow \text{Normal } s$ 
  by (rule terminates.intros)
qed
next
case (Catch  $\Theta$  F P c1 Q R c2 A)
have valid-c1:  $\Gamma, \Theta \models_{t/F} P c_1 Q, R$  by fact

```

**have**  $valid-c2: \Gamma, \Theta \models_{t/F} R \ c_2 \ Q, A$  **by** *fact*  
**show**  $\Gamma, \Theta \models_{t/F} P \ Catch \ c_1 \ c_2 \ Q, A$   
**proof** (*rule cvalidtI*)  
  **fix**  $s \ t$   
  **assume**  $ctxt: \forall (P, p, Q, A) \in \Theta. \Gamma \models_{t/F} P \ (Call \ p) \ Q, A$   
  **assume**  $exec: \Gamma \vdash \langle Catch \ c_1 \ c_2, Normal \ s \rangle \Rightarrow t$   
  **assume**  $P: s \in P$   
  **assume**  $t\text{-notin-}F: t \notin Fault \ ' \ F$   
  **from**  $exec$  **show**  $t \in Normal \ ' \ Q \cup Abrupt \ ' \ A$   
  **proof** (*cases*)  
    **fix**  $s'$   
    **assume**  $exec-c1: \Gamma \vdash \langle c_1, Normal \ s \rangle \Rightarrow Abrupt \ s'$   
    **assume**  $exec-c2: \Gamma \vdash \langle c_2, Normal \ s \rangle \Rightarrow t$   
    **from**  $cvalidt\text{-}postD \ [OF \ valid\text{-}c1 \ ctxt \ exec\text{-}c1 \ P]$   
    **have**  $Abrupt \ s' \in Abrupt \ ' \ R$   
    **by** *auto*  
    **with**  $cvalidt\text{-}postD \ [OF \ valid\text{-}c2 \ ctxt] \ exec\text{-}c2 \ t\text{-notin-}F$   
    **show** *?thesis*  
    **by** *fastforce*  
  **next**  
  **assume**  $exec-c1: \Gamma \vdash \langle c_1, Normal \ s \rangle \Rightarrow t$   
  **assume**  $notAbr: \neg isAbr \ t$   
  **from**  $cvalidt\text{-}postD \ [OF \ valid\text{-}c1 \ ctxt \ exec\text{-}c1 \ P] \ t\text{-notin-}F$   
  **have**  $t \in Normal \ ' \ Q \cup Abrupt \ ' \ R$ .  
  **with**  $notAbr$   
  **show** *?thesis*  
  **by** *auto*  
  **qed**  
**next**  
  **fix**  $s$   
  **assume**  $ctxt: \forall (P, p, Q, A) \in \Theta. \Gamma \models_{t/F} P \ (Call \ p) \ Q, A$   
  **assume**  $P: s \in P$   
  **show**  $\Gamma \vdash Catch \ c_1 \ c_2 \downarrow Normal \ s$   
  **proof** –  
  **from**  $valid\text{-}c1 \ ctxt \ P$   
  **have**  $\Gamma \vdash c_1 \downarrow Normal \ s$   
  **by** (*rule cvalidt-termD*)  
  **moreover**  
  {  
    **fix**  $r$  **assume**  $exec-c1: \Gamma \vdash \langle c_1, Normal \ s \rangle \Rightarrow Abrupt \ r$   
    **from**  $cvalidt\text{-}postD \ [OF \ valid\text{-}c1 \ ctxt \ exec\text{-}c1 \ P]$   
    **have**  $r: Abrupt \ r \in Normal \ ' \ Q \cup Abrupt \ ' \ R$   
    **by** *auto*  
    **hence**  $Abrupt \ r \in Abrupt \ ' \ R$  **by** *fast*  
    **with**  $cvalidt\text{-}termD \ [OF \ valid\text{-}c2 \ ctxt] \ exec\text{-}c1$   
    **have**  $\Gamma \vdash c_2 \downarrow Normal \ r$   
    **by** *fast*  
  }  
  **ultimately show** *?thesis*

by (*iprover intro: terminates.intros*)  
 qed  
 qed  
 next  
 case (*Conseq P  $\Theta$  F c Q A*)  
 hence *adapt*:  
 $\forall s \in P. (\exists P' Q' A'. (\Gamma, \Theta \models_{t/F} P' c Q', A') \wedge s \in P' \wedge Q' \subseteq Q \wedge A' \subseteq A)$  by  
*blast*  
 show  $\Gamma, \Theta \models_{t/F} P c Q, A$   
 proof (*rule cvalidtI*)  
 fix *s t*  
 assume *ctxt*:  $\forall (P, p, Q, A) \in \Theta. \Gamma \models_{t/F} P (Call\ p)\ Q, A$   
 assume *exec*:  $\Gamma \vdash \langle c, Normal\ s \rangle \Rightarrow t$   
 assume *P*:  $s \in P$   
 assume *t-notin-F*:  $t \notin Fault\ 'F$   
 show  $t \in Normal\ 'Q \cup Abrupt\ 'A$   
 proof –  
 from *adapt* [*rule-format, OF P*]  
 obtain *P' and Q' and A' where*  
 $valid-P'-Q': \Gamma, \Theta \models_{t/F} P' c Q', A'$   
 and *weaken*:  $s \in P' Q' \subseteq Q A' \subseteq A$   
 by *blast*  
 from *exec valid-P'-Q' ctxt t-notin-F*  
 have *P'-Q'*:  $Normal\ s \in Normal\ 'P' \longrightarrow$   
 $t \in Normal\ 'Q' \cup Abrupt\ 'A'$   
 by (*unfold cvalidt-def validt-def valid-def*) *blast*  
 hence  $s \in P' \longrightarrow t \in Normal\ 'Q' \cup Abrupt\ 'A'$   
 by *blast*  
 with *weaken*  
 show *?thesis*  
 by *blast*  
 qed  
 next  
 fix *s*  
 assume *ctxt*:  $\forall (P, p, Q, A) \in \Theta. \Gamma \models_{t/F} P (Call\ p)\ Q, A$   
 assume *P*:  $s \in P$   
 show  $\Gamma \vdash c \downarrow Normal\ s$   
 proof –  
 from *P adapt*  
 obtain *P' and Q' and A' where*  
 $\Gamma, \Theta \models_{t/F} P' c Q', A'$   
 $s \in P'$   
 by *blast*  
 with *ctxt*  
 show *?thesis*  
 by (*simp add: cvalidt-def validt-def*)  
 qed  
 qed

```

next
  case (Asm  $P\ p\ Q\ A\ \Theta\ F$ )
  assume  $(P, p, Q, A) \in \Theta$ 
  then show  $\Gamma, \Theta \models_{t/F} P\ (Call\ p)\ Q, A$ 
    by (auto simp add: cvalidt-def)
next
  case ExFalso thus ?case by iprover
qed

```

```

lemma hoaret-sound':
 $\Gamma, \{\} \vdash_{t/F} P\ c\ Q, A \implies \Gamma \models_{t/F} P\ c\ Q, A$ 
  apply (drule hoaret-sound)
  apply (simp add: cvalidt-def)
done

```

```

theorem total-to-partial:
assumes total:  $\Gamma, \{\} \vdash_{t/F} P\ c\ Q, A$  shows  $\Gamma, \{\} \vdash_{t/F} P\ c\ Q, A$ 
proof –
  from total have  $\Gamma, \{\} \models_{t/F} P\ c\ Q, A$ 
    by (rule hoaret-sound)
  hence  $\Gamma \models_{t/F} P\ c\ Q, A$ 
    by (simp add: cvalidt-def validt-def cvalid-def)
  thus ?thesis
    by (rule hoare-complete)
qed

```

## 10.2 Completeness

```

lemma MGT-valid:
 $\Gamma \models_{t/F} \{s. s = Z \wedge \Gamma \vdash \langle c, Normal\ s \rangle \Rightarrow \notin (\{Stuck\} \cup Fault\ '(-F)) \wedge \Gamma \vdash c \downarrow Normal\ s\}$ 
 $c$ 
   $\{t. \Gamma \vdash \langle c, Normal\ Z \rangle \Rightarrow Normal\ t\}, \{t. \Gamma \vdash \langle c, Normal\ Z \rangle \Rightarrow Abrupt\ t\}$ 
proof (rule validtI)
  fix  $s\ t$ 
  assume  $\Gamma \vdash \langle c, Normal\ s \rangle \Rightarrow t$ 
   $s \in \{s. s = Z \wedge \Gamma \vdash \langle c, Normal\ s \rangle \Rightarrow \notin (\{Stuck\} \cup Fault\ '(-F)) \wedge \Gamma \vdash c \downarrow Normal\ s\}$ 
   $t \notin Fault\ 'F$ 
  thus  $t \in Normal\ ' \{t. \Gamma \vdash \langle c, Normal\ Z \rangle \Rightarrow Normal\ t\} \cup$ 
   $Abrupt\ ' \{t. \Gamma \vdash \langle c, Normal\ Z \rangle \Rightarrow Abrupt\ t\}$ 
  apply (cases t)
  apply (auto simp add: final-notin-def)
done
next
  fix  $s$ 
  assume  $s \in \{s. s = Z \wedge \Gamma \vdash \langle c, Normal\ s \rangle \Rightarrow \notin (\{Stuck\} \cup Fault\ '(-F)) \wedge \Gamma \vdash c \downarrow Normal\ s\}$ 
  thus  $\Gamma \vdash c \downarrow Normal\ s$ 
  by blast

```



**qed**

The consequence rule where the existential  $Z$  is instantiated to  $s$ . Usefull in proof of *MGT-lemma*.

**lemma** *ConseqMGT*:

**assumes** *modif*:  $\forall Z::'a. \Gamma, \Theta \vdash_{t/F} (P' Z::'a \text{ assn}) \ c \ (Q' Z), (A' Z)$

**assumes** *impl*:  $\bigwedge s. s \in P \implies s \in P' s \wedge (\forall t. t \in Q' s \longrightarrow t \in Q) \wedge$   
 $(\forall t. t \in A' s \longrightarrow t \in A)$

**shows**  $\Gamma, \Theta \vdash_{t/F} P \ c \ Q, A$

**using** *impl*

**by**  $-$  (*rule conseq* [*OF modif*], *blast*)

**lemma** *MGT-implies-complete*:

**assumes** *MGT*:  $\forall Z. \Gamma, \{\} \vdash_{t/F} \{s. s=Z \wedge \Gamma \vdash \langle c, \text{Normal } s \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault ' (-F)}) \wedge$

$\Gamma \vdash c \downarrow \text{Normal } s\}$

$c$

$\{t. \Gamma \vdash \langle c, \text{Normal } Z \rangle \Rightarrow \text{Normal } t\},$

$\{t. \Gamma \vdash \langle c, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\}$

**assumes** *valid*:  $\Gamma \models_{t/F} P \ c \ Q, A$

**shows**  $\Gamma, \{\} \vdash_{t/F} P \ c \ Q, A$

**using** *MGT*

**apply** (*rule ConseqMGT*)

**apply** (*insert valid*)

**apply** (*auto simp add: validt-def valid-def intro!: final-notinI*)

**done**

**lemma** *conseq-extract-state-indep-prop*:

**assumes** *state-indep-prop*:  $\forall s \in P. R$

**assumes** *to-show*:  $R \implies \Gamma, \Theta \vdash_{t/F} P \ c \ Q, A$

**shows**  $\Gamma, \Theta \vdash_{t/F} P \ c \ Q, A$

**apply** (*rule Conseq*)

**apply** (*clarify*)

**apply** (*rule-tac x=P in exI*)

**apply** (*rule-tac x=Q in exI*)

**apply** (*rule-tac x=A in exI*)

**using** *state-indep-prop to-show*

**by** *blast*

**lemma** *MGT-lemma*:

**assumes** *MGT-Calls*:

$\forall p \in \text{dom } \Gamma. \forall Z. \Gamma, \Theta \vdash_{t/F}$

$\{s. s=Z \wedge \Gamma \vdash \langle \text{Call } p, \text{Normal } s \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault ' (-F)}) \wedge$

$\Gamma \vdash (\text{Call } p) \downarrow \text{Normal } s\}$

$(\text{Call } p)$

$\{t. \Gamma \vdash \langle \text{Call } p, \text{Normal } Z \rangle \Rightarrow \text{Normal } t\},$

$\{t. \Gamma \vdash \langle \text{Call } p, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\}$

**shows**  $\bigwedge Z. \Gamma, \Theta \vdash_{t/F} \{s. s=Z \wedge \Gamma \vdash \langle c, \text{Normal } s \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault ' (-F)}) \wedge$

$\Gamma \vdash c \downarrow \text{Normal } s \}$

$c$

$\{t. \Gamma \vdash \langle c, \text{Normal } Z \rangle \Rightarrow \text{Normal } t\}, \{t. \Gamma \vdash \langle c, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\}$

**proof** (*induct c*)

**case** *Skip*

**show**  $\Gamma, \Theta \vdash_{t/F} \{s. s = Z \wedge \Gamma \vdash \langle \text{Skip}, \text{Normal } s \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F)) \wedge$

$\Gamma \vdash \text{Skip} \downarrow \text{Normal } s \}$

*Skip*

$\{t. \Gamma \vdash \langle \text{Skip}, \text{Normal } Z \rangle \Rightarrow \text{Normal } t\}, \{t. \Gamma \vdash \langle \text{Skip}, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\}$

*t*)

**by** (*rule hoaret.Skip [THEN conseqPre]*)

(*auto elim: exec-elim-cases simp add: final-notin-def*

*intro: exec.intros terminates.intros*)

**next**

**case** (*Basic f*)

**show**  $\Gamma, \Theta \vdash_{t/F} \{s. s = Z \wedge \Gamma \vdash \langle \text{Basic } f, \text{Normal } s \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F)) \wedge$

$\Gamma \vdash \text{Basic } f \downarrow \text{Normal } s \}$

*Basic f*

$\{t. \Gamma \vdash \langle \text{Basic } f, \text{Normal } Z \rangle \Rightarrow \text{Normal } t\},$

$\{t. \Gamma \vdash \langle \text{Basic } f, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\}$

**by** (*rule hoaret.Basic [THEN conseqPre]*)

(*auto elim: exec-elim-cases simp add: final-notin-def*

*intro: exec.intros terminates.intros*)

**next**

**case** (*Spec r*)

**show**  $\Gamma, \Theta \vdash_{t/F} \{s. s = Z \wedge \Gamma \vdash \langle \text{Spec } r, \text{Normal } s \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F)) \wedge$

$\Gamma \vdash \text{Spec } r \downarrow \text{Normal } s \}$

*Spec r*

$\{t. \Gamma \vdash \langle \text{Spec } r, \text{Normal } Z \rangle \Rightarrow \text{Normal } t\},$

$\{t. \Gamma \vdash \langle \text{Spec } r, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\}$

**apply** (*rule hoaret.Spec [THEN conseqPre]*)

**apply** (*clarsimp simp add: final-notin-def*)

**apply** (*case-tac*  $\exists t. (Z, t) \in r$ )

**apply** (*auto elim: exec-elim-cases simp add: final-notin-def intro: exec.intros*)

**done**

**next**

**case** (*Seq c1 c2*)

**have** *hyp-c1*:  $\forall Z. \Gamma, \Theta \vdash_{t/F} \{s. s = Z \wedge \Gamma \vdash \langle c1, \text{Normal } s \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F)) \wedge$

$(-F) \wedge$

$\Gamma \vdash c1 \downarrow \text{Normal } s \}$

*c1*

$\{t. \Gamma \vdash \langle c1, \text{Normal } Z \rangle \Rightarrow \text{Normal } t\},$

$\{t. \Gamma \vdash \langle c1, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\}$

**using** *Seq.hyps* **by** *iprover*

**have** *hyp-c2*:  $\forall Z. \Gamma, \Theta \vdash_{t/F} \{s. s = Z \wedge \Gamma \vdash \langle c2, \text{Normal } s \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F)) \wedge$

$(-F) \wedge$

$\Gamma \vdash c2 \downarrow \text{Normal } s \}$

*c2*

$$\begin{array}{l}
\{t. \Gamma \vdash \langle c2, Normal Z \rangle \Rightarrow Normal t\}, \\
\{t. \Gamma \vdash \langle c2, Normal Z \rangle \Rightarrow Abrupt t\} \\
\text{using } Seq.hyps \text{ by } iprover \\
\text{from } hyp-c1 \\
\text{have } \Gamma, \Theta \vdash_{t/F} \{s. s=Z \wedge \Gamma \vdash \langle Seq\ c1\ c2, Normal\ s \rangle \Rightarrow \notin(\{Stuck\} \cup Fault\ ' (-F))\} \\
\wedge \\
\Gamma \vdash Seq\ c1\ c2 \downarrow Normal\ s\} \ c1 \\
\{t. \Gamma \vdash \langle c1, Normal Z \rangle \Rightarrow Normal t \wedge \Gamma \vdash \langle c2, Normal t \rangle \Rightarrow \notin(\{Stuck\} \cup Fault\ ' (-F))\} \wedge \\
(-F) \wedge \\
\Gamma \vdash c2 \downarrow Normal\ t\}, \\
\{t. \Gamma \vdash \langle Seq\ c1\ c2, Normal Z \rangle \Rightarrow Abrupt t\} \\
\text{by (rule } ConseqMGT) \\
(\text{auto dest: } Seq-NoFaultStuckD1 \text{ [simplified] } Seq-NoFaultStuckD2 \text{ [simplified]} \\
\text{elim: } terminates-Normal-elim-cases \\
\text{intro: } exec.intros) \\
\text{thus } \Gamma, \Theta \vdash_{t/F} \{s. s=Z \wedge \Gamma \vdash \langle Seq\ c1\ c2, Normal\ s \rangle \Rightarrow \notin(\{Stuck\} \cup Fault\ ' (-F))\} \\
\wedge \\
\Gamma \vdash Seq\ c1\ c2 \downarrow Normal\ s\} \\
Seq\ c1\ c2 \\
\{t. \Gamma \vdash \langle Seq\ c1\ c2, Normal Z \rangle \Rightarrow Normal t\}, \\
\{t. \Gamma \vdash \langle Seq\ c1\ c2, Normal Z \rangle \Rightarrow Abrupt t\} \\
\text{proof (rule } hoaret.Seq) \\
\text{show } \Gamma, \Theta \vdash_{t/F} \{t. \Gamma \vdash \langle c1, Normal Z \rangle \Rightarrow Normal t \wedge \\
\Gamma \vdash \langle c2, Normal t \rangle \Rightarrow \notin(\{Stuck\} \cup Fault\ ' (-F)) \wedge \Gamma \vdash c2 \downarrow Normal \\
t\} \\
c2 \\
\{t. \Gamma \vdash \langle Seq\ c1\ c2, Normal Z \rangle \Rightarrow Normal t\}, \\
\{t. \Gamma \vdash \langle Seq\ c1\ c2, Normal Z \rangle \Rightarrow Abrupt t\} \\
\text{proof (rule } ConseqMGT \text{ [OF hyp-c2], safe)} \\
\text{fix } r\ t \\
\text{assume } \Gamma \vdash \langle c1, Normal Z \rangle \Rightarrow Normal\ r \ \Gamma \vdash \langle c2, Normal\ r \rangle \Rightarrow Normal\ t \\
\text{then show } \Gamma \vdash \langle Seq\ c1\ c2, Normal Z \rangle \Rightarrow Normal\ t \\
\text{by (rule } exec.intros) \\
\text{next} \\
\text{fix } r\ t \\
\text{assume } \Gamma \vdash \langle c1, Normal Z \rangle \Rightarrow Normal\ r \ \Gamma \vdash \langle c2, Normal\ r \rangle \Rightarrow Abrupt\ t \\
\text{then show } \Gamma \vdash \langle Seq\ c1\ c2, Normal Z \rangle \Rightarrow Abrupt\ t \\
\text{by (rule } exec.intros) \\
\text{qed} \\
\text{qed} \\
\text{next} \\
\text{case (Cond } b\ c1\ c2) \\
\text{have } \forall Z. \Gamma, \Theta \vdash_{t/F} \{s. s=Z \wedge \Gamma \vdash \langle c1, Normal s \rangle \Rightarrow \notin(\{Stuck\} \cup Fault\ ' (-F)) \wedge \\
\Gamma \vdash c1 \downarrow Normal\ s\} \\
c1 \\
\{t. \Gamma \vdash \langle c1, Normal Z \rangle \Rightarrow Normal t\}, \\
\{t. \Gamma \vdash \langle c1, Normal Z \rangle \Rightarrow Abrupt t\} \\
\text{using } Cond.hyps \text{ by } iprover \\
\text{hence } \Gamma, \Theta \vdash_{t/F} \{s. s=Z \wedge \Gamma \vdash \langle Cond\ b\ c1\ c2, Normal\ s \rangle \Rightarrow \notin(\{Stuck\} \cup Fault\ ' (-F))\}
\end{array}$$

$(-F)) \wedge$   
 $\Gamma \vdash (\text{Cond } b \ c1 \ c2) \downarrow \text{Normal } s \} \cap b$   
 $c1$   
 $\{t. \Gamma \vdash \langle \text{Cond } b \ c1 \ c2, \text{Normal } Z \rangle \Rightarrow \text{Normal } t\},$   
 $\{t. \Gamma \vdash \langle \text{Cond } b \ c1 \ c2, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\}$   
**by** (rule *ConseqMGT*)  
*(fastforce simp add: final-notin-def intro: exec.CondTrue*  
*elim: terminates-Normal-elim-cases)*  
**moreover**  
**have**  $\forall Z. \Gamma, \Theta \vdash_{t/F} \{s. s=Z \wedge \Gamma \vdash \langle c2, \text{Normal } s \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F)) \wedge$   
 $\Gamma \vdash c2 \downarrow \text{Normal } s\}$   
 $c2$   
 $\{t. \Gamma \vdash \langle c2, \text{Normal } Z \rangle \Rightarrow \text{Normal } t\},$   
 $\{t. \Gamma \vdash \langle c2, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\}$   
**using** *Cond.hyps* **by** *iprover*  
**hence**  $\Gamma, \Theta \vdash_{t/F} \{s. s=Z \wedge \Gamma \vdash \langle \text{Cond } b \ c1 \ c2, \text{Normal } s \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F)) \wedge$   
 $(-F)) \wedge$   
 $\Gamma \vdash (\text{Cond } b \ c1 \ c2) \downarrow \text{Normal } s \} \cap -b$   
 $c2$   
 $\{t. \Gamma \vdash \langle \text{Cond } b \ c1 \ c2, \text{Normal } Z \rangle \Rightarrow \text{Normal } t\},$   
 $\{t. \Gamma \vdash \langle \text{Cond } b \ c1 \ c2, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\}$   
**by** (rule *ConseqMGT*)  
*(fastforce simp add: final-notin-def intro: exec.CondFalse*  
*elim: terminates-Normal-elim-cases)*  
**ultimately**  
**show**  $\Gamma, \Theta \vdash_{t/F} \{s. s=Z \wedge \Gamma \vdash \langle \text{Cond } b \ c1 \ c2, \text{Normal } s \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F)) \wedge$   
 $(-F)) \wedge$   
 $\Gamma \vdash (\text{Cond } b \ c1 \ c2) \downarrow \text{Normal } s\}$   
 $(\text{Cond } b \ c1 \ c2)$   
 $\{t. \Gamma \vdash \langle \text{Cond } b \ c1 \ c2, \text{Normal } Z \rangle \Rightarrow \text{Normal } t\},$   
 $\{t. \Gamma \vdash \langle \text{Cond } b \ c1 \ c2, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\}$   
**by** (rule *hoaret.Cond*)  
**next**  
**case** (*While b c*)  
**let**  $?unroll = (\{(s,t). s \in b \wedge \Gamma \vdash \langle c, \text{Normal } s \rangle \Rightarrow \text{Normal } t\})^*$   
**let**  $?P' = \lambda Z. \{t. (Z,t) \in ?unroll \wedge$   
 $(\forall e. (Z,e) \in ?unroll \longrightarrow e \in b$   
 $\longrightarrow \Gamma \vdash \langle c, \text{Normal } e \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F)) \wedge$   
 $(\forall u. \Gamma \vdash \langle c, \text{Normal } e \rangle \Rightarrow \text{Abrupt } u \longrightarrow$   
 $\Gamma \vdash \langle \text{While } b \ c, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } u)) \wedge$   
 $\Gamma \vdash (\text{While } b \ c) \downarrow \text{Normal } t\}$   
**let**  $?A = \lambda Z. \{t. \Gamma \vdash \langle \text{While } b \ c, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\}$   
**let**  $?r = \{(t,s). \Gamma \vdash (\text{While } b \ c) \downarrow \text{Normal } s \wedge s \in b \wedge$   
 $\Gamma \vdash \langle c, \text{Normal } s \rangle \Rightarrow \text{Normal } t\}$   
**show**  $\Gamma, \Theta \vdash_{t/F} \{s. s=Z \wedge \Gamma \vdash \langle \text{While } b \ c, \text{Normal } s \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F))$   
 $\wedge$   
 $\Gamma \vdash (\text{While } b \ c) \downarrow \text{Normal } s\}$   
 $(\text{While } b \ c)$   
 $\{t. \Gamma \vdash \langle \text{While } b \ c, \text{Normal } Z \rangle \Rightarrow \text{Normal } t\},$

$\{t. \Gamma \vdash \langle \text{While } b \ c, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\}$   
**proof** (rule *ConseqMGT* [where  $?P' = \lambda Z. ?P' Z$   
**and**  $?Q' = \lambda Z. ?P' Z \cap - b$ ])  
**have** *wf-r*: *wf ?r* **by** (rule *wf-terminates-while*)  
**show**  $\forall Z. \Gamma, \Theta \vdash_{t/F} (?P' Z) (\text{While } b \ c) (?P' Z \cap - b), (?A Z)$   
**proof** (rule *allI*, rule *hoaret.While* [*OF wf-r*])  
**fix**  $Z$   
**from** *While*  
**have** *hyp-c*:  $\forall Z. \Gamma, \Theta \vdash_{t/F} \{s. s = Z \wedge \Gamma \vdash \langle c, \text{Normal } s \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F)) \wedge$   
 $(-F)) \wedge$   
 $\Gamma \vdash c \downarrow \text{Normal } s\}$   
 $c$   
 $\{t. \Gamma \vdash \langle c, \text{Normal } Z \rangle \Rightarrow \text{Normal } t\},$   
 $\{t. \Gamma \vdash \langle c, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\}$  **by** *iprover*  
**show**  $\forall \sigma. \Gamma, \Theta \vdash_{t/F} (\{\sigma\} \cap ?P' Z \cap b) \ c$   
 $(\{t. (t, \sigma) \in ?r\} \cap ?P' Z), (?A Z)$   
**proof** (rule *allI*, rule *ConseqMGT* [*OF hyp-c*])  
**fix**  $\sigma \ s$   
**assume**  $s \in \{\sigma\} \cap$   
 $\{t. (Z, t) \in ?unroll \wedge$   
 $(\forall e. (Z, e) \in ?unroll \longrightarrow e \in b$   
 $\longrightarrow \Gamma \vdash \langle c, \text{Normal } e \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F)) \wedge$   
 $(\forall u. \Gamma \vdash \langle c, \text{Normal } e \rangle \Rightarrow \text{Abrupt } u \longrightarrow$   
 $\Gamma \vdash \langle \text{While } b \ c, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } u)) \wedge$   
 $\Gamma \vdash (\text{While } b \ c) \downarrow \text{Normal } t\}$   
 $\cap b$   
**then obtain**  
*s-eq-σ*:  $s = \sigma$  **and**  
*Z-s-unroll*:  $(Z, s) \in ?unroll$  **and**  
*noabort*:  $\forall e. (Z, e) \in ?unroll \longrightarrow e \in b$   
 $\longrightarrow \Gamma \vdash \langle c, \text{Normal } e \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F)) \wedge$   
 $(\forall u. \Gamma \vdash \langle c, \text{Normal } e \rangle \Rightarrow \text{Abrupt } u \longrightarrow$   
 $\Gamma \vdash \langle \text{While } b \ c, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } u)$  **and**  
*while-term*:  $\Gamma \vdash (\text{While } b \ c) \downarrow \text{Normal } s$  **and**  
*s-in-b*:  $s \in b$   
**by** *blast*  
**show**  $s \in \{t. t = s \wedge \Gamma \vdash \langle c, \text{Normal } t \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F)) \wedge$   
 $\Gamma \vdash c \downarrow \text{Normal } t\} \wedge$   
 $(\forall t. t \in \{t. \Gamma \vdash \langle c, \text{Normal } s \rangle \Rightarrow \text{Normal } t\} \longrightarrow$   
 $t \in \{t. (t, \sigma) \in ?r\} \cap$   
 $\{t. (Z, t) \in ?unroll \wedge$   
 $(\forall e. (Z, e) \in ?unroll \longrightarrow e \in b$   
 $\longrightarrow \Gamma \vdash \langle c, \text{Normal } e \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F)) \wedge$   
 $(\forall u. \Gamma \vdash \langle c, \text{Normal } e \rangle \Rightarrow \text{Abrupt } u \longrightarrow$   
 $\Gamma \vdash \langle \text{While } b \ c, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } u)) \wedge$   
 $\Gamma \vdash (\text{While } b \ c) \downarrow \text{Normal } t\}) \wedge$   
 $(\forall t. t \in \{t. \Gamma \vdash \langle c, \text{Normal } s \rangle \Rightarrow \text{Abrupt } t\} \longrightarrow$   
 $t \in \{t. \Gamma \vdash \langle \text{While } b \ c, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\})$   
**(is ?C1  $\wedge$  ?C2  $\wedge$  ?C3)**

**proof** (*intro conj1*)  
**from** *Z-s-unroll noabort s-in-b while-term* **show** ?C1  
**by** (*blast elim: terminates-Normal-elim-cases*)  
**next**  
{  
**fix** *t*  
**assume** *s-t*:  $\Gamma \langle c, \text{Normal } s \rangle \Rightarrow \text{Normal } t$   
**with** *s-eq-σ while-term s-in-b* **have**  $(t, \sigma) \in ?r$   
**by** *blast*  
**moreover**  
**from** *Z-s-unroll s-t s-in-b*  
**have**  $(Z, t) \in ?unroll$   
**by** (*blast intro: rtrancl-into-rtrancl*)  
**moreover from** *while-term s-t s-in-b*  
**have**  $\Gamma \vdash (\text{While } b \ c) \downarrow \text{Normal } t$   
**by** (*blast elim: terminates-Normal-elim-cases*)  
**moreover note** *noabort*  
**ultimately**  
**have**  $(t, \sigma) \in ?r \wedge (Z, t) \in ?unroll \wedge$   
 $(\forall e. (Z, e) \in ?unroll \longrightarrow e \in b$   
 $\longrightarrow \Gamma \vdash \langle c, \text{Normal } e \rangle \Rightarrow \notin (\{\text{Stuck}\} \cup \text{Fault } '(-F))) \wedge$   
 $(\forall u. \Gamma \vdash \langle c, \text{Normal } e \rangle \Rightarrow \text{Abrupt } u \longrightarrow$   
 $\Gamma \vdash \langle \text{While } b \ c, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } u)) \wedge$   
 $\Gamma \vdash (\text{While } b \ c) \downarrow \text{Normal } t$   
**by** *iprover*  
}  
**then show** ?C2 **by** *blast*  
**next**  
{  
**fix** *t*  
**assume** *s-t*:  $\Gamma \vdash \langle c, \text{Normal } s \rangle \Rightarrow \text{Abrupt } t$   
**from** *Z-s-unroll noabort s-t s-in-b*  
**have**  $\Gamma \vdash \langle \text{While } b \ c, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t$   
**by** *blast*  
**} thus** ?C3 **by** *simp*  
**qed**  
**qed**  
**qed**  
**next**  
**fix** *s*  
**assume** *P*:  $s \in \{s. s=Z \wedge \Gamma \vdash \langle \text{While } b \ c, \text{Normal } s \rangle \Rightarrow \notin (\{\text{Stuck}\} \cup \text{Fault } '(-F))\}$   
 $\wedge$   
 $\Gamma \vdash \text{While } b \ c \downarrow \text{Normal } s\}$   
**hence** *WhileNoFault*:  $\Gamma \vdash \langle \text{While } b \ c, \text{Normal } Z \rangle \Rightarrow \notin (\{\text{Stuck}\} \cup \text{Fault } '(-F))$   
**by** *auto*  
**show**  $s \in ?P' \ s \wedge$   
 $(\forall t. t \in (?P' \ s \cap - \ b) \longrightarrow$   
 $t \in \{t. \Gamma \vdash \langle \text{While } b \ c, \text{Normal } Z \rangle \Rightarrow \text{Normal } t\}) \wedge$   
 $(\forall t. t \in ?A \ s \longrightarrow t \in ?A \ Z)$

```

proof (intro conjI)
{
  fix e
  assume  $(Z, e) \in ?unroll\ e \in b$ 
  from this WhileNoFault
  have  $\Gamma \vdash \langle c, Normal\ e \rangle \Rightarrow \notin(\{Stuck\} \cup Fault\ '(-F)) \wedge$ 
     $(\forall u. \Gamma \vdash \langle c, Normal\ e \rangle \Rightarrow Abrupt\ u \longrightarrow$ 
       $\Gamma \vdash \langle While\ b\ c, Normal\ Z \rangle \Rightarrow Abrupt\ u)$  (is ?Prop Z e)
  proof (induct rule: converse-rtrancl-induct [consumes 1])
    assume e-in-b: e ∈ b
    assume WhileNoFault:  $\Gamma \vdash \langle While\ b\ c, Normal\ e \rangle \Rightarrow \notin(\{Stuck\} \cup Fault\ '(-F))$ 
    (-F))
    with e-in-b WhileNoFault
    have cNoFault:  $\Gamma \vdash \langle c, Normal\ e \rangle \Rightarrow \notin(\{Stuck\} \cup Fault\ '(-F))$ 
      by (auto simp add: final-notin-def intro: exec.intros)
    moreover
    {
      fix u assume  $\Gamma \vdash \langle c, Normal\ e \rangle \Rightarrow Abrupt\ u$ 
      with e-in-b have  $\Gamma \vdash \langle While\ b\ c, Normal\ e \rangle \Rightarrow Abrupt\ u$ 
      by (blast intro: exec.intros)
    }
    ultimately
    show ?Prop e e
    by iprover
  next
  fix Z r
  assume e-in-b: e ∈ b
  assume WhileNoFault:  $\Gamma \vdash \langle While\ b\ c, Normal\ Z \rangle \Rightarrow \notin(\{Stuck\} \cup Fault\ '(-F))$ 
  (-F))
  assume hyp:  $[[e \in b; \Gamma \vdash \langle While\ b\ c, Normal\ r \rangle \Rightarrow \notin(\{Stuck\} \cup Fault\ '(-F))]]$ 
     $\implies ?Prop\ r\ e$ 
  assume Z-r:
     $(Z, r) \in \{(Z, r). Z \in b \wedge \Gamma \vdash \langle c, Normal\ Z \rangle \Rightarrow Normal\ r\}$ 
  with WhileNoFault
  have  $\Gamma \vdash \langle While\ b\ c, Normal\ r \rangle \Rightarrow \notin(\{Stuck\} \cup Fault\ '(-F))$ 
    by (auto simp add: final-notin-def intro: exec.intros)
  from hyp [OF e-in-b this] obtain
    cNoFault:  $\Gamma \vdash \langle c, Normal\ e \rangle \Rightarrow \notin(\{Stuck\} \cup Fault\ '(-F))$  and
    Abrupt-r:  $\forall u. \Gamma \vdash \langle c, Normal\ e \rangle \Rightarrow Abrupt\ u \longrightarrow$ 
       $\Gamma \vdash \langle While\ b\ c, Normal\ r \rangle \Rightarrow Abrupt\ u$ 
  by simp

  {
    fix u assume  $\Gamma \vdash \langle c, Normal\ e \rangle \Rightarrow Abrupt\ u$ 
    with Abrupt-r have  $\Gamma \vdash \langle While\ b\ c, Normal\ r \rangle \Rightarrow Abrupt\ u$  by simp
    moreover from Z-r obtain
       $Z \in b\ \Gamma \vdash \langle c, Normal\ Z \rangle \Rightarrow Normal\ r$ 
    by simp
    ultimately have  $\Gamma \vdash \langle While\ b\ c, Normal\ Z \rangle \Rightarrow Abrupt\ u$ 
  }
}

```

```

      by (blast intro: exec.intros)
    }
    with cNoFault show ?Prop Z e
      by iprover
  qed
}
with P show s ∈ ?P' s
  by blast
next
{
  fix t
  assume termination: t ∉ b
  assume (Z, t) ∈ ?unroll
  hence  $\Gamma \vdash \langle \text{While } b \ c, \text{Normal } Z \rangle \Rightarrow \text{Normal } t$ 
  proof (induct rule: converse-rtrancl-induct [consumes 1])
    from termination
    show  $\Gamma \vdash \langle \text{While } b \ c, \text{Normal } t \rangle \Rightarrow \text{Normal } t$ 
      by (blast intro: exec.WhileFalse)
  next
  fix Z r
  assume first-body:
     $(Z, r) \in \{(s, t). s \in b \wedge \Gamma \vdash \langle c, \text{Normal } s \rangle \Rightarrow \text{Normal } t\}$ 
  assume (r, t) ∈ ?unroll
  assume rest-loop:  $\Gamma \vdash \langle \text{While } b \ c, \text{Normal } r \rangle \Rightarrow \text{Normal } t$ 
  show  $\Gamma \vdash \langle \text{While } b \ c, \text{Normal } Z \rangle \Rightarrow \text{Normal } t$ 
  proof -
    from first-body obtain
       $Z \in b \ \Gamma \vdash \langle c, \text{Normal } Z \rangle \Rightarrow \text{Normal } r$ 
    by fast
    moreover
    from rest-loop have
       $\Gamma \vdash \langle \text{While } b \ c, \text{Normal } r \rangle \Rightarrow \text{Normal } t$ 
    by fast
    ultimately show  $\Gamma \vdash \langle \text{While } b \ c, \text{Normal } Z \rangle \Rightarrow \text{Normal } t$ 
      by (rule exec.WhileTrue)
  qed
  qed
}
with P
show  $(\forall t. t \in (?P' s \cap - b) \longrightarrow t \in \{t. \Gamma \vdash \langle \text{While } b \ c, \text{Normal } Z \rangle \Rightarrow \text{Normal } t\})$ 
  by blast
next
from P show  $\forall t. t \in ?A s \longrightarrow t \in ?A Z$ 
  by simp
qed
qed
next
case (Call p)

```



**from** *noStuck-Call*  
**have**  $\forall s \in \{s. s=Z \wedge \Gamma \vdash \langle \text{Call } p, \text{Normal } s \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F)) \wedge \Gamma \vdash \text{Call } p \downarrow \text{Normal } s\}.$   
 $p \in \text{dom } \Gamma$   
**by** (*fastforce simp add: final-notin-def*)  
**then show** *?case*  
**proof** (*rule conseq-extract-state-indep-prop*)  
**assume** *p-defined: p ∈ dom Γ*  
**with** *MGT-Calls show*  
 $\Gamma, \Theta \vdash_{t/F} \{s. s=Z \wedge$   
 $\Gamma \vdash \langle \text{Call } p, \text{Normal } s \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F)) \wedge$   
 $\Gamma \vdash \text{Call } p \downarrow \text{Normal } s\}$   
 $(\text{Call } p)$   
 $\{t. \Gamma \vdash \langle \text{Call } p, \text{Normal } Z \rangle \Rightarrow \text{Normal } t\},$   
 $\{t. \Gamma \vdash \langle \text{Call } p, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\}$   
**by** (*auto*)  
**qed**  
**next**  
**case** (*DynCom c*)  
**have** *hyp:*  
 $\wedge s'. \forall Z. \Gamma, \Theta \vdash_{t/F} \{s. s = Z \wedge \Gamma \vdash \langle c \ s', \text{Normal } s \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F))$   
 $\wedge$   
 $\Gamma \vdash c \ s' \downarrow \text{Normal } s\} \ c \ s'$   
 $\{t. \Gamma \vdash \langle c \ s', \text{Normal } Z \rangle \Rightarrow \text{Normal } t\}, \{t. \Gamma \vdash \langle c \ s', \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\}$   
**using** *DynCom by simp*  
**have** *hyp':*  
 $\Gamma, \Theta \vdash_{t/F} \{s. s = Z \wedge \Gamma \vdash \langle \text{DynCom } c, \text{Normal } s \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F)) \wedge$   
 $\Gamma \vdash \text{DynCom } c \downarrow \text{Normal } s\}$   
 $(c \ Z)$   
 $\{t. \Gamma \vdash \langle \text{DynCom } c, \text{Normal } Z \rangle \Rightarrow \text{Normal } t\}, \{t. \Gamma \vdash \langle \text{DynCom } c, \text{Normal } Z \rangle \Rightarrow$   
 $\text{Abrupt } t\}$   
**by** (*rule ConseqMGT [OF hyp]*)  
*(fastforce simp add: final-notin-def intro: exec.intros*  
*elim: terminates-Normal-elim-cases)*  
**show**  $\Gamma, \Theta \vdash_{t/F} \{s. s=Z \wedge \Gamma \vdash \langle \text{DynCom } c, \text{Normal } s \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F))$   
 $\wedge$   
 $\Gamma \vdash \text{DynCom } c \downarrow \text{Normal } s\}$   
 $\text{DynCom } c$   
 $\{t. \Gamma \vdash \langle \text{DynCom } c, \text{Normal } Z \rangle \Rightarrow \text{Normal } t\},$   
 $\{t. \Gamma \vdash \langle \text{DynCom } c, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\}$   
**apply** (*rule hoaret.DynCom*)  
**apply** (*clarsimp*)  
**apply** (*rule hyp' [simplified]*)  
**done**  
**next**  
**case** (*Guard f g c*)  
**have** *hyp-c: ∀ Z. Γ, Θ ⊢<sub>t/F</sub> {s. s=Z ∧ Γ ⊢ ⟨c, Normal s⟩ ⇒ ∉({Stuck} ∪ Fault '(-F)) ∧*  
 $\Gamma \vdash c \downarrow \text{Normal } s\}$

$$\begin{array}{l}
\begin{array}{c}
c \\
\{t. \Gamma \vdash \langle c, Normal Z \rangle \Rightarrow Normal t\}, \\
\{t. \Gamma \vdash \langle c, Normal Z \rangle \Rightarrow Abrupt t\}
\end{array} \\
\text{using } Guard \text{ by } iprover \\
\text{show } \Gamma, \Theta \vdash_{t/F} \{s. s = Z \wedge \Gamma \vdash \langle Guard f g c, Normal s \rangle \Rightarrow \notin(\{Stuck\} \cup Fault \text{ ' } \\
(-F)) \wedge \\
\begin{array}{c}
\Gamma \vdash Guard f g c \downarrow Normal s\} \\
Guard f g c \\
\{t. \Gamma \vdash \langle Guard f g c, Normal Z \rangle \Rightarrow Normal t\}, \\
\{t. \Gamma \vdash \langle Guard f g c, Normal Z \rangle \Rightarrow Abrupt t\}
\end{array} \\
\text{proof (cases } f \in F) \\
\text{case } True \\
\text{from } hyp-c \\
\text{have } \Gamma, \Theta \vdash_{t/F} (g \cap \{s. s=Z \wedge \\
\begin{array}{c}
\Gamma \vdash \langle Guard f g c, Normal s \rangle \Rightarrow \notin(\{Stuck\} \cup Fault \text{ ' } (-F)) \wedge \\
\Gamma \vdash Guard f g c \downarrow Normal s\}) \\
c \\
\{t. \Gamma \vdash \langle Guard f g c, Normal Z \rangle \Rightarrow Normal t\}, \\
\{t. \Gamma \vdash \langle Guard f g c, Normal Z \rangle \Rightarrow Abrupt t\}
\end{array} \\
\text{apply (rule } ConseqMGT) \\
\text{apply (insert } True) \\
\text{apply (auto simp add: final-notin-def intro: exec.intros \\
elim: terminates-Normal-elim-cases) \\
done \\
from } True \text{ this} \\
\text{show } ?thesis \\
\text{by (rule } conseqPre [OF Guarantee]) \text{ auto} \\
\text{next} \\
\text{case } False \\
\text{from } hyp-c \\
\text{have } \Gamma, \Theta \vdash_{t/F} (g \cap \{s. s \in g \wedge s=Z \wedge \\
\begin{array}{c}
\Gamma \vdash \langle Guard f g c, Normal s \rangle \Rightarrow \notin(\{Stuck\} \cup Fault \text{ ' } (-F)) \wedge \\
\Gamma \vdash Guard f g c \downarrow Normal s\} ) \\
c \\
\{t. \Gamma \vdash \langle Guard f g c, Normal Z \rangle \Rightarrow Normal t\}, \\
\{t. \Gamma \vdash \langle Guard f g c, Normal Z \rangle \Rightarrow Abrupt t\}
\end{array} \\
\text{apply (rule } ConseqMGT) \\
\text{apply clarify} \\
\text{apply (frule } Guard-noFaultStuckD [OF - False]) \\
\text{apply (auto simp add: final-notin-def intro: exec.intros \\
elim: terminates-Normal-elim-cases) \\
done \\
then show } ?thesis \\
\text{apply (rule } conseqPre [OF hoaret.Guard]) \\
\text{apply clarify} \\
\text{apply (frule } Guard-noFaultStuckD [OF - False]) \\
\text{apply auto} \\
\text{done} \\
\text{qed}
\end{array}$$

**next**  
**case** *Throw*  
**show**  $\Gamma, \Theta \vdash_{t/F} \{s. s = Z \wedge \Gamma \vdash \langle \text{Throw}, \text{Normal } s \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F))\}$   
 $\wedge$   
 $\Gamma \vdash \text{Throw} \downarrow \text{Normal } s$   
 $\text{Throw}$   
 $\{t. \Gamma \vdash \langle \text{Throw}, \text{Normal } Z \rangle \Rightarrow \text{Normal } t\},$   
 $\{t. \Gamma \vdash \langle \text{Throw}, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\}$   
**by** (*rule conseqPre [OF hoaret.Throw]*)  
*(blast intro: exec.intros terminates.intros)*  
**next**  
**case** (*Catch*  $c_1$   $c_2$ )  
**have**  $\forall Z. \Gamma, \Theta \vdash_{t/F} \{s. s = Z \wedge \Gamma \vdash \langle c_1, \text{Normal } s \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F))\}$   
 $\wedge$   
 $\Gamma \vdash c_1 \downarrow \text{Normal } s$   
 $c_1$   
 $\{t. \Gamma \vdash \langle c_1, \text{Normal } Z \rangle \Rightarrow \text{Normal } t\},$   
 $\{t. \Gamma \vdash \langle c_1, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\}$   
**using** *Catch.hyps by iprover*  
**hence**  $\Gamma, \Theta \vdash_{t/F} \{s. s = Z \wedge \Gamma \vdash \langle \text{Catch } c_1 \ c_2, \text{Normal } s \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F))\}$   
 $(-F) \wedge$   
 $\Gamma \vdash \text{Catch } c_1 \ c_2 \downarrow \text{Normal } s$   
 $c_1$   
 $\{t. \Gamma \vdash \langle \text{Catch } c_1 \ c_2, \text{Normal } Z \rangle \Rightarrow \text{Normal } t\},$   
 $\{t. \Gamma \vdash \langle c_1, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t \wedge \Gamma \vdash c_2 \downarrow \text{Normal } t \wedge$   
 $\Gamma \vdash \langle c_2, \text{Normal } t \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F))\}$   
**by** (*rule ConseqMGT*)  
*(fastforce intro: exec.intros terminates.intros*  
*elim: terminates-Normal-elim-cases*  
*simp add: final-notin-def)*  
**moreover**  
**have**  
 $\forall Z. \Gamma, \Theta \vdash_{t/F} \{s. s = Z \wedge \Gamma \vdash \langle c_2, \text{Normal } s \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F)) \wedge$   
 $\Gamma \vdash c_2 \downarrow \text{Normal } s\}$   $c_2$   
 $\{t. \Gamma \vdash \langle c_2, \text{Normal } Z \rangle \Rightarrow \text{Normal } t\},$   
 $\{t. \Gamma \vdash \langle c_2, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\}$   
**using** *Catch.hyps by iprover*  
**hence**  $\Gamma, \Theta \vdash_{t/F} \{s. \Gamma \vdash \langle c_1, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } s \wedge \Gamma \vdash c_2 \downarrow \text{Normal } s \wedge$   
 $\Gamma \vdash \langle c_2, \text{Normal } s \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F))\}$   
 $c_2$   
 $\{t. \Gamma \vdash \langle \text{Catch } c_1 \ c_2, \text{Normal } Z \rangle \Rightarrow \text{Normal } t\},$   
 $\{t. \Gamma \vdash \langle \text{Catch } c_1 \ c_2, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\}$   
**by** (*rule ConseqMGT*)  
*(fastforce intro: exec.intros terminates.intros*  
*simp add: noFault-def')*  
**ultimately**  
**show**  $\Gamma, \Theta \vdash_{t/F} \{s. s = Z \wedge \Gamma \vdash \langle \text{Catch } c_1 \ c_2, \text{Normal } s \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F)) \wedge$   
 $(-F) \wedge$   
 $\Gamma \vdash \text{Catch } c_1 \ c_2 \downarrow \text{Normal } s\}$

$Catch\ c_1\ c_2$   
 $\{t. \Gamma \vdash \langle Catch\ c_1\ c_2, Normal\ Z \rangle \Rightarrow Normal\ t\},$   
 $\{t. \Gamma \vdash \langle Catch\ c_1\ c_2, Normal\ Z \rangle \Rightarrow Abrupt\ t\}$

by (rule hoaret.Catch )

qed

**lemma** *Call-lemma'*:

**assumes** *Call-hyp*:

$$\forall q \in dom\ \Gamma. \forall Z. \Gamma, \Theta \vdash_{t/F} \{s. s=Z \wedge \Gamma \vdash \langle Call\ q, Normal\ s \rangle \Rightarrow \notin(\{Stuck\} \cup Fault\ '(-F)) \wedge (-F)\} \wedge$$

$$\Gamma \vdash Call\ q \downarrow Normal\ s \wedge ((s,q),(\sigma,p)) \in termi-call-steps\ \Gamma\}$$

$$(Call\ q)$$

$$\{t. \Gamma \vdash \langle Call\ q, Normal\ Z \rangle \Rightarrow Normal\ t\},$$

$$\{t. \Gamma \vdash \langle Call\ q, Normal\ Z \rangle \Rightarrow Abrupt\ t\}$$

**shows**  $\bigwedge Z. \Gamma, \Theta \vdash_{t/F}$

$$\{s. s=Z \wedge \Gamma \vdash \langle c, Normal\ s \rangle \Rightarrow \notin(\{Stuck\} \cup Fault\ '(-F)) \wedge \Gamma \vdash Call\ p \downarrow Normal\ \sigma \wedge$$

$$(\exists c'. \Gamma \vdash (Call\ p, Normal\ \sigma) \rightarrow^+ (c', Normal\ s) \wedge c \in redexes\ c')\}$$

$$c$$

$$\{t. \Gamma \vdash \langle c, Normal\ Z \rangle \Rightarrow Normal\ t\},$$

$$\{t. \Gamma \vdash \langle c, Normal\ Z \rangle \Rightarrow Abrupt\ t\}$$

**proof** (*induct c*)

**case** *Skip*

**show**  $\Gamma, \Theta \vdash_{t/F} \{s. s = Z \wedge \Gamma \vdash \langle Skip, Normal\ s \rangle \Rightarrow \notin(\{Stuck\} \cup Fault\ '(-F)) \wedge$

$$\Gamma \vdash Call\ p \downarrow Normal\ \sigma \wedge$$

$$(\exists c'. \Gamma \vdash (Call\ p, Normal\ \sigma) \rightarrow^+ (c', Normal\ s) \wedge Skip \in redexes\ c')\}$$

$$Skip$$

$$\{t. \Gamma \vdash \langle Skip, Normal\ Z \rangle \Rightarrow Normal\ t\},$$

$$\{t. \Gamma \vdash \langle Skip, Normal\ Z \rangle \Rightarrow Abrupt\ t\}$$

by (rule hoaret.Skip [THEN conseqPre]) (blast intro: exec.Skip)

**next**

**case** (*Basic f*)

**show**  $\Gamma, \Theta \vdash_{t/F} \{s. s=Z \wedge \Gamma \vdash \langle Basic\ f, Normal\ s \rangle \Rightarrow \notin(\{Stuck\} \cup Fault\ '(-F)) \wedge$

$$\Gamma \vdash Call\ p \downarrow Normal\ \sigma \wedge$$

$$(\exists c'. \Gamma \vdash (Call\ p, Normal\ \sigma) \rightarrow^+ (c', Normal\ s) \wedge$$

$$Basic\ f \in redexes\ c')\}$$

$$Basic\ f$$

$$\{t. \Gamma \vdash \langle Basic\ f, Normal\ Z \rangle \Rightarrow Normal\ t\},$$

$$\{t. \Gamma \vdash \langle Basic\ f, Normal\ Z \rangle \Rightarrow Abrupt\ t\}$$

by (rule hoaret.Basic [THEN conseqPre]) (blast intro: exec.Basic)

**next**

**case** (*Spec r*)

**show**  $\Gamma, \Theta \vdash_{t/F} \{s. s=Z \wedge \Gamma \vdash \langle Spec\ r, Normal\ s \rangle \Rightarrow \notin(\{Stuck\} \cup Fault\ '(-F)) \wedge$

$$\Gamma \vdash Call\ p \downarrow Normal\ \sigma \wedge$$

$$(\exists c'. \Gamma \vdash (Call\ p, Normal\ \sigma) \rightarrow^+ (c', Normal\ s) \wedge$$

$$Spec\ r \in redexes\ c')\}$$

$$Spec\ r$$

$$\{t. \Gamma \vdash \langle Spec\ r, Normal\ Z \rangle \Rightarrow Normal\ t\},$$

$\{t. \Gamma \vdash \langle \text{Spec } r, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\}$   
**apply** (*rule hoaret.Spec [THEN conseqPre]*)  
**apply** (*clarsimp*)  
**apply** (*case-tac*  $\exists t. (Z, t) \in r$ )  
**apply** (*auto elim: exec-elim-cases simp add: final-notin-def intro: exec.intros*)  
**done**  
**next**  
**case** (*Seq c1 c2*)  
**have** *hyp-c1*:  
 $\forall Z. \Gamma, \Theta \vdash_{t/F} \{s. s=Z \wedge \Gamma \vdash \langle c1, \text{Normal } s \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F)) \wedge$   
 $\Gamma \vdash \text{Call } p \downarrow \text{Normal } \sigma \wedge$   
 $(\exists c'. \Gamma \vdash (\text{Call } p, \text{Normal } \sigma) \rightarrow^+ (c', \text{Normal } s) \wedge c1 \in \text{redexes } c')\}$   
 $c1$   
 $\{t. \Gamma \vdash \langle c1, \text{Normal } Z \rangle \Rightarrow \text{Normal } t\},$   
 $\{t. \Gamma \vdash \langle c1, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\}$   
**using** *Seq.hyps by iprover*  
**have** *hyp-c2*:  
 $\forall Z. \Gamma, \Theta \vdash_{t/F} \{s. s=Z \wedge \Gamma \vdash \langle c2, \text{Normal } s \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F)) \wedge$   
 $\Gamma \vdash \text{Call } p \downarrow \text{Normal } \sigma \wedge$   
 $(\exists c'. \Gamma \vdash (\text{Call } p, \text{Normal } \sigma) \rightarrow^+ (c', \text{Normal } s) \wedge c2 \in \text{redexes } c')\}$   
 $c2$   
 $\{t. \Gamma \vdash \langle c2, \text{Normal } Z \rangle \Rightarrow \text{Normal } t\},$   
 $\{t. \Gamma \vdash \langle c2, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\}$   
**using** *Seq.hyps (2) by iprover*  
**have** *c1*:  $\Gamma, \Theta \vdash_{t/F} \{s. s=Z \wedge \Gamma \vdash \langle \text{Seq } c1 \text{ } c2, \text{Normal } s \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F)) \wedge$   
 $(-F)) \wedge$   
 $\Gamma \vdash \text{Call } p \downarrow \text{Normal } \sigma \wedge$   
 $(\exists c'. \Gamma \vdash (\text{Call } p, \text{Normal } \sigma) \rightarrow^+ (c', \text{Normal } s) \wedge$   
 $\text{Seq } c1 \text{ } c2 \in \text{redexes } c')\}$   
 $c1$   
 $\{t. \Gamma \vdash \langle c1, \text{Normal } Z \rangle \Rightarrow \text{Normal } t \wedge$   
 $\Gamma \vdash \langle c2, \text{Normal } t \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F)) \wedge$   
 $\Gamma \vdash \text{Call } p \downarrow \text{Normal } \sigma \wedge$   
 $(\exists c'. \Gamma \vdash (\text{Call } p, \text{Normal } \sigma) \rightarrow^+ (c', \text{Normal } t) \wedge$   
 $c2 \in \text{redexes } c')\},$   
 $\{t. \Gamma \vdash \langle \text{Seq } c1 \text{ } c2, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\}$   
**proof** (*rule ConseqMGT [OF hyp-c1], clarify, safe*)  
**assume**  $\Gamma \vdash \langle \text{Seq } c1 \text{ } c2, \text{Normal } Z \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F))$   
**thus**  $\Gamma \vdash \langle c1, \text{Normal } Z \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F))$   
**by** (*blast dest: Seq-NoFaultStuckD1*)  
**next**  
**fix** *c'*  
**assume** *steps-c'*:  $\Gamma \vdash (\text{Call } p, \text{Normal } \sigma) \rightarrow^+ (c', \text{Normal } Z)$   
**assume** *red*:  $\text{Seq } c1 \text{ } c2 \in \text{redexes } c'$   
**from** *redexes-subset [OF red] steps-c'*  
**show**  $\exists c'. \Gamma \vdash (\text{Call } p, \text{Normal } \sigma) \rightarrow^+ (c', \text{Normal } Z) \wedge c1 \in \text{redexes } c'$   
**by** (*auto iff: root-in-redexes*)  
**next**  
**fix** *t*

```

assume  $\Gamma \vdash \langle \text{Seq } c1 \ c2, \text{Normal } Z \rangle \Rightarrow \notin (\{\text{Stuck}\} \cup \text{Fault } '(-F))$ 
       $\Gamma \vdash \langle c1, \text{Normal } Z \rangle \Rightarrow \text{Normal } t$ 
thus  $\Gamma \vdash \langle c2, \text{Normal } t \rangle \Rightarrow \notin (\{\text{Stuck}\} \cup \text{Fault } '(-F))$ 
      by (blast dest: Seq-NoFaultStuckD2)
next
fix  $c' \ t$ 
assume  $\text{steps-}c': \Gamma \vdash (\text{Call } p, \text{Normal } \sigma) \rightarrow^+ (c', \text{Normal } Z)$ 
assume  $\text{red}: \text{Seq } c1 \ c2 \in \text{redexes } c'$ 
assume  $\text{exec-}c1: \Gamma \vdash \langle c1, \text{Normal } Z \rangle \Rightarrow \text{Normal } t$ 
show  $\exists c'. \Gamma \vdash (\text{Call } p, \text{Normal } \sigma) \rightarrow^+ (c', \text{Normal } t) \wedge c2 \in \text{redexes } c'$ 
proof –
  note  $\text{steps-}c'$ 
  also
  from  $\text{exec-impl-steps-Normal}$  [OF exec-c1]
  have  $\Gamma \vdash (c1, \text{Normal } Z) \rightarrow^* (\text{Skip}, \text{Normal } t)$ .
  from  $\text{steps-redexes-Seq}$  [OF this red]
  obtain  $c''$  where
     $\text{steps-}c'': \Gamma \vdash (c', \text{Normal } Z) \rightarrow^* (c'', \text{Normal } t)$  and
     $\text{Skip}: \text{Seq } \text{Skip } c2 \in \text{redexes } c''$ 
    by blast
  note  $\text{steps-}c''$ 
  also
  have  $\text{step-Skip}: \Gamma \vdash (\text{Seq } \text{Skip } c2, \text{Normal } t) \rightarrow (c2, \text{Normal } t)$ 
    by (rule step.SeqSkip)
  from  $\text{step-redexes}$  [OF step-Skip Skip]
  obtain  $c'''$  where
     $\text{step-}c''': \Gamma \vdash (c'', \text{Normal } t) \rightarrow (c''', \text{Normal } t)$  and
     $c2: c2 \in \text{redexes } c'''$ 
    by blast
  note  $\text{step-}c'''$ 
  finally show ?thesis
    using  $c2$ 
    by blast
  qed
next
fix  $t$ 
assume  $\Gamma \vdash \langle c1, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t$ 
thus  $\Gamma \vdash \langle \text{Seq } c1 \ c2, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t$ 
      by (blast intro: exec.intros)
qed
show  $\Gamma, \Theta \vdash_{t/F} \{s. s=Z \wedge \Gamma \vdash \langle \text{Seq } c1 \ c2, \text{Normal } s \rangle \Rightarrow \notin (\{\text{Stuck}\} \cup \text{Fault } '(-F))$ 
 $\wedge$ 
       $\Gamma \vdash \text{Call } p \downarrow \text{Normal } \sigma \wedge$ 
       $(\exists c'. \Gamma \vdash (\text{Call } p, \text{Normal } \sigma) \rightarrow^+ (c', \text{Normal } s) \wedge \text{Seq } c1 \ c2 \in \text{redexes}$ 
 $c')\}$ 
       $\text{Seq } c1 \ c2$ 
       $\{t. \Gamma \vdash \langle \text{Seq } c1 \ c2, \text{Normal } Z \rangle \Rightarrow \text{Normal } t\},$ 
       $\{t. \Gamma \vdash \langle \text{Seq } c1 \ c2, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\}$ 
      by (rule hoaret.Seq [OF c1 ConseqMGT] [OF hyp-c2]])
```

(blast intro: exec.intros)

**next**

**case** (*Cond b c1 c2*)

**have** *hyp-c1*:

$$\forall Z. \Gamma, \Theta \vdash_{t/F} \{s. s=Z \wedge \Gamma \vdash \langle c1, Normal \ s \rangle \Rightarrow \notin(\{Stuck\} \cup Fault \ '(-F)) \wedge$$

$$\Gamma \vdash Call \ p \downarrow Normal \ \sigma \wedge$$

$$(\exists c'. \Gamma \vdash (Call \ p, Normal \ \sigma) \rightarrow^+ (c', Normal \ s) \wedge c1 \in redexes \ c')\}$$

$$c1$$

$$\{t. \Gamma \vdash \langle c1, Normal \ Z \rangle \Rightarrow Normal \ t\},$$

$$\{t. \Gamma \vdash \langle c1, Normal \ Z \rangle \Rightarrow Abrupt \ t\}$$

**using** *Cond.hyps* **by** *iprover*

**have**

$$\Gamma, \Theta \vdash_{t/F} (\{s. s=Z \wedge \Gamma \vdash \langle Cond \ b \ c1 \ c2, Normal \ s \rangle \Rightarrow \notin(\{Stuck\} \cup Fault \ '(-F))$$

$$\wedge$$

$$\Gamma \vdash Call \ p \downarrow Normal \ \sigma \wedge$$

$$(\exists c'. \Gamma \vdash (Call \ p, Normal \ \sigma) \rightarrow^+ (c', Normal \ s) \wedge$$

$$Cond \ b \ c1 \ c2 \in redexes \ c')\}$$

$$\cap b)$$

$$c1$$

$$\{t. \Gamma \vdash \langle Cond \ b \ c1 \ c2, Normal \ Z \rangle \Rightarrow Normal \ t\},$$

$$\{t. \Gamma \vdash \langle Cond \ b \ c1 \ c2, Normal \ Z \rangle \Rightarrow Abrupt \ t\}$$

**proof** (*rule ConseqMGT [OF hyp-c1], safe*)

**assume**  $Z \in b \ \Gamma \vdash \langle Cond \ b \ c1 \ c2, Normal \ Z \rangle \Rightarrow \notin(\{Stuck\} \cup Fault \ '(-F))$

**thus**  $\Gamma \vdash \langle c1, Normal \ Z \rangle \Rightarrow \notin(\{Stuck\} \cup Fault \ '(-F))$

**by** (*auto simp add: final-notin-def intro: exec.CondTrue*)

**next**

**fix**  $c'$

**assume**  $b: Z \in b$

**assume** *steps-c'*:  $\Gamma \vdash (Call \ p, Normal \ \sigma) \rightarrow^+ (c', Normal \ Z)$

**assume** *redex-c'*:  $Cond \ b \ c1 \ c2 \in redexes \ c'$

**show**  $\exists c'. \Gamma \vdash (Call \ p, Normal \ \sigma) \rightarrow^+ (c', Normal \ Z) \wedge c1 \in redexes \ c'$

**proof** –

**note** *steps-c'*

**also**

**from**  $b$

**have**  $\Gamma \vdash (Cond \ b \ c1 \ c2, Normal \ Z) \rightarrow (c1, Normal \ Z)$

**by** (*rule step.CondTrue*)

**from** *step-redexes [OF this redex-c']* **obtain**  $c''$  **where**

*step-c''*:  $\Gamma \vdash (c', Normal \ Z) \rightarrow (c'', Normal \ Z)$  **and**

$c1: c1 \in redexes \ c''$

**by** *blast*

**note** *step-c''*

**finally show** *?thesis*

**using**  $c1$

**by** *blast*

**qed**

**next**

**fix**  $t$  **assume**  $Z \in b \ \Gamma \vdash \langle c1, Normal \ Z \rangle \Rightarrow Normal \ t$

**thus**  $\Gamma \vdash \langle Cond \ b \ c1 \ c2, Normal \ Z \rangle \Rightarrow Normal \ t$

by (*blast intro: exec.CondTrue*)  
**next**  
 fix  $t$  assume  $Z \in b \Gamma \vdash \langle c1, Normal Z \rangle \Rightarrow Abrupt t$   
 thus  $\Gamma \vdash \langle Cond b c1 c2, Normal Z \rangle \Rightarrow Abrupt t$   
 by (*blast intro: exec.CondTrue*)  
**qed**  
**moreover**  
**have** *hyp-c2*:  
 $\forall Z. \Gamma, \Theta \vdash_{t/F} \{s. s=Z \wedge \Gamma \vdash \langle c2, Normal s \rangle \Rightarrow \notin(\{Stuck\} \cup Fault \text{ ' } (-F)) \wedge$   
 $\Gamma \vdash Call p \downarrow Normal \sigma \wedge$   
 $(\exists c'. \Gamma \vdash (Call p, Normal \sigma) \rightarrow^+ (c', Normal s) \wedge c2 \in redexes c')\}$   
 $c2$   
 $\{t. \Gamma \vdash \langle c2, Normal Z \rangle \Rightarrow Normal t\},$   
 $\{t. \Gamma \vdash \langle c2, Normal Z \rangle \Rightarrow Abrupt t\}$   
**using** *Cond.hyps* **by** *iprover*  
**have**  
 $\Gamma, \Theta \vdash_{t/F} (\{s. s=Z \wedge \Gamma \vdash \langle Cond b c1 c2, Normal s \rangle \Rightarrow \notin(\{Stuck\} \cup Fault \text{ ' } (-F))$   
 $\wedge$   
 $\Gamma \vdash Call p \downarrow Normal \sigma \wedge$   
 $(\exists c'. \Gamma \vdash (Call p, Normal \sigma) \rightarrow^+ (c', Normal s) \wedge$   
 $Cond b c1 c2 \in redexes c')\}$   
 $\cap -b)$   
 $c2$   
 $\{t. \Gamma \vdash \langle Cond b c1 c2, Normal Z \rangle \Rightarrow Normal t\},$   
 $\{t. \Gamma \vdash \langle Cond b c1 c2, Normal Z \rangle \Rightarrow Abrupt t\}$   
**proof** (*rule ConseqMGT [OF hyp-c2], safe*)  
 assume  $Z \notin b \Gamma \vdash \langle Cond b c1 c2, Normal Z \rangle \Rightarrow \notin(\{Stuck\} \cup Fault \text{ ' } (-F))$   
 thus  $\Gamma \vdash \langle c2, Normal Z \rangle \Rightarrow \notin(\{Stuck\} \cup Fault \text{ ' } (-F))$   
 by (*auto simp add: final-notin-def intro: exec.CondFalse*)  
**next**  
**fix**  $c'$   
**assume**  $b: Z \notin b$   
**assume** *steps-c'*:  $\Gamma \vdash (Call p, Normal \sigma) \rightarrow^+ (c', Normal Z)$   
**assume** *redex-c'*:  $Cond b c1 c2 \in redexes c'$   
**show**  $\exists c'. \Gamma \vdash (Call p, Normal \sigma) \rightarrow^+ (c', Normal Z) \wedge c2 \in redexes c'$   
**proof** –  
**note** *steps-c'*  
**also**  
**from**  $b$   
**have**  $\Gamma \vdash (Cond b c1 c2, Normal Z) \rightarrow (c2, Normal Z)$   
**by** (*rule step.CondFalse*)  
**from** *step-redexes [OF this redex-c']* **obtain**  $c''$  **where**  
 $step-c'': \Gamma \vdash (c', Normal Z) \rightarrow (c'', Normal Z)$  **and**  
 $c1: c2 \in redexes c''$   
**by** *blast*  
**note** *step-c''*  
**finally show** *?thesis*  
**using**  $c1$   
**by** *blast*



**qed**  
**next**  
**fix**  $t$  **assume**  $Z \notin b \Gamma \vdash \langle c2, Normal Z \rangle \Rightarrow Normal t$   
**thus**  $\Gamma \vdash \langle Cond b c1 c2, Normal Z \rangle \Rightarrow Normal t$   
**by** (*blast intro: exec.CondFalse*)  
**next**  
**fix**  $t$  **assume**  $Z \notin b \Gamma \vdash \langle c2, Normal Z \rangle \Rightarrow Abrupt t$   
**thus**  $\Gamma \vdash \langle Cond b c1 c2, Normal Z \rangle \Rightarrow Abrupt t$   
**by** (*blast intro: exec.CondFalse*)  
**qed**  
**ultimately**  
**show**  
 $\Gamma, \Theta \vdash_{t/F} \{s. s=Z \wedge \Gamma \vdash \langle Cond b c1 c2, Normal s \rangle \Rightarrow \notin (\{Stuck\} \cup Fault \text{ ' } (-F)) \wedge$   
 $\Gamma \vdash Call p \downarrow Normal \sigma \wedge$   
 $(\exists c'. \Gamma \vdash \langle Call p, Normal \sigma \rangle \rightarrow^+ (c', Normal s) \wedge$   
 $Cond b c1 c2 \in redexes c')\}$   
 $(Cond b c1 c2)$   
 $\{t. \Gamma \vdash \langle Cond b c1 c2, Normal Z \rangle \Rightarrow Normal t\},$   
 $\{t. \Gamma \vdash \langle Cond b c1 c2, Normal Z \rangle \Rightarrow Abrupt t\}$   
**by** (*rule hoaret.Cond*)  
**next**  
**case** (*While b c*)  
**let**  $?unroll = (\{(s,t). s \in b \wedge \Gamma \vdash \langle c, Normal s \rangle \Rightarrow Normal t\})^*$   
**let**  $?P' = \lambda Z. \{t. (Z,t) \in ?unroll \wedge$   
 $(\forall e. (Z,e) \in ?unroll \rightarrow e \in b$   
 $\rightarrow \Gamma \vdash \langle c, Normal e \rangle \Rightarrow \notin (\{Stuck\} \cup Fault \text{ ' } (-F)) \wedge$   
 $(\forall u. \Gamma \vdash \langle c, Normal e \rangle \Rightarrow Abrupt u \rightarrow$   
 $\Gamma \vdash \langle While b c, Normal Z \rangle \Rightarrow Abrupt u)\} \wedge$   
 $\Gamma \vdash Call p \downarrow Normal \sigma \wedge$   
 $(\exists c'. \Gamma \vdash \langle Call p, Normal \sigma \rangle \rightarrow^+$   
 $(c', Normal t) \wedge While b c \in redexes c')\}$   
**let**  $?A = \lambda Z. \{t. \Gamma \vdash \langle While b c, Normal Z \rangle \Rightarrow Abrupt t\}$   
**let**  $?r = \{(t,s). \Gamma \vdash \langle While b c \rangle \downarrow Normal s \wedge s \in b \wedge$   
 $\Gamma \vdash \langle c, Normal s \rangle \Rightarrow Normal t\}$   
**show**  $\Gamma, \Theta \vdash_{t/F}$   
 $\{s. s=Z \wedge \Gamma \vdash \langle While b c, Normal s \rangle \Rightarrow \notin (\{Stuck\} \cup Fault \text{ ' } (-F)) \wedge$   
 $\Gamma \vdash Call p \downarrow Normal \sigma \wedge$   
 $(\exists c'. \Gamma \vdash \langle Call p, Normal \sigma \rangle \rightarrow^+ (c', Normal s) \wedge While b c \in redexes c')\}$   
 $(While b c)$   
 $\{t. \Gamma \vdash \langle While b c, Normal Z \rangle \Rightarrow Normal t\},$   
 $\{t. \Gamma \vdash \langle While b c, Normal Z \rangle \Rightarrow Abrupt t\}$   
**proof** (*rule ConseqMGT [where ?P'= $\lambda Z. ?P' Z$*   
**and**  $?Q' = \lambda Z. ?P' Z \cap - b]$ )  
**have** *wf-r*: *wf ?r by (rule wf-terminates-while)*  
**show**  $\forall Z. \Gamma, \Theta \vdash_{t/F} (?P' Z) (While b c) (?P' Z \cap - b), (?A Z)$   
**proof** (*rule allI, rule hoaret.While [OF wf-r]*)  
**fix**  $Z$   
**from** *While*  
**have** *hyp-c*:  $\forall Z. \Gamma, \Theta \vdash_{t/F}$

$\{s. s=Z \wedge \Gamma \vdash \langle c, \text{Normal } s \rangle \Rightarrow \notin (\{Stuck\} \cup \text{Fault } '(-F)) \wedge$   
 $\Gamma \vdash \text{Call } p \downarrow \text{Normal } \sigma \wedge$   
 $(\exists c'. \Gamma \vdash (\text{Call } p, \text{Normal } \sigma) \rightarrow^+ (c', \text{Normal } s) \wedge c \in \text{redexes } c')\}$   
 $c$   
 $\{t. \Gamma \vdash \langle c, \text{Normal } Z \rangle \Rightarrow \text{Normal } t\},$   
 $\{t. \Gamma \vdash \langle c, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\}$  **by iprover**  
**show**  $\forall \sigma. \Gamma, \Theta \vdash_{t/F} (\{\sigma\} \cap ?P' Z \cap b) c$   
 $(\{t. (t, \sigma) \in ?r\} \cap ?P' Z), (?A Z)$   
**proof** (rule *allI*, rule *ConseqMGT* [*OF hyp-c*])  
**fix**  $\tau s$   
**assume**  $asm: s \in \{\tau\} \cap$   
 $\{t. (Z, t) \in ?unroll \wedge$   
 $(\forall e. (Z, e) \in ?unroll \rightarrow e \in b$   
 $\rightarrow \Gamma \vdash \langle c, \text{Normal } e \rangle \Rightarrow \notin (\{Stuck\} \cup \text{Fault } '(-F)) \wedge$   
 $(\forall u. \Gamma \vdash \langle c, \text{Normal } e \rangle \Rightarrow \text{Abrupt } u \rightarrow$   
 $\Gamma \vdash \langle \text{While } b \ c, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } u)\} \wedge$   
 $\Gamma \vdash \text{Call } p \downarrow \text{Normal } \sigma \wedge$   
 $(\exists c'. \Gamma \vdash (\text{Call } p, \text{Normal } \sigma) \rightarrow^+$   
 $(c', \text{Normal } t) \wedge \text{While } b \ c \in \text{redexes } c')\}$   
 $\cap b$   
**then obtain**  $c'$  **where**  
 $s\text{-eq-}\tau: s = \tau$  **and**  
 $Z\text{-s-unroll}: (Z, s) \in ?unroll$  **and**  
 $noabort: \forall e. (Z, e) \in ?unroll \rightarrow e \in b$   
 $\rightarrow \Gamma \vdash \langle c, \text{Normal } e \rangle \Rightarrow \notin (\{Stuck\} \cup \text{Fault } '(-F)) \wedge$   
 $(\forall u. \Gamma \vdash \langle c, \text{Normal } e \rangle \Rightarrow \text{Abrupt } u \rightarrow$   
 $\Gamma \vdash \langle \text{While } b \ c, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } u)$  **and**  
 $termi: \Gamma \vdash \text{Call } p \downarrow \text{Normal } \sigma$  **and**  
 $reach: \Gamma \vdash (\text{Call } p, \text{Normal } \sigma) \rightarrow^+ (c', \text{Normal } s)$  **and**  
 $red\text{-}c': \text{While } b \ c \in \text{redexes } c'$  **and**  
 $s\text{-in-}b: s \in b$   
**by blast**  
**obtain**  $c''$  **where**  
 $reach\text{-}c: \Gamma \vdash (\text{Call } p, \text{Normal } \sigma) \rightarrow^+ (c'', \text{Normal } s)$   
 $Seq \ c \ (\text{While } b \ c) \in \text{redexes } c''$   
**proof** –  
**note**  $reach$   
**also from**  $s\text{-in-}b$   
**have**  $\Gamma \vdash (\text{While } b \ c, \text{Normal } s) \rightarrow (Seq \ c \ (\text{While } b \ c), \text{Normal } s)$   
**by** (rule *step.WhileTrue*)  
**from** *step-redexes* [*OF this red-c'*] **obtain**  $c''$  **where**  
 $step: \Gamma \vdash (c', \text{Normal } s) \rightarrow (c'', \text{Normal } s)$  **and**  
 $red\text{-}c'': Seq \ c \ (\text{While } b \ c) \in \text{redexes } c''$   
**by blast**  
**note**  $step$   
**finally**  
**show**  $?thesis$   
**using**  $red\text{-}c''$   
**by** (*blast intro: that*)

**qed**  
**from** *reach termi*  
**have**  $\Gamma \vdash c' \downarrow \text{Normal } s$   
**by** (*rule steps-preserves-termination'*)  
**from** *redexes-preserves-termination* [*OF this red-c'*]  
**have** *termi-while*:  $\Gamma \vdash \text{While } b \ c \downarrow \text{Normal } s$  .  
**show**  $s \in \{t. t = s \wedge \Gamma \vdash \langle c, \text{Normal } t \rangle \Rightarrow \notin (\{\text{Stuck}\} \cup \text{Fault } '(-F)) \wedge$   
 $\Gamma \vdash \text{Call } p \downarrow \text{Normal } \sigma \wedge$   
 $(\exists c'. \Gamma \vdash (\text{Call } p, \text{Normal } \sigma) \rightarrow^+ (c', \text{Normal } t) \wedge c \in \text{redexes } c') \wedge$   
 $(\forall t. t \in \{t. \Gamma \vdash \langle c, \text{Normal } s \rangle \Rightarrow \text{Normal } t\} \rightarrow$   
 $t \in \{t. (t, \tau) \in ?r\} \cap$   
 $\{t. (Z, t) \in ?\text{unroll} \wedge$   
 $(\forall e. (Z, e) \in ?\text{unroll} \rightarrow e \in b$   
 $\rightarrow \Gamma \vdash \langle c, \text{Normal } e \rangle \Rightarrow \notin (\{\text{Stuck}\} \cup \text{Fault } '(-F)) \wedge$   
 $(\forall u. \Gamma \vdash \langle c, \text{Normal } e \rangle \Rightarrow \text{Abrupt } u \rightarrow$   
 $\Gamma \vdash \langle \text{While } b \ c, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } u)) \wedge$   
 $\Gamma \vdash \text{Call } p \downarrow \text{Normal } \sigma \wedge$   
 $(\exists c'. \Gamma \vdash (\text{Call } p, \text{Normal } \sigma) \rightarrow^+ (c', \text{Normal } t) \wedge$   
 $\text{While } b \ c \in \text{redexes } c') \wedge$   
 $(\forall t. t \in \{t. \Gamma \vdash \langle c, \text{Normal } s \rangle \Rightarrow \text{Abrupt } t\} \rightarrow$   
 $t \in \{t. \Gamma \vdash \langle \text{While } b \ c, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\})$   
**(is** *?C1*  $\wedge$  *?C2*  $\wedge$  *?C3*)  
**proof** (*intro conjI*)  
**from** *Z-s-unroll noabort s-in-b termi reach-c* **show** *?C1*  
**apply** *clarsimp*  
**apply** (*drule redexes-subset*)  
**apply** *simp*  
**apply** (*blast intro: root-in-redexes*)  
**done**  
**next**  
**{**  
**fix** *t*  
**assume** *s-t*:  $\Gamma \vdash \langle c, \text{Normal } s \rangle \Rightarrow \text{Normal } t$   
**with** *s-eq- $\tau$  termi-while s-in-b* **have**  $(t, \tau) \in ?r$   
**by** *blast*  
**moreover**  
**from** *Z-s-unroll s-t s-in-b*  
**have**  $(Z, t) \in ?\text{unroll}$   
**by** (*blast intro: rtrancl-into-rtrancl*)  
**moreover**  
**obtain** *c''* **where**  
 $\text{reach-}c'': \Gamma \vdash (\text{Call } p, \text{Normal } \sigma) \rightarrow^+ (c'', \text{Normal } t)$   
 $(\text{While } b \ c) \in \text{redexes } c''$   
**proof** –  
**note** *reach-c* (1)  
**also from** *s-in-b*  
**have**  $\Gamma \vdash (\text{While } b \ c, \text{Normal } s) \rightarrow (\text{Seq } c \ (\text{While } b \ c), \text{Normal } s)$   
**by** (*rule step.WhileTrue*)  
**have**  $\Gamma \vdash (\text{Seq } c \ (\text{While } b \ c), \text{Normal } s) \rightarrow^+$

(While b c, Normal t)

**proof** –

**from** *exec-impl-steps-Normal* [OF s-t]

**have**  $\Gamma \vdash (c, \text{Normal } s) \rightarrow^* (\text{Skip}, \text{Normal } t)$ .

**hence**  $\Gamma \vdash (\text{Seq } c (\text{While } b c), \text{Normal } s) \rightarrow^*$   
(Seq Skip (While b c), Normal t)

**by** (rule SeqSteps) auto

**moreover**

**have**  $\Gamma \vdash (\text{Seq Skip } (\text{While } b c), \text{Normal } t) \rightarrow (\text{While } b c, \text{Normal } t)$

**by** (rule step.SeqSkip)

**ultimately show** ?thesis **by** (rule rtranclp-into-tranclp1)

**qed**

**from** *steps-redexes'* [OF this reach-c (2)]

**obtain**  $c'''$  **where**

*step*:  $\Gamma \vdash (c'', \text{Normal } s) \rightarrow^+ (c''', \text{Normal } t)$  **and**

*red-c''*:  $\text{While } b c \in \text{redexes } c'''$

**by** blast

**note** *step*

**finally**

**show** ?thesis

**using** *red-c''*

**by** (blast intro: that)

**qed**

**moreover note** *noabort termi*

**ultimately**

**have**  $(t, \tau) \in ?r \wedge (Z, t) \in ?\text{unroll} \wedge$   
(∀ e. (Z, e) ∈ ?unroll → e ∈ b  
→  $\Gamma \vdash \langle c, \text{Normal } e \rangle \Rightarrow \notin (\{\text{Stuck}\} \cup \text{Fault } (-F)) \wedge$   
(∀ u.  $\Gamma \vdash \langle c, \text{Normal } e \rangle \Rightarrow \text{Abrupt } u \rightarrow$   
 $\Gamma \vdash \langle \text{While } b c, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } u)$ )  $\wedge$   
 $\Gamma \vdash \text{Call } p \downarrow \text{Normal } \sigma \wedge$   
(∃ c'.  $\Gamma \vdash (\text{Call } p, \text{Normal } \sigma) \rightarrow^+ (c', \text{Normal } t) \wedge$   
 $\text{While } b c \in \text{redexes } c')$

**by** *iprover*

}

**then show** ?C2 **by** blast

**next**

{

**fix** t

**assume** *s-t*:  $\Gamma \vdash \langle c, \text{Normal } s \rangle \Rightarrow \text{Abrupt } t$

**from** *Z-s-unroll noabort s-t s-in-b*

**have**  $\Gamma \vdash \langle \text{While } b c, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t$

**by** blast

} **thus** ?C3 **by** *simp*

**qed**

**qed**

**qed**

**next**

**fix** s

**assume**  $P: s \in \{s. s=Z \wedge \Gamma \vdash \langle \text{While } b \ c, \text{Normal } s \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F))\}$   
 $\wedge$ 

$$\Gamma \vdash \text{Call } p \downarrow \text{Normal } \sigma \wedge$$

$$(\exists c'. \Gamma \vdash (\text{Call } p, \text{Normal } \sigma) \rightarrow^+ (c', \text{Normal } s) \wedge$$

$$\text{While } b \ c \in \text{redexes } c')$$
**hence**  $\text{WhileNoFault}: \Gamma \vdash \langle \text{While } b \ c, \text{Normal } Z \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F))$   
**by auto**  
**show**  $s \in ?P' \ s \wedge$   
 $(\forall t. t \in (?P' \ s \cap - \ b) \rightarrow$   
 $t \in \{t. \Gamma \vdash \langle \text{While } b \ c, \text{Normal } Z \rangle \Rightarrow \text{Normal } t\}) \wedge$   
 $(\forall t. t \in ?A \ s \rightarrow t \in ?A \ Z)$   
**proof** (*intro conjI*)  
 $\{$   
**fix**  $e$   
**assume**  $(Z, e) \in ?\text{unroll } e \in b$   
**from** *this WhileNoFault*  
**have**  $\Gamma \vdash \langle c, \text{Normal } e \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F)) \wedge$   
 $(\forall u. \Gamma \vdash \langle c, \text{Normal } e \rangle \Rightarrow \text{Abrupt } u \rightarrow$   
 $\Gamma \vdash \langle \text{While } b \ c, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } u)$  (**is**  $?Prop \ Z \ e$ )  
**proof** (*induct rule: converse-rtrancl-induct [consumes 1]*)  
**assume**  $e\text{-in-}b: e \in b$   
**assume**  $\text{WhileNoFault}: \Gamma \vdash \langle \text{While } b \ c, \text{Normal } e \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F))$   
 $(-F)$   
**with**  $e\text{-in-}b \ \text{WhileNoFault}$   
**have**  $c\text{NoFault}: \Gamma \vdash \langle c, \text{Normal } e \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F))$   
**by** (*auto simp add: final-notin-def intro: exec.intros*)  
**moreover**  
 $\{$   
**fix**  $u$  **assume**  $\Gamma \vdash \langle c, \text{Normal } e \rangle \Rightarrow \text{Abrupt } u$   
**with**  $e\text{-in-}b$  **have**  $\Gamma \vdash \langle \text{While } b \ c, \text{Normal } e \rangle \Rightarrow \text{Abrupt } u$   
**by** (*blast intro: exec.intros*)  
 $\}$   
**ultimately**  
**show**  $?Prop \ e \ e$   
**by** *iprover*  
**next**  
**fix**  $Z \ r$   
**assume**  $e\text{-in-}b: e \in b$   
**assume**  $\text{WhileNoFault}: \Gamma \vdash \langle \text{While } b \ c, \text{Normal } Z \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F))$   
 $(-F)$   
**assume**  $\text{hyp}: \llbracket e \in b; \Gamma \vdash \langle \text{While } b \ c, \text{Normal } r \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F)) \rrbracket$   
 $\implies ?Prop \ r \ e$   
**assume**  $Z\text{-}r:$   
 $(Z, r) \in \{(Z, r). Z \in b \wedge \Gamma \vdash \langle c, \text{Normal } Z \rangle \Rightarrow \text{Normal } r\}$   
**with** *WhileNoFault*  
**have**  $\Gamma \vdash \langle \text{While } b \ c, \text{Normal } r \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F))$   
**by** (*auto simp add: final-notin-def intro: exec.intros*)  
**from** *hyp [OF e-in-b this]* **obtain**  
 $c\text{NoFault}: \Gamma \vdash \langle c, \text{Normal } e \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F))$  **and**

*Abrupt-r*:  $\forall u. \Gamma \vdash \langle c, \text{Normal } e \rangle \Rightarrow \text{Abrupt } u \longrightarrow$   
 $\Gamma \vdash \langle \text{While } b \ c, \text{Normal } r \rangle \Rightarrow \text{Abrupt } u$   
**by** *simp*

{  
**fix** *u* **assume**  $\Gamma \vdash \langle c, \text{Normal } e \rangle \Rightarrow \text{Abrupt } u$   
**with** *Abrupt-r* **have**  $\Gamma \vdash \langle \text{While } b \ c, \text{Normal } r \rangle \Rightarrow \text{Abrupt } u$  **by** *simp*  
**moreover from** *Z-r* **obtain**  
 $Z \in b \ \Gamma \vdash \langle c, \text{Normal } Z \rangle \Rightarrow \text{Normal } r$   
**by** *simp*  
**ultimately have**  $\Gamma \vdash \langle \text{While } b \ c, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } u$   
**by** (*blast intro: exec.intros*)  
}  
**with** *cNoFault* **show**  $?Prop \ Z \ e$   
**by** *iprover*  
**qed**  
}  
**with** *P* **show**  $s \in ?P' \ s$   
**by** *blast*  
**next**  
{  
**fix** *t*  
**assume** *termination*:  $t \notin b$   
**assume**  $(Z, t) \in ?unroll$   
**hence**  $\Gamma \vdash \langle \text{While } b \ c, \text{Normal } Z \rangle \Rightarrow \text{Normal } t$   
**proof** (*induct rule: converse-rtrancl-induct [consumes 1]*)  
**from** *termination*  
**show**  $\Gamma \vdash \langle \text{While } b \ c, \text{Normal } t \rangle \Rightarrow \text{Normal } t$   
**by** (*blast intro: exec.WhileFalse*)  
**next**  
**fix** *Z r*  
**assume** *first-body*:  
 $(Z, r) \in \{(s, t). s \in b \wedge \Gamma \vdash \langle c, \text{Normal } s \rangle \Rightarrow \text{Normal } t\}$   
**assume**  $(r, t) \in ?unroll$   
**assume** *rest-loop*:  $\Gamma \vdash \langle \text{While } b \ c, \text{Normal } r \rangle \Rightarrow \text{Normal } t$   
**show**  $\Gamma \vdash \langle \text{While } b \ c, \text{Normal } Z \rangle \Rightarrow \text{Normal } t$   
**proof** –  
**from** *first-body* **obtain**  
 $Z \in b \ \Gamma \vdash \langle c, \text{Normal } Z \rangle \Rightarrow \text{Normal } r$   
**by** *fast*  
**moreover**  
**from** *rest-loop* **have**  
 $\Gamma \vdash \langle \text{While } b \ c, \text{Normal } r \rangle \Rightarrow \text{Normal } t$   
**by** *fast*  
**ultimately show**  $\Gamma \vdash \langle \text{While } b \ c, \text{Normal } Z \rangle \Rightarrow \text{Normal } t$   
**by** (*rule exec.WhileTrue*)  
**qed**  
**qed**  
}  
}

**with**  $P$   
**show**  $\forall t. t \in (?P' s \cap - b)$   
 $\longrightarrow t \in \{t. \Gamma \vdash \langle \text{While } b \ c, \text{Normal } Z \rangle \Rightarrow \text{Normal } t\}$   
**by** *blast*  
**next**  
**from**  $P$  **show**  $\forall t. t \in ?A \ s \longrightarrow t \in ?A \ Z$   
**by** *simp*  
**qed**  
**qed**  
**next**  
**case**  $(\text{Call } q)$   
**let**  $?P = \{s. s = Z \wedge \Gamma \vdash \langle \text{Call } q, \text{Normal } s \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F)) \wedge$   
 $\Gamma \vdash \text{Call } p \downarrow \text{Normal } \sigma \wedge$   
 $(\exists c'. \Gamma \vdash \langle \text{Call } p, \text{Normal } \sigma \rangle \rightarrow^+ (c', \text{Normal } s) \wedge \text{Call } q \in \text{redexes } c')\}$   
**from** *noStuck-Call*  
**have**  $\forall s \in ?P. q \in \text{dom } \Gamma$   
**by** *(fastforce simp add: final-notin-def)*  
**then show**  $?case$   
**proof** *(rule conseq-extract-state-indep-prop)*  
**assume**  $q\text{-defined}: q \in \text{dom } \Gamma$   
**from** *Call-hyp* **have**  
 $\forall q \in \text{dom } \Gamma. \forall Z.$   
 $\Gamma, \Theta \vdash_{t/F} \{s. s = Z \wedge \Gamma \vdash \langle \text{Call } q, \text{Normal } s \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F)) \wedge$   
 $\Gamma \vdash \text{Call } q \downarrow \text{Normal } s \wedge ((s, q), (\sigma, p)) \in \text{termi-call-steps } \Gamma\}$   
 $(\text{Call } q)$   
 $\{t. \Gamma \vdash \langle \text{Call } q, \text{Normal } Z \rangle \Rightarrow \text{Normal } t\},$   
 $\{t. \Gamma \vdash \langle \text{Call } q, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\}$   
**by** *(simp add: exec-Call-body' noFaultStuck-Call-body' [simplified]*  
*terminates-Normal-Call-body)*  
**from** *Call-hyp*  $q\text{-defined}$  **have** *Call-hyp'*:  
 $\forall Z. \Gamma, \Theta \vdash_{t/F} \{s. s = Z \wedge \Gamma \vdash \langle \text{Call } q, \text{Normal } s \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F)) \wedge$   
 $\Gamma \vdash \text{Call } q \downarrow \text{Normal } s \wedge ((s, q), (\sigma, p)) \in \text{termi-call-steps } \Gamma\}$   
 $(\text{Call } q)$   
 $\{t. \Gamma \vdash \langle \text{Call } q, \text{Normal } Z \rangle \Rightarrow \text{Normal } t\},$   
 $\{t. \Gamma \vdash \langle \text{Call } q, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\}$   
**by** *auto*  
**show**  
 $\Gamma, \Theta \vdash_{t/F} ?P$   
 $(\text{Call } q)$   
 $\{t. \Gamma \vdash \langle \text{Call } q, \text{Normal } Z \rangle \Rightarrow \text{Normal } t\},$   
 $\{t. \Gamma \vdash \langle \text{Call } q, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\}$   
**proof** *(rule ConseqMGT [OF Call-hyp'], safe)*  
**fix**  $c'$   
**assume**  $\text{termi}: \Gamma \vdash \text{Call } p \downarrow \text{Normal } \sigma$   
**assume**  $\text{steps-c}': \Gamma \vdash (\text{Call } p, \text{Normal } \sigma) \rightarrow^+ (c', \text{Normal } Z)$   
**assume**  $\text{red-c}': \text{Call } q \in \text{redexes } c'$   
**show**  $\Gamma \vdash \text{Call } q \downarrow \text{Normal } Z$   
**proof** –  
**from** *steps-preserves-termination' [OF steps-c' termi]*

```

    have  $\Gamma \vdash c' \downarrow \text{Normal } Z$  .
    from redexes-preserves-termination [OF this red-c']
    show ?thesis .
qed
next
fix  $c'$ 
assume termi:  $\Gamma \vdash \text{Call } p \downarrow \text{Normal } \sigma$ 
assume steps-c':  $\Gamma \vdash (\text{Call } p, \text{Normal } \sigma) \rightarrow^+ (c', \text{Normal } Z)$ 
assume red-c':  $\text{Call } q \in \text{redexes } c'$ 
from redex-redexes [OF this]
have redex  $c' = \text{Call } q$ 
  by auto
with termi steps-c'
show  $((Z, q), \sigma, p) \in \text{termi-call-steps } \Gamma$ 
  by (auto simp add: termi-call-steps-def)
qed
qed
next
case (DynCom  $c$ )
have hyp:
 $\bigwedge s'. \forall Z. \Gamma, \Theta \vdash_t /_F$ 
 $\{s. s = Z \wedge \Gamma \vdash \langle c \ s', \text{Normal } s \rangle \Rightarrow \notin (\{\text{Stuck}\} \cup \text{Fault } '(-F)) \wedge$ 
 $\Gamma \vdash \text{Call } p \downarrow \text{Normal } \sigma \wedge$ 
 $(\exists c'. \Gamma \vdash (\text{Call } p, \text{Normal } \sigma) \rightarrow^+ (c', \text{Normal } s) \wedge c \ s' \in \text{redexes } c')\}$ 
 $(c \ s')$ 
 $\{t. \Gamma \vdash \langle c \ s', \text{Normal } Z \rangle \Rightarrow \text{Normal } t\}, \{t. \Gamma \vdash \langle c \ s', \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\}$ 
using DynCom by simp
have hyp':
 $\Gamma, \Theta \vdash_t /_F \{s. s = Z \wedge \Gamma \vdash \langle \text{DynCom } c, \text{Normal } s \rangle \Rightarrow \notin (\{\text{Stuck}\} \cup \text{Fault } '(-F)) \wedge$ 
 $\Gamma \vdash \text{Call } p \downarrow \text{Normal } \sigma \wedge$ 
 $(\exists c'. \Gamma \vdash (\text{Call } p, \text{Normal } \sigma) \rightarrow^+ (c', \text{Normal } s) \wedge \text{DynCom } c \in \text{redexes}$ 
 $c')\}$ 
 $(c \ Z)$ 
 $\{t. \Gamma \vdash \langle \text{DynCom } c, \text{Normal } Z \rangle \Rightarrow \text{Normal } t\}, \{t. \Gamma \vdash \langle \text{DynCom } c, \text{Normal } Z \rangle \Rightarrow$ 
Abrupt  $t\}$ 
proof (rule ConseqMGT [OF hyp], safe)
  assume  $\Gamma \vdash \langle \text{DynCom } c, \text{Normal } Z \rangle \Rightarrow \notin (\{\text{Stuck}\} \cup \text{Fault } '(-F))$ 
  then show  $\Gamma \vdash \langle c \ Z, \text{Normal } Z \rangle \Rightarrow \notin (\{\text{Stuck}\} \cup \text{Fault } '(-F))$ 
    by (fastforce simp add: final-notin-def intro: exec.intros)
next
fix  $c'$ 
assume steps:  $\Gamma \vdash (\text{Call } p, \text{Normal } \sigma) \rightarrow^+ (c', \text{Normal } Z)$ 
assume c':  $\text{DynCom } c \in \text{redexes } c'$ 
have  $\Gamma \vdash (\text{DynCom } c, \text{Normal } Z) \rightarrow (c \ Z, \text{Normal } Z)$ 
  by (rule step.DynCom)
from step-redexes [OF this c'] obtain  $c''$  where
  step:  $\Gamma \vdash (c', \text{Normal } Z) \rightarrow (c'', \text{Normal } Z)$  and c'':  $c \ Z \in \text{redexes } c''$ 
  by blast
note steps also note step

```



**finally show**  $\exists c'. \Gamma \vdash (\text{Call } p, \text{Normal } \sigma) \rightarrow^+ (c', \text{Normal } Z) \wedge c Z \in \text{redexes}$   
 $c'$   
**using**  $c''$  **by** *blast*  
**next**  
**fix**  $t$   
**assume**  $\Gamma \vdash \langle c Z, \text{Normal } Z \rangle \Rightarrow \text{Normal } t$   
**thus**  $\Gamma \vdash \langle \text{DynCom } c, \text{Normal } Z \rangle \Rightarrow \text{Normal } t$   
**by** (*auto intro: exec.intros*)  
**next**  
**fix**  $t$   
**assume**  $\Gamma \vdash \langle c Z, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t$   
**thus**  $\Gamma \vdash \langle \text{DynCom } c, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t$   
**by** (*auto intro: exec.intros*)  
**qed**  
**show** *?case*  
**apply** (*rule hoaret.DynCom*)  
**apply** *safe*  
**apply** (*rule hyp'*)  
**done**  
**next**  
**case** (*Guard f g c*)  
**have** *hyp-c*:  $\forall Z. \Gamma, \Theta \vdash_{t/F}$   
 $\{s. s=Z \wedge \Gamma \vdash \langle c, \text{Normal } s \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F)) \wedge$   
 $\Gamma \vdash \text{Call } p \downarrow \text{Normal } \sigma \wedge$   
 $(\exists c'. \Gamma \vdash (\text{Call } p, \text{Normal } \sigma) \rightarrow^+ (c', \text{Normal } s) \wedge c \in \text{redexes } c')\}$   
 $c$   
 $\{t. \Gamma \vdash \langle c, \text{Normal } Z \rangle \Rightarrow \text{Normal } t\},$   
 $\{t. \Gamma \vdash \langle c, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\}$   
**using** *Guard.hyps by iprover*  
**show**  $\Gamma, \Theta \vdash_{t/F} \{s. s=Z \wedge \Gamma \vdash \langle \text{Guard } f g c, \text{Normal } s \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F)) \wedge$   
 $(-F) \wedge$   
 $\Gamma \vdash \text{Call } p \downarrow \text{Normal } \sigma \wedge$   
 $(\exists c'. \Gamma \vdash (\text{Call } p, \text{Normal } \sigma) \rightarrow^+ (c', \text{Normal } s) \wedge \text{Guard } f g c \in \text{redexes}$   
 $c')\}$   
 $\text{Guard } f g c$   
 $\{t. \Gamma \vdash \langle \text{Guard } f g c, \text{Normal } Z \rangle \Rightarrow \text{Normal } t\},$   
 $\{t. \Gamma \vdash \langle \text{Guard } f g c, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\}$   
**proof** (*cases f ∈ F*)  
**case** *True*  
**have**  $\Gamma, \Theta \vdash_{t/F} (g \cap \{s. s=Z \wedge$   
 $\Gamma \vdash \langle \text{Guard } f g c, \text{Normal } s \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F)) \wedge$   
 $\Gamma \vdash \text{Call } p \downarrow \text{Normal } \sigma \wedge$   
 $(\exists c'. \Gamma \vdash (\text{Call } p, \text{Normal } \sigma) \rightarrow^+ (c', \text{Normal } s) \wedge$   
 $\text{Guard } f g c \in \text{redexes } c')\}$   
 $c$   
 $\{t. \Gamma \vdash \langle \text{Guard } f g c, \text{Normal } Z \rangle \Rightarrow \text{Normal } t\},$   
 $\{t. \Gamma \vdash \langle \text{Guard } f g c, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\}$   
**proof** (*rule ConseqMGT [OF hyp-c], safe*)  
**assume**  $\Gamma \vdash \langle \text{Guard } f g c, \text{Normal } Z \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F)) Z \in g$

**thus**  $\Gamma \langle c, Normal Z \rangle \Rightarrow \notin(\{Stuck\} \cup Fault \text{ ' } (-F))$   
**by** (*auto simp add: final-notin-def intro: exec.intros*)  
**next**  
**fix**  $c'$   
**assume** *steps*:  $\Gamma \langle Call p, Normal \sigma \rangle \rightarrow^+ (c', Normal Z)$   
**assume**  $c'$ :  $Guard f g c \in redexes c'$   
**assume**  $Z \in g$   
**from** *this* **have**  $\Gamma \langle Guard f g c, Normal Z \rangle \rightarrow (c, Normal Z)$   
**by** (*rule step.Guard*)  
**from** *step-redexes [OF this c']* **obtain**  $c''$  **where**  
*step*:  $\Gamma \langle c', Normal Z \rangle \rightarrow (c'', Normal Z)$  **and**  $c''$ :  $c \in redexes c''$   
**by** *blast*  
**note** *steps* **also note** *step*  
**finally show**  $\exists c'. \Gamma \langle Call p, Normal \sigma \rangle \rightarrow^+ (c', Normal Z) \wedge c \in redexes$   
 $c'$   
**using**  $c''$  **by** *blast*  
**next**  
**fix**  $t$   
**assume**  $\Gamma \langle Guard f g c, Normal Z \rangle \Rightarrow \notin(\{Stuck\} \cup Fault \text{ ' } (-F))$   
 $\Gamma \langle c, Normal Z \rangle \Rightarrow Normal t Z \in g$   
**thus**  $\Gamma \langle Guard f g c, Normal Z \rangle \Rightarrow Normal t$   
**by** (*auto simp add: final-notin-def intro: exec.intros*)  
**next**  
**fix**  $t$   
**assume**  $\Gamma \langle Guard f g c, Normal Z \rangle \Rightarrow \notin(\{Stuck\} \cup Fault \text{ ' } (-F))$   
 $\Gamma \langle c, Normal Z \rangle \Rightarrow Abrupt t Z \in g$   
**thus**  $\Gamma \langle Guard f g c, Normal Z \rangle \Rightarrow Abrupt t$   
**by** (*auto simp add: final-notin-def intro: exec.intros*)  
**qed**  
**from** *True this* **show** *?thesis*  
**by** (*rule conseqPre [OF Guarantee] auto*)  
**next**  
**case** *False*  
**have**  $\Gamma, \Theta \vdash_{t/F} (g \cap \{s. s=Z \wedge$   
 $\Gamma \langle Guard f g c, Normal s \rangle \Rightarrow \notin(\{Stuck\} \cup Fault \text{ ' } (-F)) \wedge$   
 $\Gamma \langle Call p \downarrow Normal \sigma \rangle \wedge$   
 $(\exists c'. \Gamma \langle Call p, Normal \sigma \rangle \rightarrow^+ (c', Normal s) \wedge$   
 $Guard f g c \in redexes c'))$   
 $c$   
 $\{t. \Gamma \langle Guard f g c, Normal Z \rangle \Rightarrow Normal t\},$   
 $\{t. \Gamma \langle Guard f g c, Normal Z \rangle \Rightarrow Abrupt t\}$   
**proof** (*rule ConseqMGT [OF hyp-c], safe*)  
**assume**  $\Gamma \langle Guard f g c, Normal Z \rangle \Rightarrow \notin(\{Stuck\} \cup Fault \text{ ' } (-F))$   
**thus**  $\Gamma \langle c, Normal Z \rangle \Rightarrow \notin(\{Stuck\} \cup Fault \text{ ' } (-F))$   
**using** *False*  
**by** (*cases Z \in g*) (*auto simp add: final-notin-def intro: exec.intros*)  
**next**  
**fix**  $c'$   
**assume** *steps*:  $\Gamma \langle Call p, Normal \sigma \rangle \rightarrow^+ (c', Normal Z)$

**assume**  $c'$ :  $\text{Guard } f g c \in \text{redexes } c'$

**assume**  $Z \in g$

**from** *this* **have**  $\Gamma \vdash (\text{Guard } f g c, \text{Normal } Z) \rightarrow (c, \text{Normal } Z)$

**by** (*rule step.Guard*)

**from** *step-redexes* [*OF this c'*] **obtain**  $c''$  **where**

*step*:  $\Gamma \vdash (c', \text{Normal } Z) \rightarrow (c'', \text{Normal } Z)$  **and**  $c''$ :  $c \in \text{redexes } c''$

**by** *blast*

**note** *steps also note step*

**finally show**  $\exists c'. \Gamma \vdash (\text{Call } p, \text{Normal } \sigma) \rightarrow^+ (c', \text{Normal } Z) \wedge c \in \text{redexes } c'$

**using**  $c''$  **by** *blast*

**next**

**fix**  $t$

**assume**  $\Gamma \vdash \langle \text{Guard } f g c, \text{Normal } Z \rangle \Rightarrow \notin (\{\text{Stuck}\} \cup \text{Fault } '(-F))$

$\Gamma \vdash \langle c, \text{Normal } Z \rangle \Rightarrow \text{Normal } t$

**thus**  $\Gamma \vdash \langle \text{Guard } f g c, \text{Normal } Z \rangle \Rightarrow \text{Normal } t$

**using** *False*

**by** (*cases Z ∈ g*) (*auto simp add: final-notin-def intro: exec.intros*)

**next**

**fix**  $t$

**assume**  $\Gamma \vdash \langle \text{Guard } f g c, \text{Normal } Z \rangle \Rightarrow \notin (\{\text{Stuck}\} \cup \text{Fault } '(-F))$

$\Gamma \vdash \langle c, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t$

**thus**  $\Gamma \vdash \langle \text{Guard } f g c, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t$

**using** *False*

**by** (*cases Z ∈ g*) (*auto simp add: final-notin-def intro: exec.intros*)

**qed**

**then show** *?thesis*

**apply** (*rule conseqPre* [*OF hoaret.Guard*])

**apply** *clarify*

**apply** (*frule Guard-noFaultStuckD* [*OF - False*])

**apply** *auto*

**done**

**qed**

**next**

**case** *Throw*

**show**  $\Gamma, \Theta \vdash_{t/F} \{s. s = Z \wedge \Gamma \vdash \langle \text{Throw}, \text{Normal } s \rangle \Rightarrow \notin (\{\text{Stuck}\} \cup \text{Fault } '(-F)) \wedge$

$\Gamma \vdash \text{Call } p \downarrow \text{Normal } \sigma \wedge$

$(\exists c'. \Gamma \vdash (\text{Call } p, \text{Normal } \sigma) \rightarrow^+ (c', \text{Normal } s) \wedge \text{Throw} \in \text{redexes } c')\}$

$\text{Throw}$

$\{t. \Gamma \vdash \langle \text{Throw}, \text{Normal } Z \rangle \Rightarrow \text{Normal } t\},$

$\{t. \Gamma \vdash \langle \text{Throw}, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\}$

**by** (*rule conseqPre* [*OF hoaret.Throw*])

(*blast intro: exec.intros terminates.intros*)

**next**

**case** (*Catch*  $c_1$   $c_2$ )

**have** *hyp-c1*:

$\forall Z. \Gamma, \Theta \vdash_{t/F} \{s. s = Z \wedge \Gamma \vdash \langle c_1, \text{Normal } s \rangle \Rightarrow \notin (\{\text{Stuck}\} \cup \text{Fault } '(-F)) \wedge$

$\Gamma \vdash \text{Call } p \downarrow \text{Normal } \sigma \wedge$

$$\begin{array}{l}
(\exists c'. \Gamma \vdash (\text{Call } p, \text{Normal } \sigma) \rightarrow^+ (c', \text{Normal } s) \wedge \\
\quad c_1 \in \text{redexes } c') \\
\quad c_1 \\
\quad \{t. \Gamma \vdash \langle c_1, \text{Normal } Z \rangle \Rightarrow \text{Normal } t\}, \{t. \Gamma \vdash \langle c_1, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\} \\
\text{using Catch.hyps by iprover} \\
\text{have hyp-c2:} \\
\forall Z. \Gamma, \Theta \vdash_{t/F} \{s. s = Z \wedge \Gamma \vdash \langle c_2, \text{Normal } s \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F)) \wedge \\
\quad \Gamma \vdash \text{Call } p \downarrow \text{Normal } \sigma \wedge \\
\quad (\exists c'. \Gamma \vdash (\text{Call } p, \text{Normal } \sigma) \rightarrow^+ (c', \text{Normal } s) \wedge c_2 \in \text{redexes } c')\} \\
\quad c_2 \\
\quad \{t. \Gamma \vdash \langle c_2, \text{Normal } Z \rangle \Rightarrow \text{Normal } t\}, \{t. \Gamma \vdash \langle c_2, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\} \\
\text{using Catch.hyps by iprover} \\
\text{have} \\
\Gamma, \Theta \vdash_{t/F} \{s. s = Z \wedge \Gamma \vdash \langle \text{Catch } c_1 \ c_2, \text{Normal } s \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F))\} \\
\wedge \\
\quad \Gamma \vdash \text{Call } p \downarrow \text{Normal } \sigma \wedge \\
\quad (\exists c'. \Gamma \vdash (\text{Call } p, \text{Normal } \sigma) \rightarrow^+ (c', \text{Normal } s) \wedge \\
\quad \quad \text{Catch } c_1 \ c_2 \in \text{redexes } c') \\
\quad c_1 \\
\quad \{t. \Gamma \vdash \langle \text{Catch } c_1 \ c_2, \text{Normal } Z \rangle \Rightarrow \text{Normal } t\}, \\
\quad \{t. \Gamma \vdash \langle c_1, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t \wedge \\
\quad \quad \Gamma \vdash \langle c_2, \text{Normal } t \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F)) \wedge \Gamma \vdash \text{Call } p \downarrow \text{Normal } \sigma \wedge \\
\quad \quad (\exists c'. \Gamma \vdash (\text{Call } p, \text{Normal } \sigma) \rightarrow^+ (c', \text{Normal } t) \wedge c_2 \in \text{redexes } c')\} \\
\text{proof (rule ConseqMGT [OF hyp-c1], clarify, safe)} \\
\text{assume } \Gamma \vdash \langle \text{Catch } c_1 \ c_2, \text{Normal } Z \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F)) \\
\text{thus } \Gamma \vdash \langle c_1, \text{Normal } Z \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F)) \\
\text{by (fastforce simp add: final-notin-def intro: exec.intros)} \\
\text{next} \\
\text{fix } c' \\
\text{assume steps: } \Gamma \vdash (\text{Call } p, \text{Normal } \sigma) \rightarrow^+ (c', \text{Normal } Z) \\
\text{assume } c': \text{Catch } c_1 \ c_2 \in \text{redexes } c' \\
\text{from steps redexes-subset [OF this]} \\
\text{show } \exists c'. \Gamma \vdash (\text{Call } p, \text{Normal } \sigma) \rightarrow^+ (c', \text{Normal } Z) \wedge c_1 \in \text{redexes } c' \\
\text{by (auto iff: root-in-redexes)} \\
\text{next} \\
\text{fix } t \\
\text{assume } \Gamma \vdash \langle c_1, \text{Normal } Z \rangle \Rightarrow \text{Normal } t \\
\text{thus } \Gamma \vdash \langle \text{Catch } c_1 \ c_2, \text{Normal } Z \rangle \Rightarrow \text{Normal } t \\
\text{by (auto intro: exec.intros)} \\
\text{next} \\
\text{fix } t \\
\text{assume } \Gamma \vdash \langle \text{Catch } c_1 \ c_2, \text{Normal } Z \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F)) \\
\quad \Gamma \vdash \langle c_1, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t \\
\text{thus } \Gamma \vdash \langle c_2, \text{Normal } t \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F)) \\
\text{by (auto simp add: final-notin-def intro: exec.intros)} \\
\text{next} \\
\text{fix } c' \ t \\
\text{assume steps-c': } \Gamma \vdash (\text{Call } p, \text{Normal } \sigma) \rightarrow^+ (c', \text{Normal } Z) \\
\text{assume red: } \text{Catch } c_1 \ c_2 \in \text{redexes } c'
\end{array}$$

**assume**  $exec-c_1: \Gamma \vdash \langle c_1, Normal Z \rangle \Rightarrow Abrupt t$   
**show**  $\exists c'. \Gamma \vdash (Call\ p, Normal\ \sigma) \rightarrow^+ (c', Normal\ t) \wedge c_2 \in redexes\ c'$   
**proof** –  
**note**  $steps-c'$   
**also**  
**from**  $exec-impl-steps-Normal-Abrupt$  [OF  $exec-c_1$ ]  
**have**  $\Gamma \vdash (c_1, Normal\ Z) \rightarrow^* (Throw, Normal\ t)$ .  
**from**  $steps-redexes-Catch$  [OF  $this\ red$ ]  
**obtain**  $c''$  **where**  
 $steps-c'': \Gamma \vdash (c', Normal\ Z) \rightarrow^* (c'', Normal\ t)$  **and**  
 $Catch: Catch\ Throw\ c_2 \in redexes\ c''$   
**by**  $blast$   
**note**  $steps-c''$   
**also**  
**have**  $step-Catch: \Gamma \vdash (Catch\ Throw\ c_2, Normal\ t) \rightarrow (c_2, Normal\ t)$   
**by**  $(rule\ step.CatchThrow)$   
**from**  $step-redexes$  [OF  $step-Catch\ Catch$ ]  
**obtain**  $c'''$  **where**  
 $step-c''': \Gamma \vdash (c'', Normal\ t) \rightarrow (c''', Normal\ t)$  **and**  
 $c_2: c_2 \in redexes\ c'''$   
**by**  $blast$   
**note**  $step-c'''$   
**finally show**  $?thesis$   
**using**  $c_2$   
**by**  $blast$   
**qed**  
**qed**  
**moreover**  
**have**  $\Gamma, \Theta \vdash_{t/F} \{t. \Gamma \vdash \langle c_1, Normal\ Z \rangle \Rightarrow Abrupt\ t \wedge$   
 $\Gamma \vdash \langle c_2, Normal\ t \rangle \Rightarrow \notin(\{Stuck\} \cup Fault\ '(-F)) \wedge$   
 $\Gamma \vdash Call\ p \downarrow Normal\ \sigma \wedge$   
 $(\exists c'. \Gamma \vdash (Call\ p, Normal\ \sigma) \rightarrow^+ (c', Normal\ t) \wedge c_2 \in redexes\ c')\}$   
 $c_2$   
 $\{t. \Gamma \vdash \langle Catch\ c_1\ c_2, Normal\ Z \rangle \Rightarrow Normal\ t\},$   
 $\{t. \Gamma \vdash \langle Catch\ c_1\ c_2, Normal\ Z \rangle \Rightarrow Abrupt\ t\}$   
**by**  $(rule\ ConseqMGT$  [OF  $hyp-c_2$ ])  $(fastforce\ intro: exec.intros)$   
**ultimately show**  $?case$   
**by**  $(rule\ hoaret.Catch)$   
**qed**

To prove a procedure implementation correct it suffices to assume only the procedure specifications of procedures that actually occur during evaluation of the body.

**lemma** *Call-lemma*:

**assumes**  $A$ :

$\forall q \in dom\ \Gamma. \forall Z. \Gamma, \Theta \vdash_{t/F}$

$\{s. s=Z \wedge \Gamma \vdash \langle Call\ q, Normal\ s \rangle \Rightarrow \notin(\{Stuck\} \cup Fault\ '(-F)) \wedge$   
 $\Gamma \vdash Call\ q \downarrow Normal\ s \wedge ((s, q), (\sigma, p)) \in termi-call-steps\ \Gamma\}$   
 $(Call\ q)$

$\{t. \Gamma \vdash \langle \text{Call } q, \text{Normal } Z \rangle \Rightarrow \text{Normal } t\},$   
 $\{t. \Gamma \vdash \langle \text{Call } q, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\}$   
**assumes**  $p\text{def}: p \in \text{dom } \Gamma$   
**shows**  $\bigwedge Z. \Gamma, \Theta \vdash_{t/F}$   
 $(\{\sigma\} \cap \{s. s=Z \wedge \Gamma \vdash \langle \text{the } (\Gamma \ p), \text{Normal } s \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F))$   
 $\wedge$   
 $\Gamma \vdash \text{the } (\Gamma \ p) \downarrow \text{Normal } s\}$   
 $\text{the } (\Gamma \ p)$   
 $\{t. \Gamma \vdash \langle \text{the } (\Gamma \ p), \text{Normal } Z \rangle \Rightarrow \text{Normal } t\},$   
 $\{t. \Gamma \vdash \langle \text{the } (\Gamma \ p), \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\}$   
**apply** (*rule conseqPre*)  
**apply** (*rule Call-lemma' [OF A]*)  
**using**  $p\text{def}$   
**apply** (*fastforce intro: terminates.intros tranclp.r-into-trancl [of (step  $\Gamma$ ), OF step.Call]*  
*root-in-redexes*)  
**done**

**lemma** *Call-lemma-switch-Call-body:*

**assumes**  
 $\text{call}: \forall q \in \text{dom } \Gamma. \forall Z. \Gamma, \Theta \vdash_{t/F}$   
 $\{s. s=Z \wedge \Gamma \vdash \langle \text{Call } q, \text{Normal } s \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F)) \wedge$   
 $\Gamma \vdash \text{Call } q \downarrow \text{Normal } s \wedge ((s, q), (\sigma, p)) \in \text{termi-call-steps } \Gamma\}$   
 $(\text{Call } q)$   
 $\{t. \Gamma \vdash \langle \text{Call } q, \text{Normal } Z \rangle \Rightarrow \text{Normal } t\},$   
 $\{t. \Gamma \vdash \langle \text{Call } q, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\}$   
**assumes**  $p\text{-defined}: p \in \text{dom } \Gamma$   
**shows**  $\bigwedge Z. \Gamma, \Theta \vdash_{t/F}$   
 $(\{\sigma\} \cap \{s. s=Z \wedge \Gamma \vdash \langle \text{Call } p, \text{Normal } s \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F)) \wedge$   
 $\Gamma \vdash \text{Call } p \downarrow \text{Normal } s\}$   
 $\text{the } (\Gamma \ p)$   
 $\{t. \Gamma \vdash \langle \text{Call } p, \text{Normal } Z \rangle \Rightarrow \text{Normal } t\},$   
 $\{t. \Gamma \vdash \langle \text{Call } p, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\}$   
**apply** (*simp only: exec-Call-body' [OF p-defined] noFaultStuck-Call-body' [OF p-defined]*  
*terminates-Normal-Call-body [OF p-defined]*)  
**apply** (*rule conseqPre*)  
**apply** (*rule Call-lemma'*)  
**apply** (*rule call*)  
**using**  $p\text{-defined}$   
**apply** (*fastforce intro: terminates.intros tranclp.r-into-trancl [of (step  $\Gamma$ ), OF step.Call]*  
*root-in-redexes*)  
**done**

**lemma** *MGT-Call:*

$\forall p \in \text{dom } \Gamma. \forall Z.$   
 $\Gamma, \Theta \vdash_{t/F} \{s. s=Z \wedge \Gamma \vdash \langle \text{Call } p, \text{Normal } s \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F)) \wedge$   
 $\Gamma \vdash (\text{Call } p) \downarrow \text{Normal } s\}$   
 $(\text{Call } p)$   
 $\{t. \Gamma \vdash \langle \text{Call } p, \text{Normal } Z \rangle \Rightarrow \text{Normal } t\},$   
 $\{t. \Gamma \vdash \langle \text{Call } p, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\}$

```

apply (intro ballI allI)
apply (rule CallRec' [where Procs=dom  $\Gamma$  and
   $P=\lambda p Z. \{s. s=Z \wedge \Gamma \vdash \langle \text{Call } p, \text{Normal } s \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F)) \wedge$ 
     $\Gamma \vdash \text{Call } p \downarrow \text{Normal } s\}$  and
   $Q=\lambda p Z. \{t. \Gamma \vdash \langle \text{Call } p, \text{Normal } Z \rangle \Rightarrow \text{Normal } t\}$  and
   $A=\lambda p Z. \{t. \Gamma \vdash \langle \text{Call } p, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\}$  and
   $r=\text{termi-call-steps } \Gamma$ 
])
apply simp
apply simp
apply (rule wf-termi-call-steps)
apply (intro ballI allI)
apply simp
apply (rule Call-lemma-switch-Call-body [rule-format, simplified])
apply (rule hoaret.Asm)
apply fastforce
apply assumption
done

```

**lemma** CollInt-iff:  $\{s. P s\} \cap \{s. Q s\} = \{s. P s \wedge Q s\}$   
 by auto

**lemma** image-Un-conv:  $f ' (\bigcup_{p \in \text{dom } \Gamma} \bigcup Z. \{x p Z\}) = (\bigcup_{p \in \text{dom } \Gamma} \bigcup Z. \{f (x p Z)\})$   
 by (auto iff: not-None-eq)

Another proof of *MGT-Call*, maybe a little more readable

**lemma**

$\forall p \in \text{dom } \Gamma. \forall Z.$

$\Gamma, \{\} \vdash_{\iota/F} \{s. s=Z \wedge \Gamma \vdash \langle \text{Call } p, \text{Normal } s \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F)) \wedge$   
 $\Gamma \vdash (\text{Call } p) \downarrow \text{Normal } s\}$   
 (*Call p*)  
 $\{t. \Gamma \vdash \langle \text{Call } p, \text{Normal } Z \rangle \Rightarrow \text{Normal } t\},$   
 $\{t. \Gamma \vdash \langle \text{Call } p, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\}$

**proof** –

```

{
  fix p Z  $\sigma$ 
  assume defined:  $p \in \text{dom } \Gamma$ 
  define Specs where Specs = ( $\bigcup_{p \in \text{dom } \Gamma} \bigcup Z.$ 
     $\{(\{s. s=Z \wedge$ 
       $\Gamma \vdash \langle \text{Call } p, \text{Normal } s \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F)) \wedge$ 
       $\Gamma \vdash \text{Call } p \downarrow \text{Normal } s\},$ 
      p,
       $\{t. \Gamma \vdash \langle \text{Call } p, \text{Normal } Z \rangle \Rightarrow \text{Normal } t\},$ 
       $\{t. \Gamma \vdash \langle \text{Call } p, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\})\}$ )
  define Specs-wf where Specs-wf p  $\sigma = (\lambda(P,q,Q,A).$ 
    ( $P \cap \{s. ((s,q),\sigma,p) \in \text{termi-call-steps } \Gamma\}, q, Q, A)) ' \text{Specs for}$ 

```

$p \sigma$

```

have  $\Gamma, \text{Specs-wf } p \sigma$ 
   $\vdash_{t/F}(\{\sigma\} \cap$ 
     $\{s. s = Z \wedge \Gamma \vdash \langle \text{the } (\Gamma p), \text{Normal } s \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F)) \wedge$ 
     $\Gamma \vdash \text{the } (\Gamma p) \downarrow \text{Normal } s\}$ 
     $(\text{the } (\Gamma p))$ 
     $\{t. \Gamma \vdash \langle \text{the } (\Gamma p), \text{Normal } Z \rangle \Rightarrow \text{Normal } t\},$ 
     $\{t. \Gamma \vdash \langle \text{the } (\Gamma p), \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\}$ 
  apply (rule Call-lemma [rule-format, OF - defined])
  apply (rule hoaret.Asm)
  apply (clarsimp simp add: Specs-wf-def Specs-def image-Un-conv)
  apply (rule-tac  $x=q$  in bexI)
  apply (rule-tac  $x=Z$  in exI)
  apply (clarsimp simp add: CollInt-iff)
  apply auto
  done
hence  $\Gamma, \text{Specs-wf } p \sigma$ 
   $\vdash_{t/F}(\{\sigma\} \cap$ 
     $\{s. s = Z \wedge \Gamma \vdash \langle \text{Call } p, \text{Normal } s \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F)) \wedge$ 
     $\Gamma \vdash \text{Call } p \downarrow \text{Normal } s\}$ 
     $(\text{the } (\Gamma p))$ 
     $\{t. \Gamma \vdash \langle \text{Call } p, \text{Normal } Z \rangle \Rightarrow \text{Normal } t\},$ 
     $\{t. \Gamma \vdash \langle \text{Call } p, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\}$ 
  by (simp only: exec-Call-body' [OF defined]
    noFaultStuck-Call-body' [OF defined]
    terminates-Normal-Call-body [OF defined])
  note bdy=this
show ?thesis
  apply (intro ballI allI)
  apply (rule hoaret.CallRec [where  $\text{Specs}=(\bigcup p \in \text{dom } \Gamma. \bigcup Z.$ 
     $\{(\{s. s=Z \wedge$ 
     $\Gamma \vdash \langle \text{Call } p, \text{Normal } s \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F)) \wedge$ 
     $\Gamma \vdash \text{Call } p \downarrow \text{Normal } s\},$ 
     $p,$ 
     $\{t. \Gamma \vdash \langle \text{Call } p, \text{Normal } Z \rangle \Rightarrow \text{Normal } t\},$ 
     $\{t. \Gamma \vdash \langle \text{Call } p, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\}\}$ ),
    OF - wf-termi-call-steps [of  $\Gamma$ ] refl])
  apply fastforce
  apply clarify
  apply (rule conjI)
  apply fastforce
  apply (rule allI)
  apply (simp (no-asm-use) only : Un-empty-left)
  apply (rule bdy)
  apply auto
  done
qed

```

**theorem** *hoaret-complete*:  $\Gamma \models_{t/F} P \text{ c } Q, A \implies \Gamma, \{\} \vdash_{t/F} P \text{ c } Q, A$



by (iprover intro: MGT-implies-complete MGT-lemma [OF MGT-Call])

**lemma** *hoaret-complete'*:

**assumes** *cvalid*:  $\Gamma, \Theta \models_{t/F} P \text{ c } Q, A$

**shows**  $\Gamma, \Theta \vdash_{t/F} P \text{ c } Q, A$

**proof** (cases  $\Gamma \models_{t/F} P \text{ c } Q, A$ )

**case** *True*

**hence**  $\Gamma, \{\} \vdash_{t/F} P \text{ c } Q, A$

**by** (rule *hoaret-complete*)

**thus**  $\Gamma, \Theta \vdash_{t/F} P \text{ c } Q, A$

**by** (rule *hoaret-augment-context*) *simp*

**next**

**case** *False*

**with** *cvalid*

**show** *?thesis*

**by** (rule *ExFalso*)

**qed**

## 10.3 And Now: Some Useful Rules

### 10.3.1 Modify Return

**lemma** *Proc-exnModifyReturn-sound*:

**assumes** *valid-call*:  $\Gamma, \Theta \models_{t/F} P \text{ call-exn } \textit{init } p \text{ return}' \textit{ result-exn } \textit{c } Q, A$

**assumes** *valid-modif*:

$\forall \sigma. \Gamma, \Theta \models_{UNIV} \{\sigma\} (\textit{Call } p) (\textit{Modif } \sigma), (\textit{ModifAbr } \sigma)$

**assumes** *res-modif*:

$\forall s \ t. t \in \textit{Modif } (\textit{init } s) \longrightarrow \textit{return}' \ s \ t = \textit{return } \ s \ t$

**assumes** *ret-modifAbr*:

$\forall s \ t. t \in \textit{ModifAbr } (\textit{init } s) \longrightarrow \textit{result-exn } (\textit{return}' \ s \ t) \ t = \textit{result-exn } (\textit{return } \ s \ t) \ t$

**shows**  $\Gamma, \Theta \models_{t/F} P (\textit{call-exn } \textit{init } p \textit{ return } \textit{result-exn } \textit{c}) \ Q, A$

**proof** (rule *cvalidtI*)

**fix** *s t*

**assume** *ctxt*:  $\forall (P, p, Q, A) \in \Theta. \Gamma \models_{t/F} P (\textit{Call } p) \ Q, A$

**hence**  $\forall (P, p, Q, A) \in \Theta. \Gamma \models_{/F} P (\textit{Call } p) \ Q, A$

**by** (auto *simp add: validt-def*)

**then have** *ctxt'*:  $\forall (P, p, Q, A) \in \Theta. \Gamma \models_{UNIV} P (\textit{Call } p) \ Q, A$

**by** (auto *intro: valid-augment-Faults*)

**assume** *exec*:  $\Gamma \vdash \langle \textit{call-exn } \textit{init } p \textit{ return } \textit{result-exn } \textit{c}, \textit{Normal } s \rangle \Rightarrow t$

**assume** *P*:  $s \in P$

**assume** *t-notin-F*:  $t \notin \textit{Fault } \textit{F}$

**from** *exec*

**show**  $t \in \textit{Normal } \textit{Q} \cup \textit{Abrupt } \textit{A}$

**proof** (cases *rule: exec-call-exn-Normal-elim*)

**fix** *bdy t'*

**assume** *bdy*:  $\Gamma \vdash p = \textit{Some } \textit{bdy}$

**assume** *exec-body*:  $\Gamma \vdash \langle \textit{bdy}, \textit{Normal } (\textit{init } s) \rangle \Rightarrow \textit{Normal } t'$

```

assume exec-c:  $\Gamma \vdash \langle c \ s \ t', Normal \ (return \ s \ t') \rangle \Rightarrow t$ 
from exec-body bdy
have  $\Gamma \vdash \langle (Call \ p), Normal \ (init \ s) \rangle \Rightarrow Normal \ t'$ 
  by (auto simp add: intro: exec.intros)
from cvalidD [OF valid-modif [rule-format, of init s] ctxt' this] P
have  $t' \in Modif \ (init \ s)$ 
  by auto
with res-modif have  $Normal \ (return' \ s \ t') = Normal \ (return \ s \ t')$ 
  by simp
with exec-body exec-c bdy
have  $\Gamma \vdash \langle call\text{-exn} \ init \ p \ return' \ result\text{-exn} \ c, Normal \ s \rangle \Rightarrow t$ 
  by (auto intro: exec-call-exn)
from cvalidt-postD [OF valid-call ctxt this] P t-notin-F
show ?thesis
  by simp
next
fix bdy t'
assume bdy:  $\Gamma \ p = Some \ bdy$ 
assume exec-body:  $\Gamma \vdash \langle bdy, Normal \ (init \ s) \rangle \Rightarrow Abrupt \ t'$ 
assume t:  $t = Abrupt \ (result\text{-exn} \ (return \ s \ t') \ t')$ 
also from exec-body bdy
have  $\Gamma \vdash \langle (Call \ p), Normal \ (init \ s) \rangle \Rightarrow Abrupt \ t'$ 
  by (auto simp add: intro: exec.intros)
from cvalidD [OF valid-modif [rule-format, of init s] ctxt' this] P
have  $t' \in ModifAbr \ (init \ s)$ 
  by auto
with ret-modifAbr have  $Abrupt \ (result\text{-exn} \ (return \ s \ t') \ t') = Abrupt \ (result\text{-exn} \ (return' \ s \ t') \ t')$ 
  by simp
finally have  $t = Abrupt \ (result\text{-exn} \ (return' \ s \ t') \ t')$  .
with exec-body bdy
have  $\Gamma \vdash \langle call\text{-exn} \ init \ p \ return' \ result\text{-exn} \ c, Normal \ s \rangle \Rightarrow t$ 
  by (auto intro: exec-call-exnAbrupt)
from cvalidt-postD [OF valid-call ctxt this] P t-notin-F
show ?thesis
  by simp
next
fix bdy f
assume bdy:  $\Gamma \ p = Some \ bdy$ 
assume  $\Gamma \vdash \langle bdy, Normal \ (init \ s) \rangle \Rightarrow Fault \ f$  and
  t:  $t = Fault \ f$ 
with bdy have  $\Gamma \vdash \langle call\text{-exn} \ init \ p \ return' \ result\text{-exn} \ c, Normal \ s \rangle \Rightarrow t$ 
  by (auto intro: exec-call-exnFault)
from cvalidt-postD [OF valid-call ctxt this] P t-notin-F
show ?thesis
  by simp
next
fix bdy
assume bdy:  $\Gamma \ p = Some \ bdy$ 

```

```

assume  $\Gamma \vdash \langle bdy, Normal (init\ s) \rangle \Rightarrow Stuck$ 
   $t = Stuck$ 
with  $bdy$  have  $\Gamma \vdash \langle call\text{-}exn\ init\ p\ return'\ result\text{-}exn\ c, Normal\ s \rangle \Rightarrow t$ 
  by (auto intro: exec-call-exnStuck)
from valid-call ctxt this P t-notin-F
show ?thesis
  by (rule cvalidt-postD)
next
assume  $\Gamma\ p = None\ t = Stuck$ 
hence  $\Gamma \vdash \langle call\text{-}exn\ init\ p\ return'\ result\text{-}exn\ c, Normal\ s \rangle \Rightarrow t$ 
  by (auto intro: exec-call-exnUndefined)
from valid-call ctxt this P t-notin-F
show ?thesis
  by (rule cvalidt-postD)
qed
next
fix  $s$ 
assume  $ctxt: \forall (P, p, Q, A) \in \Theta. \Gamma \models_{t/F} P (Call\ p)\ Q, A$ 
hence  $\forall (P, p, Q, A) \in \Theta. \Gamma \models_{/F} P (Call\ p)\ Q, A$ 
  by (auto simp add: validt-def)
then have  $ctxt': \forall (P, p, Q, A) \in \Theta. \Gamma \models_{/UNIV} P (Call\ p)\ Q, A$ 
  by (auto intro: valid-augment-Faults)
assume  $P: s \in P$ 
from valid-call ctxt P
have  $call: \Gamma \vdash call\text{-}exn\ init\ p\ return'\ result\text{-}exn\ c \downarrow Normal\ s$ 
  by (rule cvalidt-termD)
show  $\Gamma \vdash call\text{-}exn\ init\ p\ return\ result\text{-}exn\ c \downarrow Normal\ s$ 
proof (cases p \in dom \Gamma)
  case True
  with  $call$  obtain  $bdy$  where
     $bdy: \Gamma\ p = Some\ bdy$  and  $termi\text{-}bdy: \Gamma \vdash bdy \downarrow Normal (init\ s)$  and
     $termi\text{-}c: \forall t. \Gamma \vdash \langle bdy, Normal (init\ s) \rangle \Rightarrow Normal\ t \longrightarrow$ 
       $\Gamma \vdash c\ s\ t \downarrow Normal (return'\ s\ t)$ 
  by cases auto
  {
  fix  $t$ 
assume  $exec\text{-}bdy: \Gamma \vdash \langle bdy, Normal (init\ s) \rangle \Rightarrow Normal\ t$ 
hence  $\Gamma \vdash c\ s\ t \downarrow Normal (return\ s\ t)$ 
proof –
  from  $exec\text{-}bdy\ bdy$ 
have  $\Gamma \vdash \langle (Call\ p), Normal (init\ s) \rangle \Rightarrow Normal\ t$ 
  by (auto simp add: intro: exec.intros)
from  $cvalidD [OF\ valid\text{-}modif [rule\text{-}format, of\ init\ s] ctxt'\ this] P$ 
  res-modif
have  $return'\ s\ t = return\ s\ t$ 
  by auto
with  $termi\text{-}c\ exec\text{-}bdy$  show ?thesis by auto
qed
  }
}

```

**with** *bdy termi-bdy*  
**show** *?thesis*  
**by** (*iprover intro: terminates-call-exn*)  
**next**  
**case** *False*  
**thus** *?thesis*  
**by** (*auto intro: terminates-call-exnUndefined*)  
**qed**  
**qed**

**lemma** *ProcModifyReturn-sound*:  
**assumes** *valid-call*:  $\Gamma, \Theta \Vdash_{t/F} P \text{ call init } p \text{ return}' c \ Q, A$   
**assumes** *valid-modif*:  
 $\forall \sigma. \Gamma, \Theta \Vdash_{UNIV} \{\sigma\} \ (Call \ p) \ (Modif \ \sigma), (ModifAbr \ \sigma)$   
**assumes** *res-modif*:  
 $\forall s \ t. t \in Modif \ (init \ s) \longrightarrow \text{return}' \ s \ t = \text{return} \ s \ t$   
**assumes** *ret-modifAbr*:  
 $\forall s \ t. t \in ModifAbr \ (init \ s) \longrightarrow \text{return}' \ s \ t = \text{return} \ s \ t$   
**shows**  $\Gamma, \Theta \Vdash_{t/F} P \ (call \ init \ p \ \text{return} \ c) \ Q, A$   
**using** *valid-call valid-modif res-modif ret-modifAbr unfolding call-call-exn*  
**by** (*rule Proc-exnModifyReturn-sound*)

**lemma** *Proc-exnModifyReturn*:  
**assumes** *spec*:  $\Gamma, \Theta \Vdash_{t/F} P \ (call\text{-exn} \ \text{init} \ p \ \text{return}' \ \text{result}\text{-exn} \ c) \ Q, A$   
**assumes** *res-modif*:  
 $\forall s \ t. t \in Modif \ (init \ s) \longrightarrow (\text{return}' \ s \ t) = (\text{return} \ s \ t)$   
**assumes** *ret-modifAbr*:  
 $\forall s \ t. t \in ModifAbr \ (init \ s) \longrightarrow (\text{result}\text{-exn} \ (\text{return}' \ s \ t) \ t) = (\text{result}\text{-exn} \ (\text{return} \ s \ t) \ t)$   
**assumes** *modifies-spec*:  
 $\forall \sigma. \Gamma, \Theta \Vdash_{UNIV} \{\sigma\} \ (Call \ p) \ (Modif \ \sigma), (ModifAbr \ \sigma)$   
**shows**  $\Gamma, \Theta \Vdash_{t/F} P \ (call\text{-exn} \ \text{init} \ p \ \text{return} \ \text{result}\text{-exn} \ c) \ Q, A$   
**apply** (*rule hoaret-complete'*)  
**apply** (*rule Proc-exnModifyReturn-sound [where Modif=Modif and ModifAbr=ModifAbr, OF - - res-modif ret-modifAbr]*)  
**apply** (*rule hoaret-sound [OF spec]*)  
**using** *modifies-spec*  
**apply** (*blast intro: hoare-sound*)  
**done**

**lemma** *ProcModifyReturn*:  
**assumes** *spec*:  $\Gamma, \Theta \Vdash_{t/F} P \ (call \ \text{init} \ p \ \text{return}' \ c) \ Q, A$   
**assumes** *res-modif*:  
 $\forall s \ t. t \in Modif \ (init \ s) \longrightarrow (\text{return}' \ s \ t) = (\text{return} \ s \ t)$   
**assumes** *ret-modifAbr*:  
 $\forall s \ t. t \in ModifAbr \ (init \ s) \longrightarrow (\text{return}' \ s \ t) = (\text{return} \ s \ t)$   
**assumes** *modifies-spec*:

$\forall \sigma. \Gamma, \Theta \vdash_{UNIV} \{\sigma\} (Call\ p) (Modif\ \sigma), (ModifAbr\ \sigma)$   
**shows**  $\Gamma, \Theta \vdash_{t/F} P (call\ init\ p\ return\ c) Q, A$   
**using** *spec res-modif ret-modifAbr modifies-spec unfolding call-call-exn by (rule Proc-exnModifyReturn)*

**lemma** *Proc-exnModifyReturnSameFaults-sound:*

**assumes** *valid-call:*  $\Gamma, \Theta \models_{t/F} P\ call\ exn\ init\ p\ return'\ result\ exn\ c\ Q, A$

**assumes** *valid-modif:*

$\forall \sigma. \Gamma, \Theta \models_{/F} \{\sigma\} Call\ p (Modif\ \sigma), (ModifAbr\ \sigma)$

**assumes** *res-modif:*

$\forall s\ t. t \in Modif (init\ s) \longrightarrow return'\ s\ t = return\ s\ t$

**assumes** *ret-modifAbr:*

$\forall s\ t. t \in ModifAbr (init\ s) \longrightarrow result\ exn (return'\ s\ t)\ t = result\ exn (return\ s\ t)\ t$

**shows**  $\Gamma, \Theta \models_{t/F} P (call\ exn\ init\ p\ return\ result\ exn\ c) Q, A$

**proof** (*rule cvalidtI*)

**fix**  $s\ t$

**assume** *ctxt:*  $\forall (P, p, Q, A) \in \Theta. \Gamma \models_{t/F} P (Call\ p) Q, A$

**hence** *ctxt':*  $\forall (P, p, Q, A) \in \Theta. \Gamma \models_{/F} P (Call\ p) Q, A$

**by** (*auto simp add: validt-def*)

**assume** *exec:*  $\Gamma \vdash \langle call\ exn\ init\ p\ return\ result\ exn\ c, Normal\ s \rangle \Rightarrow t$

**assume**  $P: s \in P$

**assume** *t-notin-F:*  $t \notin Fault\ 'F$

**from** *exec*

**show**  $t \in Normal\ 'Q \cup Abrupt\ 'A$

**proof** (*cases rule: exec-call-exn-Normal-elim*)

**fix**  $bdy\ t'$

**assume** *bdy:*  $\Gamma\ p = Some\ bdy$

**assume** *exec-body:*  $\Gamma \vdash \langle bdy, Normal (init\ s) \rangle \Rightarrow Normal\ t'$

**assume** *exec-c:*  $\Gamma \vdash \langle c\ s\ t', Normal (return\ s\ t') \rangle \Rightarrow t$

**from** *exec-body bdy*

**have**  $\Gamma \vdash \langle (Call\ p), Normal (init\ s) \rangle \Rightarrow Normal\ t'$

**by** (*auto simp add: intro: exec.intros*)

**from** *cvalidD [OF valid-modif [rule-format, of init s] ctxt' this] P*

**have**  $t' \in Modif (init\ s)$

**by** *auto*

**with** *res-modif* **have**  $Normal (return'\ s\ t') = Normal (return\ s\ t')$

**by** *simp*

**with** *exec-body exec-c bdy*

**have**  $\Gamma \vdash \langle call\ exn\ init\ p\ return'\ result\ exn\ c, Normal\ s \rangle \Rightarrow t$

**by** (*auto intro: exec-call-exn*)

**from** *cvalidt-postD [OF valid-call ctxt this] P t-notin-F*

**show** *?thesis*

**by** *simp*

**next**

**fix**  $bdy\ t'$

**assume** *bdy:*  $\Gamma\ p = Some\ bdy$

**assume** *exec-body:*  $\Gamma \vdash \langle bdy, Normal (init\ s) \rangle \Rightarrow Abrupt\ t'$

```

assume  $t: t = \text{Abrupt}(\text{result-exn}(\text{return } s \ t') \ t')$ 
also
from  $\text{exec-body } bdy$ 
have  $\Gamma \vdash \langle \text{Call } p, \text{Normal}(\text{init } s) \rangle \Rightarrow \text{Abrupt } t'$ 
  by  $(\text{auto simp add: intro: exec.intros})$ 
from  $\text{cvalidD} \ [OF \ \text{valid-modif} \ [\text{rule-format}, \ \text{of } \text{init } s] \ \text{ctxt}' \ \text{this}] \ P$ 
have  $t' \in \text{ModifAbr}(\text{init } s)$ 
  by  $\text{auto}$ 
with  $\text{ret-modifAbr}$  have  $\text{Abrupt}(\text{result-exn}(\text{return } s \ t') \ t') = \text{Abrupt}(\text{result-exn}(\text{return}' \ s \ t') \ t')$ 
  by  $\text{simp}$ 
finally have  $t = \text{Abrupt}(\text{result-exn}(\text{return}' \ s \ t') \ t')$  .
with  $\text{exec-body } bdy$ 
have  $\Gamma \vdash \langle \text{call-exn } \text{init } p \ \text{return}' \ \text{result-exn } c, \text{Normal } s \rangle \Rightarrow t$ 
  by  $(\text{auto intro: exec-call-exnAbrupt})$ 
from  $\text{cvalidt-postD} \ [OF \ \text{valid-call } \ \text{ctxt}' \ \text{this}] \ P \ t\text{-notin-}F$ 
show  $?thesis$ 
  by  $\text{simp}$ 
next
fix  $bdy \ f$ 
assume  $bdy: \Gamma \ p = \text{Some } bdy$ 
assume  $\Gamma \vdash \langle bdy, \text{Normal}(\text{init } s) \rangle \Rightarrow \text{Fault } f$  and
   $t: t = \text{Fault } f$ 
with  $bdy$  have  $\Gamma \vdash \langle \text{call-exn } \text{init } p \ \text{return}' \ \text{result-exn } c, \text{Normal } s \rangle \Rightarrow t$ 
  by  $(\text{auto intro: exec-call-exnFault})$ 
from  $\text{cvalidt-postD} \ [OF \ \text{valid-call } \ \text{ctxt}' \ \text{this}] \ P \ t\text{-notin-}F$ 
show  $?thesis$ 
  by  $\text{simp}$ 
next
fix  $bdy$ 
assume  $bdy: \Gamma \ p = \text{Some } bdy$ 
assume  $\Gamma \vdash \langle bdy, \text{Normal}(\text{init } s) \rangle \Rightarrow \text{Stuck}$ 
   $t = \text{Stuck}$ 
with  $bdy$  have  $\Gamma \vdash \langle \text{call-exn } \text{init } p \ \text{return}' \ \text{result-exn } c, \text{Normal } s \rangle \Rightarrow t$ 
  by  $(\text{auto intro: exec-call-exnStuck})$ 
from  $\text{valid-call } \ \text{ctxt}' \ \text{this} \ P \ t\text{-notin-}F$ 
show  $?thesis$ 
  by  $(\text{rule } \text{cvalidt-postD})$ 
next
assume  $\Gamma \ p = \text{None} \ t = \text{Stuck}$ 
hence  $\Gamma \vdash \langle \text{call-exn } \text{init } p \ \text{return}' \ \text{result-exn } c, \text{Normal } s \rangle \Rightarrow t$ 
  by  $(\text{auto intro: exec-call-exnUndefined})$ 
from  $\text{valid-call } \ \text{ctxt}' \ \text{this} \ P \ t\text{-notin-}F$ 
show  $?thesis$ 
  by  $(\text{rule } \text{cvalidt-postD})$ 
qed
next
fix  $s$ 
assume  $\text{ctxt}: \forall (P, p, Q, A) \in \Theta. \Gamma \models_{t/F} P \ (\text{Call } p) \ Q, A$ 

```

**hence**  $ctxt'$ :  $\forall (P, p, Q, A) \in \Theta. \Gamma \models_{/F} P (Call\ p)\ Q, A$   
**by** (*auto simp add: validt-def*)  
**assume**  $P: s \in P$   
**from** *valid-call ctxt P*  
**have**  $call: \Gamma \vdash call\text{-exn}\ init\ p\ return'\ result\text{-exn}\ c \downarrow Normal\ s$   
**by** (*rule cvalidt-termD*)  
**show**  $\Gamma \vdash call\text{-exn}\ init\ p\ return\ result\text{-exn}\ c \downarrow Normal\ s$   
**proof** (*cases p \in dom \Gamma*)  
**case** *True*  
**with** *call* **obtain** *bdy* **where**  
 $bdy: \Gamma\ p = Some\ bdy$  **and** *termi-bdy*:  $\Gamma \vdash bdy \downarrow Normal\ (init\ s)$  **and**  
*termi-c*:  $\forall t. \Gamma \vdash \langle bdy, Normal\ (init\ s) \rangle \Rightarrow Normal\ t \longrightarrow$   
 $\Gamma \vdash c\ s\ t \downarrow Normal\ (return'\ s\ t)$   
**by** *cases auto*  
{  
**fix**  $t$   
**assume** *exec-bdy*:  $\Gamma \vdash \langle bdy, Normal\ (init\ s) \rangle \Rightarrow Normal\ t$   
**hence**  $\Gamma \vdash c\ s\ t \downarrow Normal\ (return\ s\ t)$   
**proof** –  
**from** *exec-bdy bdy*  
**have**  $\Gamma \vdash \langle (Call\ p), Normal\ (init\ s) \rangle \Rightarrow Normal\ t$   
**by** (*auto simp add: intro: exec.intros*)  
**from** *cvalidD [OF valid-modif [rule-format, of init s] ctxt' this] P*  
*res-modif*  
**have**  $return'\ s\ t = return\ s\ t$   
**by** *auto*  
**with** *termi-c exec-bdy* **show** *?thesis* **by** *auto*  
**qed**  
}  
**with** *bdy termi-bdy*  
**show** *?thesis*  
**by** (*iprover intro: terminates-call-exn*)  
**next**  
**case** *False*  
**thus** *?thesis*  
**by** (*auto intro: terminates-call-exn Undefined*)  
**qed**  
**qed**

**lemma** *ProcModifyReturnSameFaults-sound*:  
**assumes** *valid-call*:  $\Gamma, \Theta \models_{/F} P\ call\ init\ p\ return'\ c\ Q, A$   
**assumes** *valid-modif*:  
 $\forall \sigma. \Gamma, \Theta \models_{/F} \{\sigma\}\ Call\ p\ (Modif\ \sigma), (ModifAbr\ \sigma)$   
**assumes** *res-modif*:  
 $\forall s\ t. t \in Modif\ (init\ s) \longrightarrow return'\ s\ t = return\ s\ t$   
**assumes** *ret-modifAbr*:  
 $\forall s\ t. t \in ModifAbr\ (init\ s) \longrightarrow return'\ s\ t = return\ s\ t$   
**shows**  $\Gamma, \Theta \models_{/F} P\ (call\ init\ p\ return\ c)\ Q, A$   
**using** *valid-call valid-modif res-modif ret-modifAbr* **unfolding** *call-call-exn*

by (rule Proc-exnModifyReturnSameFaults-sound)

**lemma** Proc-exnModifyReturnSameFaults:

**assumes** spec:  $\Gamma, \Theta \vdash_{t/F} P$  (call-exn init p return' result-exn c) Q,A

**assumes** res-modif:

$\forall s t. t \in \text{Modif} \text{ (init } s) \longrightarrow (\text{return}' s t) = (\text{return } s t)$

**assumes** ret-modifAbr:

$\forall s t. t \in \text{ModifAbr} \text{ (init } s) \longrightarrow \text{result-exn} (\text{return}' s t) t = \text{result-exn} (\text{return } s t) t$

**assumes** modifies-spec:

$\forall \sigma. \Gamma, \Theta \vdash_{/F} \{\sigma\}$  (Call p) (Modif  $\sigma$ ), (ModifAbr  $\sigma$ )

**shows**  $\Gamma, \Theta \vdash_{t/F} P$  (call-exn init p return result-exn c) Q,A

**apply** (rule hoaret-complete')

**apply** (rule Proc-exnModifyReturnSameFaults-sound [where Modif=Modif and ModifAbr=ModifAbr,

OF - - res-modif ret-modifAbr])

**apply** (rule hoaret-sound [OF spec])

**using** modifies-spec

**apply** (blast intro: hoare-sound)

**done**

**lemma** ProcModifyReturnSameFaults:

**assumes** spec:  $\Gamma, \Theta \vdash_{t/F} P$  (call init p return' c) Q,A

**assumes** res-modif:

$\forall s t. t \in \text{Modif} \text{ (init } s) \longrightarrow (\text{return}' s t) = (\text{return } s t)$

**assumes** ret-modifAbr:

$\forall s t. t \in \text{ModifAbr} \text{ (init } s) \longrightarrow (\text{return}' s t) = (\text{return } s t)$

**assumes** modifies-spec:

$\forall \sigma. \Gamma, \Theta \vdash_{/F} \{\sigma\}$  (Call p) (Modif  $\sigma$ ), (ModifAbr  $\sigma$ )

**shows**  $\Gamma, \Theta \vdash_{t/F} P$  (call init p return c) Q,A

**using** spec res-modif ret-modifAbr modifies-spec **unfolding** call-call-exn

**by** (rule Proc-exnModifyReturnSameFaults)

### 10.3.2 DynCall

**lemma** dynProc-exnModifyReturn-sound:

**assumes** valid-call:  $\Gamma, \Theta \models_{t/F} P$  dynCall-exn f g init p return' result-exn c Q,A

**assumes** valid-modif:

$\forall s \in P. \forall \sigma. \Gamma, \Theta \models_{/UNIV} \{\sigma\}$  (Call (p s)) (Modif  $\sigma$ ), (ModifAbr  $\sigma$ )

**assumes** ret-modif:

$\forall s t. t \in \text{Modif} \text{ (init } s) \longrightarrow \text{return}' s t = \text{return } s t$

**assumes** ret-modifAbr:  $\forall s t. t \in \text{ModifAbr} \text{ (init } s) \longrightarrow \text{result-exn} (\text{return}' s t) t = \text{result-exn} (\text{return } s t) t$

**shows**  $\Gamma, \Theta \models_{t/F} P$  (dynCall-exn f g init p return result-exn c) Q,A

**proof** (rule cvalidtI)

**fix** s t

**assume** ctxt:  $\forall (P, p, Q, A) \in \Theta. \Gamma \models_{t/F} P$  (Call p) Q,A



**hence**  $\forall (P, p, Q, A) \in \Theta. \Gamma \models_{/F} P (Call\ p)\ Q, A$   
**by** (*auto simp add: validt-def*)  
**then have**  $ctxt': \forall (P, p, Q, A) \in \Theta. \Gamma \models_{/UNIV} P (Call\ p)\ Q, A$   
**by** (*auto intro: valid-augment-Faults*)  
**assume**  $exec: \Gamma \vdash \langle dynCall\text{-}exn\ f\ g\ init\ p\ return\ result\text{-}exn\ c, Normal\ s \rangle \Rightarrow t$   
**assume**  $t\text{-notin}\text{-}F: t \notin Fault\ 'F$   
**assume**  $P: s \in P$   
**with** *valid-modif*  
**have** *valid-modif'*:  
 $\forall \sigma. \Gamma, \Theta \models_{/UNIV} \{\sigma\} (Call\ (p\ s)) (Modif\ \sigma), (ModifAbr\ \sigma)$   
**by** *blast*  
**from** *exec*  
**have**  $\Gamma \vdash \langle maybe\text{-}guard\ f\ g\ (call\text{-}exn\ init\ (p\ s)\ return\ result\text{-}exn\ c), Normal\ s \rangle \Rightarrow t$   
**by** (*cases rule: exec-dynCall-exn-Normal-elim*)  
**then show**  $t \in Normal\ 'Q \cup Abrupt\ 'A$   
**proof** (*cases rule: exec-maybe-guard-Normal-elim-cases*)  
**case** *noFault*  
**from** *noFault* **have** *guards-ok: s ∈ g by simp*  
**from** *noFault* **have**  $\Gamma \vdash \langle call\text{-}exn\ init\ (p\ s)\ return\ result\text{-}exn\ c, Normal\ s \rangle \Rightarrow t$   
**by** *simp*  
**then show**  $t \in Normal\ 'Q \cup Abrupt\ 'A$   
**proof** (*cases rule: exec-call-exn-Normal-elim*)  
**fix** *bdy t'*  
**assume**  $bdy: \Gamma (p\ s) = Some\ bdy$   
**assume**  $exec\text{-}body: \Gamma \vdash \langle bdy, Normal\ (init\ s) \rangle \Rightarrow Normal\ t'$   
**assume**  $exec\text{-}c: \Gamma \vdash \langle c\ s\ t', Normal\ (return\ s\ t') \rangle \Rightarrow t$   
**from** *exec-body bdy*  
**have**  $\Gamma \vdash \langle Call\ (p\ s), Normal\ (init\ s) \rangle \Rightarrow Normal\ t'$   
**by** (*auto simp add: intro: exec.Call*)  
**from** *cvalidD [OF valid-modif' [rule-format, of init s] ctxt' this] P*  
**have**  $t' \in Modif\ (init\ s)$   
**by** *auto*  
**with** *ret-modif* **have**  $Normal\ (return'\ s\ t') =$   
 $Normal\ (return\ s\ t')$   
**by** *simp*  
**with** *exec-body exec-c bdy*  
**have**  $\Gamma \vdash \langle call\text{-}exn\ init\ (p\ s)\ return'\ result\text{-}exn\ c, Normal\ s \rangle \Rightarrow t$   
**by** (*auto intro: exec-call-exn*)  
**from** *exec-maybe-guard-noFault [OF this guards-ok]*  
**have**  $\Gamma \vdash \langle dynCall\text{-}exn\ f\ g\ init\ p\ return'\ result\text{-}exn\ c, Normal\ s \rangle \Rightarrow t$   
**by** (*simp add: dynCall-exn-def exec-maybe-guard-DynCom*)  
**from** *cvalidt-postD [OF valid-call ctxt' this] P t-notin-F*  
**show** *?thesis*  
**by** *simp*  
**next**  
**fix** *bdy t'*  
**assume**  $bdy: \Gamma (p\ s) = Some\ bdy$   
**assume**  $exec\text{-}body: \Gamma \vdash \langle bdy, Normal\ (init\ s) \rangle \Rightarrow Abrupt\ t'$   
**assume**  $t: t = Abrupt\ (result\text{-}exn\ (return\ s\ t')\ t')$

```

also from exec-body bdy
have  $\Gamma \vdash \langle \text{Call } (p \ s) , \text{Normal } (\text{init } s) \rangle \Rightarrow \text{Abrupt } t'$ 
  by (auto simp add: intro: exec.intros)
from cvalidD [OF valid-modif' [rule-format, of init s] ctxt' this] P
have  $t' \in \text{ModifAbr } (\text{init } s)$ 
  by auto
with ret-modifAbr have  $\text{Abrupt } (\text{result-exn } (\text{return } s \ t') \ t') = \text{Abrupt } (\text{result-exn } (\text{return}' \ s \ t') \ t')$ 
  by simp
finally have  $t = \text{Abrupt } (\text{result-exn } (\text{return}' \ s \ t') \ t')$  .
with exec-body bdy
have  $\Gamma \vdash \langle \text{call-exn } \text{init } (p \ s) \ \text{return}' \ \text{result-exn } c, \text{Normal } s \rangle \Rightarrow t$ 
  by (auto intro: exec-call-exnAbrupt)
from exec-maybe-guard-noFault [OF this guards-ok]
have  $\Gamma \vdash \langle \text{dynCall-exn } f \ g \ \text{init } p \ \text{return}' \ \text{result-exn } c, \text{Normal } s \rangle \Rightarrow t$ 
  by (simp add: dynCall-exn-def exec-maybe-guard-DynCom)
from cvalidt-postD [OF valid-call ctxt this] P t-notin-F
show ?thesis
  by simp
next
fix bdy f'
assume bdy:  $\Gamma (p \ s) = \text{Some } bdy$ 
assume  $\Gamma \vdash \langle bdy, \text{Normal } (\text{init } s) \rangle \Rightarrow \text{Fault } f'$  and
   $t: t = \text{Fault } f'$ 
with bdy have  $\Gamma \vdash \langle \text{call-exn } \text{init } (p \ s) \ \text{return}' \ \text{result-exn } c , \text{Normal } s \rangle \Rightarrow t$ 
  by (auto intro: exec-call-exnFault)
from exec-maybe-guard-noFault [OF this guards-ok]
have  $\Gamma \vdash \langle \text{dynCall-exn } f \ g \ \text{init } p \ \text{return}' \ \text{result-exn } c, \text{Normal } s \rangle \Rightarrow t$ 
  by (simp add: dynCall-exn-def exec-maybe-guard-DynCom)
from cvalidt-postD [OF valid-call ctxt this P] t t-notin-F
show ?thesis
  by blast
next
fix bdy
assume bdy:  $\Gamma (p \ s) = \text{Some } bdy$ 
assume  $\Gamma \vdash \langle bdy, \text{Normal } (\text{init } s) \rangle \Rightarrow \text{Stuck}$ 
   $t = \text{Stuck}$ 
with bdy have  $\Gamma \vdash \langle \text{call-exn } \text{init } (p \ s) \ \text{return}' \ \text{result-exn } c , \text{Normal } s \rangle \Rightarrow t$ 
  by (auto intro: exec-call-exnStuck)
from exec-maybe-guard-noFault [OF this guards-ok]
have  $\Gamma \vdash \langle \text{dynCall-exn } f \ g \ \text{init } p \ \text{return}' \ \text{result-exn } c, \text{Normal } s \rangle \Rightarrow t$ 
  by (simp add: dynCall-exn-def exec-maybe-guard-DynCom)
from valid-call ctxt this P t-notin-F
show ?thesis
  by (rule cvalidt-postD)
next
fix bdy
assume  $\Gamma (p \ s) = \text{None } t = \text{Stuck}$ 
hence  $\Gamma \vdash \langle \text{call-exn } \text{init } (p \ s) \ \text{return}' \ \text{result-exn } c , \text{Normal } s \rangle \Rightarrow t$ 

```

```

    by (auto intro: exec-call-exnUndefined)
  from exec-maybe-guard-noFault [OF this guards-ok]
  have  $\Gamma \vdash \langle \text{dynCall-exn } f \ g \ \text{init } p \ \text{return}' \ \text{result-exn } c, \text{Normal } s \rangle \Rightarrow t$ 
    by (simp add: dynCall-exn-def exec-maybe-guard-DynCom)
  from valid-call ctxt this P t-notin-F
  show ?thesis
    by (rule cvalidt-postD)
qed
next
case (someFault)
then obtain guards-fail:s  $\notin g$ 
  and t:  $t = \text{Fault } f$  by simp
from exec-maybe-guard-Fault [OF guards-fail] t
have  $\Gamma \vdash \langle \text{dynCall-exn } f \ g \ \text{init } p \ \text{return}' \ \text{result-exn } c, \text{Normal } s \rangle \Rightarrow t$ 
  by (simp add: dynCall-exn-def exec-guards-DynCom)
from cvalidt-postD [OF valid-call ctxt this] P t-notin-F
show ?thesis by simp
qed
next
fix s
assume ctxt:  $\forall (P, p, Q, A) \in \Theta. \Gamma \models_{t/F} P (Call \ p) \ Q, A$ 
hence  $\forall (P, p, Q, A) \in \Theta. \Gamma \models_{/F} P (Call \ p) \ Q, A$ 
  by (auto simp add: validt-def)
then have ctxt':  $\forall (P, p, Q, A) \in \Theta. \Gamma \models_{/UNIV} P (Call \ p) \ Q, A$ 
  by (auto intro: valid-augment-Faults)
assume P:  $s \in P$ 
from valid-call ctxt P
have  $\Gamma \vdash \text{dynCall-exn } f \ g \ \text{init } p \ \text{return}' \ \text{result-exn } c \downarrow \text{Normal } s$ 
  by (rule cvalidt-termD)
thus  $\Gamma \vdash \text{dynCall-exn } f \ g \ \text{init } p \ \text{return} \ \text{result-exn } c \downarrow \text{Normal } s$ 
proof (cases rule: terminates-dynCall-exn-elim)
case noFault
then obtain guards-ok:  $s \in g$ 
  and call:  $\Gamma \vdash \text{call-exn } \text{init } (p \ s) \ \text{return}' \ \text{result-exn } c \downarrow \text{Normal } s$ 
  by auto
have  $\Gamma \vdash \text{call-exn } \text{init } (p \ s) \ \text{return} \ \text{result-exn } c \downarrow \text{Normal } s$ 
proof (cases  $p \ s \in \text{dom } \Gamma$ )
case True
with call obtain bdy where
  bdy:  $\Gamma (p \ s) = \text{Some } bdy$  and termi-bdy:  $\Gamma \vdash bdy \downarrow \text{Normal } (\text{init } s)$  and
  termi-c:  $\forall t. \Gamma \vdash \langle bdy, \text{Normal } (\text{init } s) \rangle \Rightarrow \text{Normal } t \longrightarrow$ 
     $\Gamma \vdash c \ s \ t \downarrow \text{Normal } (\text{return}' \ s \ t)$ 
  by cases auto
{
  fix t
  assume exec-bdy:  $\Gamma \vdash \langle bdy, \text{Normal } (\text{init } s) \rangle \Rightarrow \text{Normal } t$ 
  hence  $\Gamma \vdash c \ s \ t \downarrow \text{Normal } (\text{return} \ s \ t)$ 
  proof -
    from exec-bdy bdy

```

```

have  $\Gamma \vdash \langle \text{Call } (p \ s), \text{Normal } (\text{init } s) \rangle \Rightarrow \text{Normal } t$ 
  by (auto simp add: intro: exec.intros)
from cvalidD [OF valid-modif [rule-format, of s init s] ctxt' this] P
  ret-modif
have  $\text{return}' \ s \ t = \text{return } s \ t$ 
  by auto
with termi-c exec-bdy show ?thesis by auto
qed
}
with bdy termi-bdy
show ?thesis
  by (iprover intro: terminates-call-exn)
next
  case False
  thus ?thesis
    by (auto intro: terminates-call-exnUndefined)
qed
thus  $\Gamma \vdash \text{dynCall-exn } f \ g \ \text{init } p \ \text{return } \text{result-exn } c \ \downarrow \ \text{Normal } s$ 
  by (iprover intro: terminates-dynCall-exn)
next
  case (someFault)
  then have guard-fail:  $s \notin g$  by simp
  thus ?thesis
    by (simp add: terminates-maybe-guard-Fault dynCall-exn-def)
qed
qed

```

**lemma** *dynProcModifyReturn-sound*:

```

assumes valid-call:  $\Gamma, \Theta \models_{t/F} P \ \text{dynCall } \text{init } p \ \text{return}' \ c \ Q, A$ 
assumes valid-modif:
   $\forall s \in P. \forall \sigma. \Gamma, \Theta \models_{UNIV} \{\sigma\} \ (\text{Call } (p \ s)) \ (\text{Modif } \sigma), (\text{ModifAbr } \sigma)$ 
assumes ret-modif:
   $\forall s \ t. t \in \text{Modif } (\text{init } s) \longrightarrow \text{return}' \ s \ t = \text{return } s \ t$ 
assumes ret-modifAbr:  $\forall s \ t. t \in \text{ModifAbr } (\text{init } s) \longrightarrow \text{return}' \ s \ t = \text{return } s \ t$ 
shows  $\Gamma, \Theta \models_{t/F} P \ (\text{dynCall } \text{init } p \ \text{return } c) \ Q, A$ 
  using valid-call valid-modif ret-modif ret-modifAbr unfolding dynCall-dynCall-exn
  by (rule dynProc-exnModifyReturn-sound)

```

**lemma** *dynProc-exnModifyReturn*:

```

assumes dyn-call:  $\Gamma, \Theta \vdash_{t/F} P \ \text{dynCall-exn } f \ g \ \text{init } p \ \text{return}' \ \text{result-exn } c \ Q, A$ 
assumes ret-modif:
   $\forall s \ t. t \in \text{Modif } (\text{init } s) \longrightarrow \text{return}' \ s \ t = \text{return } s \ t$ 
assumes ret-modifAbr:  $\forall s \ t. t \in \text{ModifAbr } (\text{init } s) \longrightarrow \text{result-exn } (\text{return}' \ s \ t) \ t = \text{result-exn } (\text{return } s \ t) \ t$ 
assumes modif:
   $\forall s \in P. \forall \sigma. \Gamma, \Theta \vdash_{UNIV} \{\sigma\} \ \text{Call } (p \ s) \ (\text{Modif } \sigma), (\text{ModifAbr } \sigma)$ 

```

**shows**  $\Gamma, \Theta \vdash_{t/F} P$  (*dynCall-exn f g init p return result-exn c*)  $Q, A$   
**apply** (*rule hoaret-complete'*)  
**apply** (*rule dynProc-exnModifyReturn-sound*  
     [**where** *Modif=Modif and ModifAbr=ModifAbr,*  
       *OF hoaret-sound [OF dyn-call] - ret-modif ret-modifAbr*])  
**apply** (*intro ballI allI*)  
**apply** (*rule hoare-sound [OF modif [rule-format]]*)  
**apply** *assumption*  
**done**

**lemma** *dynProcModifyReturn:*

**assumes** *dyn-call:*  $\Gamma, \Theta \vdash_{t/F} P$  *dynCall init p return' c*  $Q, A$

**assumes** *ret-modif:*

$\forall s t. t \in \text{Modif} \text{ (init } s)$   
 $\longrightarrow \text{return}' s t = \text{return } s t$

**assumes** *ret-modifAbr:*  $\forall s t. t \in \text{ModifAbr} \text{ (init } s)$

$\longrightarrow \text{return}' s t = \text{return } s t$

**assumes** *modif:*

$\forall s \in P. \forall \sigma.$

$\Gamma, \Theta \vdash_{UNIV} \{\sigma\} \text{ Call } (p s) \text{ (Modif } \sigma), (\text{ModifAbr } \sigma)$

**shows**  $\Gamma, \Theta \vdash_{t/F} P$  (*dynCall init p return c*)  $Q, A$

**using** *dyn-call ret-modif ret-modifAbr modif unfolding dynCall-dynCall-exn*

**by** (*rule dynProc-exnModifyReturn*)

**lemma** *dynProc-exnModifyReturnSameFaults-sound:*

**assumes** *valid-call:*  $\Gamma, \Theta \models_{t/F} P$  *dynCall-exn f g init p return' result-exn c*  $Q, A$

**assumes** *valid-modif:*

$\forall s \in P. \forall \sigma. \Gamma, \Theta \models_{t/F} \{\sigma\} \text{ Call } (p s) \text{ (Modif } \sigma), (\text{ModifAbr } \sigma)$

**assumes** *ret-modif:*

$\forall s t. t \in \text{Modif} \text{ (init } s) \longrightarrow \text{return}' s t = \text{return } s t$

**assumes** *ret-modifAbr:*  $\forall s t. t \in \text{ModifAbr} \text{ (init } s) \longrightarrow \text{result-exn} (\text{return}' s t) t$

$= \text{result-exn} (\text{return } s t) t$

**shows**  $\Gamma, \Theta \models_{t/F} P$  (*dynCall-exn f g init p return result-exn c*)  $Q, A$

**proof** (*rule cvalidtI*)

**fix**  $s t$

**assume** *ctxt:*  $\forall (P, p, Q, A) \in \Theta. \Gamma \models_{t/F} P \text{ (Call } p) Q, A$

**hence** *ctxt':*  $\forall (P, p, Q, A) \in \Theta. \Gamma \models_{t/F} P \text{ (Call } p) Q, A$

**by** (*auto simp add: validt-def*)

**assume** *exec:*  $\Gamma \vdash \langle \text{dynCall-exn } f g \text{ init } p \text{ return result-exn } c, \text{Normal } s \rangle \Rightarrow t$

**assume** *t-notin-F:*  $t \notin \text{Fault } 'F$

**assume** *P:*  $s \in P$

**with** *valid-modif*

**have** *valid-modif':*

$\forall \sigma. \Gamma, \Theta \models_{t/F} \{\sigma\} \text{ (Call } (p s)) \text{ (Modif } \sigma), (\text{ModifAbr } \sigma)$

**by** *blast*

**from** *exec*

**have**  $\Gamma \vdash \langle \text{maybe-guard } f g \text{ (call-exn init } (p s) \text{ return result-exn } c), \text{Normal } s \rangle \Rightarrow t$

**by** (*cases rule: exec-dynCall-exn-Normal-elim*)

**then show**  $t \in \text{Normal} \text{ ' } Q \cup \text{Abrupt} \text{ ' } A$   
**proof** (cases rule: *exec-maybe-guard-Normal-elim-cases*)  
    **case** *noFault*  
    **from** *noFault* **have** *guards-ok*:  $s \in g$  **by** *simp*  
    **from** *noFault* **have**  $\Gamma \vdash \langle \text{call-exn init } (p \ s) \ \text{return result-exn } c, \text{Normal } s \rangle \Rightarrow t$   
**by** *simp*  
**then show**  $t \in \text{Normal} \text{ ' } Q \cup \text{Abrupt} \text{ ' } A$   
**proof** (cases rule: *exec-call-exn-Normal-elim*)  
    **fix** *bdy t'*  
    **assume** *bdy*:  $\Gamma (p \ s) = \text{Some } \text{bdy}$   
    **assume** *exec-body*:  $\Gamma \vdash \langle \text{bdy}, \text{Normal } (\text{init } s) \rangle \Rightarrow \text{Normal } t'$   
    **assume** *exec-c*:  $\Gamma \vdash \langle c \ s \ t', \text{Normal } (\text{return } s \ t') \rangle \Rightarrow t$   
    **from** *exec-body* *bdy*  
    **have**  $\Gamma \vdash \langle \text{Call } (p \ s), \text{Normal } (\text{init } s) \rangle \Rightarrow \text{Normal } t'$   
    **by** (*auto simp add: intro: exec.intros*)  
    **from** *cvalidD* [*OF valid-modif'* [*rule-format, of init s*] *ctxt' this*] *P*  
    **have**  $t' \in \text{Modif } (\text{init } s)$   
    **by** *auto*  
    **with** *ret-modif* **have**  $\text{Normal } (\text{return}' \ s \ t') =$   
         $\text{Normal } (\text{return } s \ t')$   
    **by** *simp*  
    **with** *exec-body* *exec-c* *bdy*  
    **have**  $\Gamma \vdash \langle \text{call-exn init } (p \ s) \ \text{return}' \ \text{result-exn } c, \text{Normal } s \rangle \Rightarrow t$   
    **by** (*auto intro: exec-call-exn*)  
    **from** *exec-maybe-guard-noFault* [*OF this guards-ok*]  
    **have**  $\Gamma \vdash \langle \text{dynCall-exn } f \ g \ \text{init } p \ \text{return}' \ \text{result-exn } c, \text{Normal } s \rangle \Rightarrow t$   
    **by** (*simp add: dynCall-exn-def exec-maybe-guard-DynCom*)  
    **from** *cvalidt-postD* [*OF valid-call ctxt this*] *P* *t-notin-F*  
    **show** *?thesis*  
    **by** *simp*  
**next**  
    **fix** *bdy t'*  
    **assume** *bdy*:  $\Gamma (p \ s) = \text{Some } \text{bdy}$   
    **assume** *exec-body*:  $\Gamma \vdash \langle \text{bdy}, \text{Normal } (\text{init } s) \rangle \Rightarrow \text{Abrupt } t'$   
    **assume** *t*:  $t = \text{Abrupt } (\text{result-exn } (\text{return } s \ t') \ t')$   
    **also from** *exec-body* *bdy*  
    **have**  $\Gamma \vdash \langle \text{Call } (p \ s) \ , \text{Normal } (\text{init } s) \rangle \Rightarrow \text{Abrupt } t'$   
    **by** (*auto simp add: intro: exec.intros*)  
    **from** *cvalidD* [*OF valid-modif'* [*rule-format, of init s*] *ctxt' this*] *P*  
    **have**  $t' \in \text{ModifAbr } (\text{init } s)$   
    **by** *auto*  
    **with** *ret-modifAbr* **have**  $\text{Abrupt } (\text{result-exn } (\text{return } s \ t') \ t') = \text{Abrupt } (\text{result-exn}$   
    (*return' s t'*) *t'*)  
    **by** *simp*  
    **finally have**  $t = \text{Abrupt } (\text{result-exn } (\text{return}' \ s \ t') \ t') .$   
    **with** *exec-body* *bdy*  
    **have**  $\Gamma \vdash \langle \text{call-exn init } (p \ s) \ \text{return}' \ \text{result-exn } c, \text{Normal } s \rangle \Rightarrow t$   
    **by** (*auto intro: exec-call-exnAbrupt*)  
    **from** *exec-maybe-guard-noFault* [*OF this guards-ok*]

```

have  $\Gamma \vdash \langle \text{dynCall-exn } f \ g \ \text{init } p \ \text{return}' \ \text{result-exn } c, \text{Normal } s \rangle \Rightarrow t$ 
  by (simp add: dynCall-exn-def exec-maybe-guard-DynCom)
from cvalidt-postD [OF valid-call ctxt this] P t-notin-F
show ?thesis
  by simp
next
fix bdy f'
assume bdy:  $\Gamma (p \ s) = \text{Some } bdy$ 
assume  $\Gamma \vdash \langle bdy, \text{Normal } (\text{init } s) \rangle \Rightarrow \text{Fault } f'$  and
  t: t = Fault f'
with bdy have  $\Gamma \vdash \langle \text{call-exn } \text{init } (p \ s) \ \text{return}' \ \text{result-exn } c, \text{Normal } s \rangle \Rightarrow t$ 
  by (auto intro: exec-call-exnFault)
from exec-maybe-guard-noFault [OF this guards-ok]
have  $\Gamma \vdash \langle \text{dynCall-exn } f \ g \ \text{init } p \ \text{return}' \ \text{result-exn } c, \text{Normal } s \rangle \Rightarrow t$ 
  by (simp add: dynCall-exn-def exec-maybe-guard-DynCom)
from cvalidt-postD [OF valid-call ctxt this P] t t-notin-F
show ?thesis
  by simp
next
fix bdy
assume bdy:  $\Gamma (p \ s) = \text{Some } bdy$ 
assume  $\Gamma \vdash \langle bdy, \text{Normal } (\text{init } s) \rangle \Rightarrow \text{Stuck}$ 
  t = Stuck
with bdy have  $\Gamma \vdash \langle \text{call-exn } \text{init } (p \ s) \ \text{return}' \ \text{result-exn } c, \text{Normal } s \rangle \Rightarrow t$ 
  by (auto intro: exec-call-exnStuck)
from exec-maybe-guard-noFault [OF this guards-ok]
have  $\Gamma \vdash \langle \text{dynCall-exn } f \ g \ \text{init } p \ \text{return}' \ \text{result-exn } c, \text{Normal } s \rangle \Rightarrow t$ 
  by (simp add: dynCall-exn-def exec-maybe-guard-DynCom)
from valid-call ctxt this P t-notin-F
show ?thesis
  by (rule cvalidt-postD)
next
fix bdy
assume  $\Gamma (p \ s) = \text{None } t = \text{Stuck}$ 
hence  $\Gamma \vdash \langle \text{call-exn } \text{init } (p \ s) \ \text{return}' \ \text{result-exn } c, \text{Normal } s \rangle \Rightarrow t$ 
  by (auto intro: exec-call-exnUndefined)
from exec-maybe-guard-noFault [OF this guards-ok]
have  $\Gamma \vdash \langle \text{dynCall-exn } f \ g \ \text{init } p \ \text{return}' \ \text{result-exn } c, \text{Normal } s \rangle \Rightarrow t$ 
  by (simp add: dynCall-exn-def exec-maybe-guard-DynCom)
from valid-call ctxt this P t-notin-F
show ?thesis
  by (rule cvalidt-postD)
qed
next
case (someFault)
then obtain guards-fail: s  $\notin$  g
  and t: t = Fault f by simp
from exec-maybe-guard-Fault [OF guards-fail] t
have  $\Gamma \vdash \langle \text{dynCall-exn } f \ g \ \text{init } p \ \text{return}' \ \text{result-exn } c, \text{Normal } s \rangle \Rightarrow t$ 

```

```

    by (simp add: dynCall-exn-def exec-guards-DynCom)
  from cvalidt-postD [OF valid-call ctxt this] P t-notin-F
  show ?thesis by simp
qed
next
fix s
assume ctxt:  $\forall (P, p, Q, A) \in \Theta. \Gamma \models_{t/F} P (Call\ p)\ Q, A$ 
hence ctxt':  $\forall (P, p, Q, A) \in \Theta. \Gamma \models_{/F} P (Call\ p)\ Q, A$ 
  by (auto simp add: validt-def)
assume P:  $s \in P$ 
from valid-call ctxt P
have  $\Gamma \vdash \text{dynCall-exn } f\ g\ \text{init } p\ \text{return}'\ \text{result-exn } c \downarrow \text{Normal } s$ 
  by (rule cvalidt-termD)
thus  $\Gamma \vdash \text{dynCall-exn } f\ g\ \text{init } p\ \text{return}\ \text{result-exn } c \downarrow \text{Normal } s$ 
proof (cases rule: terminates-dynCall-exn-elim)
  case noFault
  then obtain guards-ok:  $s \in g$ 
    and call:  $\Gamma \vdash \text{call-exn } \text{init } (p\ s)\ \text{return}'\ \text{result-exn } c \downarrow \text{Normal } s$ 
    by auto
  have  $\Gamma \vdash \text{call-exn } \text{init } (p\ s)\ \text{return}\ \text{result-exn } c \downarrow \text{Normal } s$ 
  proof (cases  $p\ s \in \text{dom } \Gamma$ )
    case True
    with call obtain bdy where
      bdy:  $\Gamma (p\ s) = \text{Some } bdy$  and termi-bdy:  $\Gamma \vdash bdy \downarrow \text{Normal } (\text{init } s)$  and
      termi-c:  $\forall t. \Gamma \vdash \langle bdy, \text{Normal } (\text{init } s) \rangle \Rightarrow \text{Normal } t \longrightarrow$ 
         $\Gamma \vdash c\ s\ t \downarrow \text{Normal } (\text{return}'\ s\ t)$ 
    by cases auto
  {
    fix t
    assume exec-bdy:  $\Gamma \vdash \langle bdy, \text{Normal } (\text{init } s) \rangle \Rightarrow \text{Normal } t$ 
    hence  $\Gamma \vdash c\ s\ t \downarrow \text{Normal } (\text{return } s\ t)$ 
    proof -
      from exec-bdy bdy
      have  $\Gamma \vdash \langle \text{Call } (p\ s), \text{Normal } (\text{init } s) \rangle \Rightarrow \text{Normal } t$ 
        by (auto simp add: intro: exec.intros)
      from cvalidD [OF valid-modif [rule-format, of s init s] ctxt' this] P
        ret-modif
      have  $\text{return}'\ s\ t = \text{return } s\ t$ 
        by auto
      with termi-c exec-bdy show ?thesis by auto
    qed
  }
  with bdy termi-bdy
  show ?thesis
  by (iprover intro: terminates-call-exn)
next
case False
thus ?thesis
  by (auto intro: terminates-call-exnUndefined)

```



**qed**  
**thus**  $\Gamma \vdash \text{dynCall-exn } f \ g \ \text{init } p \ \text{return } \text{result-exn } c \ \downarrow \ \text{Normal } s$   
**by** (*iprover intro: terminates-dynCall-exn*)  
**next**  
**case** (*someFault*)  
**then have** *guard-fail:  $s \notin g$  by simp*  
**thus** *?thesis*  
**by** (*simp add: terminates-maybe-guard-Fault dynCall-exn-def*)  
**qed**  
**qed**

**lemma** *dynProcModifyReturnSameFaults-sound:*  
**assumes** *valid-call:  $\Gamma, \Theta \Vdash_{t/F} P \ \text{dynCall } \text{init } p \ \text{return}' c \ Q, A$*   
**assumes** *valid-modif:*  
 $\forall s \in P. \forall \sigma. \Gamma, \Theta \Vdash_{t/F} \{\sigma\} \ \text{Call } (p \ s) \ (\text{Modif } \sigma), (\text{ModifAbr } \sigma)$   
**assumes** *ret-modif:*  
 $\forall s \ t. t \in \text{Modif } (\text{init } s) \longrightarrow \text{return}' s \ t = \text{return } s \ t$   
**assumes** *ret-modifAbr:  $\forall s \ t. t \in \text{ModifAbr } (\text{init } s) \longrightarrow \text{return}' s \ t = \text{return } s \ t$*   
**shows**  $\Gamma, \Theta \Vdash_{t/F} P \ (\text{dynCall } \text{init } p \ \text{return } c) \ Q, A$   
**using** *valid-call valid-modif ret-modif ret-modifAbr unfolding dynCall-dynCall-exn*  
**by** (*rule dynProc-exnModifyReturnSameFaults-sound*)

**lemma** *dynProc-exnModifyReturnSameFaults:*  
**assumes** *dyn-call:  $\Gamma, \Theta \vdash_{t/F} P \ \text{dynCall-exn } f \ g \ \text{init } p \ \text{return}' \ \text{result-exn } c \ Q, A$*   
**assumes** *ret-modif:*  
 $\forall s \ t. t \in \text{Modif } (\text{init } s) \longrightarrow \text{return}' s \ t = \text{return } s \ t$   
**assumes** *ret-modifAbr:  $\forall s \ t. t \in \text{ModifAbr } (\text{init } s) \longrightarrow \text{result-exn } (\text{return}' s \ t) \ t = \text{result-exn } (\text{return } s \ t) \ t$*   
**assumes** *modif:*  
 $\forall s \in P. \forall \sigma. \Gamma, \Theta \vdash_{t/F} \{\sigma\} \ \text{Call } (p \ s) \ (\text{Modif } \sigma), (\text{ModifAbr } \sigma)$   
**shows**  $\Gamma, \Theta \vdash_{t/F} P \ (\text{dynCall-exn } f \ g \ \text{init } p \ \text{return } \text{result-exn } c) \ Q, A$   
**apply** (*rule hoaret-complete'*)  
**apply** (*rule dynProc-exnModifyReturnSameFaults-sound*  
[**where** *Modif=Modif and ModifAbr=ModifAbr,*  
*OF hoaret-sound [OF dyn-call] - ret-modif ret-modifAbr*])  
**apply** (*intro ballI allI*)  
**apply** (*rule hoare-sound [OF modif [rule-format]]*)  
**apply** *assumption*  
**done**

**lemma** *dynProcModifyReturnSameFaults:*  
**assumes** *dyn-call:  $\Gamma, \Theta \vdash_{t/F} P \ \text{dynCall } \text{init } p \ \text{return}' c \ Q, A$*   
**assumes** *ret-modif:*  
 $\forall s \ t. t \in \text{Modif } (\text{init } s) \longrightarrow \text{return}' s \ t = \text{return } s \ t$   
**assumes** *ret-modifAbr:  $\forall s \ t. t \in \text{ModifAbr } (\text{init } s) \longrightarrow \text{return}' s \ t = \text{return } s \ t$*   
**assumes** *modif:*  
 $\forall s \in P. \forall \sigma. \Gamma, \Theta \vdash_{t/F} \{\sigma\} \ \text{Call } (p \ s) \ (\text{Modif } \sigma), (\text{ModifAbr } \sigma)$   
**shows**  $\Gamma, \Theta \vdash_{t/F} P \ (\text{dynCall } \text{init } p \ \text{return } c) \ Q, A$

using *dyn-call ret-modif ret-modifAbr modif unfolding dynCall-dynCall-exn*  
 by (rule *dynProc-exnModifyReturnSameFaults*)

### 10.3.3 Conjunction of Postcondition

**lemma** *PostConjI-sound*:

**assumes** *valid-Q*:  $\Gamma, \Theta \models_{t/F} P \text{ c } Q, A$

**assumes** *valid-R*:  $\Gamma, \Theta \models_{t/F} P \text{ c } R, B$

**shows**  $\Gamma, \Theta \models_{t/F} P \text{ c } (Q \cap R), (A \cap B)$

**proof** (rule *cvalidtI*)

**fix** *s t*

**assume** *ctxt*:  $\forall (P, p, Q, A) \in \Theta. \Gamma \models_{t/F} P \text{ (Call } p) \text{ } Q, A$

**assume** *exec*:  $\Gamma \vdash \langle c, \text{Normal } s \rangle \Rightarrow t$

**assume** *P*:  $s \in P$

**assume** *t-notin-F*:  $t \notin \text{Fault } F$

**from** *valid-Q ctxt exec P t-notin-F* **have**  $t \in \text{Normal } Q \cup \text{Abrupt } A$   
 by (rule *cvalidt-postD*)

**moreover**

**from** *valid-R ctxt exec P t-notin-F* **have**  $t \in \text{Normal } R \cup \text{Abrupt } B$   
 by (rule *cvalidt-postD*)

**ultimately show**  $t \in \text{Normal } (Q \cap R) \cup \text{Abrupt } (A \cap B)$   
 by *blast*

**next**

**fix** *s*

**assume** *ctxt*:  $\forall (P, p, Q, A) \in \Theta. \Gamma \models_{t/F} P \text{ (Call } p) \text{ } Q, A$

**assume** *P*:  $s \in P$

**from** *valid-Q ctxt P*

**show**  $\Gamma \vdash c \downarrow \text{Normal } s$

by (rule *cvalidt-termD*)

**qed**

**lemma** *PostConjI*:

**assumes** *deriv-Q*:  $\Gamma, \Theta \vdash_{t/F} P \text{ c } Q, A$

**assumes** *deriv-R*:  $\Gamma, \Theta \vdash_{t/F} P \text{ c } R, B$

**shows**  $\Gamma, \Theta \vdash_{t/F} P \text{ c } (Q \cap R), (A \cap B)$

**apply** (rule *hoaret-complete'*)

**apply** (rule *PostConjI-sound*)

**apply** (rule *hoaret-sound* [*OF deriv-Q*])

**apply** (rule *hoaret-sound* [*OF deriv-R*])

**done**

**lemma** *Merge-PostConj-sound*:

**assumes** *validF*:  $\Gamma, \Theta \models_{t/F} P \text{ c } Q, A$

**assumes** *validG*:  $\Gamma, \Theta \models_{t/G} P' \text{ c } R, X$

**assumes** *F-G*:  $F \subseteq G$

**assumes** *P-P'*:  $P \subseteq P'$

**shows**  $\Gamma, \Theta \models_{t/F} P \text{ c } (Q \cap R), (A \cap X)$

```

proof (rule cvalidtI)
  fix s t
  assume ctxt:  $\forall (P, p, Q, A) \in \Theta. \Gamma \models_{t/F} P \text{ (Call } p) \text{ } Q, A$ 
  with F-G have ctxt':  $\forall (P, p, Q, A) \in \Theta. \Gamma \models_{t/G} P \text{ (Call } p) \text{ } Q, A$ 
    by (auto intro: validt-augment-Faults)
  assume exec:  $\Gamma \vdash \langle c, \text{Normal } s \rangle \Rightarrow t$ 
  assume P:  $s \in P$ 
  with P-P' have P':  $s \in P'$ 
    by auto
  assume t-noFault:  $t \notin \text{Fault } F$ 
  show  $t \in \text{Normal } (Q \cap R) \cup \text{Abrupt } (A \cap X)$ 
  proof -
    from cvalidt-postD [OF validF [rule-format] ctxt exec P t-noFault]
    have t:  $t \in \text{Normal } Q \cup \text{Abrupt } A$ .
    then have  $t \notin \text{Fault } G$ 
      by auto
    from cvalidt-postD [OF validG [rule-format] ctxt' exec P' this]
    have  $t \in \text{Normal } R \cup \text{Abrupt } X$  .
    with t show ?thesis by auto
  qed
next
  fix s
  assume ctxt:  $\forall (P, p, Q, A) \in \Theta. \Gamma \models_{t/F} P \text{ (Call } p) \text{ } Q, A$ 
  assume P:  $s \in P$ 
  from validF ctxt P
  show  $\Gamma \vdash c \downarrow \text{Normal } s$ 
    by (rule cvalidt-termD)
  qed

```

```

lemma Merge-PostConj:
  assumes validF:  $\Gamma, \Theta \vdash_{t/F} P \text{ } c \text{ } Q, A$ 
  assumes validG:  $\Gamma, \Theta \vdash_{t/G} P' \text{ } c \text{ } R, X$ 
  assumes F-G:  $F \subseteq G$ 
  assumes P-P':  $P \subseteq P'$ 
  shows  $\Gamma, \Theta \vdash_{t/F} P \text{ } c \text{ } (Q \cap R), (A \cap X)$ 
apply (rule hoaret-complete')
apply (rule Merge-PostConj-sound [OF - - F-G P-P'])
using validF apply (blast intro:hoaret-sound)
using validG apply (blast intro:hoaret-sound)
done

```

### 10.3.4 Guards and Guarantees

```

lemma SplitGuards-sound:
  assumes valid-c1:  $\Gamma, \Theta \models_{t/F} P \text{ } c_1 \text{ } Q, A$ 
  assumes valid-c2:  $\Gamma, \Theta \models_{t/F} P \text{ } c_2 \text{ } UNIV, UNIV$ 

```

```

assumes  $c: (c_1 \cap_g c_2) = \text{Some } c$ 
shows  $\Gamma, \Theta \models_{t/F} P \ c \ Q, A$ 
proof (rule cvalidtI)
  fix  $s \ t$ 
  assume  $ctxt: \forall (P, p, Q, A) \in \Theta. \Gamma \models_{t/F} P \ (\text{Call } p) \ Q, A$ 
  hence  $ctxt': \forall (P, p, Q, A) \in \Theta. \Gamma \models_{t/F} P \ (\text{Call } p) \ Q, A$ 
    by (auto simp add: validt-def)
  assume  $exec: \Gamma \vdash \langle c, \text{Normal } s \rangle \Rightarrow t$ 
  assume  $P: s \in P$ 
  assume  $t\text{-notin-}F: t \notin \text{Fault } ' F$ 
  show  $t \in \text{Normal } ' Q \cup \text{Abrupt } ' A$ 
  proof (cases t)
    case Normal
      with inter-guards-exec-noFault [OF c exec]
      have  $\Gamma \vdash \langle c_1, \text{Normal } s \rangle \Rightarrow t$  by simp
      from valid-c1 ctxt this P t-notin-F
      show ?thesis
      by (rule cvalidt-postD)
    next
      case Abrupt
        with inter-guards-exec-noFault [OF c exec]
        have  $\Gamma \vdash \langle c_1, \text{Normal } s \rangle \Rightarrow t$  by simp
        from valid-c1 ctxt this P t-notin-F
        show ?thesis
        by (rule cvalidt-postD)
    next
      case (Fault f)
        assume  $t: t = \text{Fault } f$ 
        with exec inter-guards-exec-Fault [OF c]
        have  $\Gamma \vdash \langle c_1, \text{Normal } s \rangle \Rightarrow \text{Fault } f \vee \Gamma \vdash \langle c_2, \text{Normal } s \rangle \Rightarrow \text{Fault } f$ 
          by auto
        then show ?thesis
        proof (cases rule: disjE [consumes 1])
          assume  $\Gamma \vdash \langle c_1, \text{Normal } s \rangle \Rightarrow \text{Fault } f$ 
          from cvalidt-postD [OF valid-c1 ctxt this P] t t-notin-F
          show ?thesis
          by blast
        next
          assume  $\Gamma \vdash \langle c_2, \text{Normal } s \rangle \Rightarrow \text{Fault } f$ 
          from cvalidD [OF valid-c2 ctxt' this P] t t-notin-F
          show ?thesis
          by blast
        qed
    next
      case Stuck
        with inter-guards-exec-noFault [OF c exec]
        have  $\Gamma \vdash \langle c_1, \text{Normal } s \rangle \Rightarrow t$  by simp
        from valid-c1 ctxt this P t-notin-F
        show ?thesis

```

```

    by (rule cvalidt-postD)
  qed
next
fix s
assume ctxt:  $\forall (P, p, Q, A) \in \Theta. \Gamma \models_{t/F} P (Call\ p)\ Q, A$ 
assume P:  $s \in P$ 
show  $\Gamma \vdash c \downarrow Normal\ s$ 
proof -
  from valid-c1 ctxt P
  have  $\Gamma \vdash_{c_1} \downarrow Normal\ s$ 
  by (rule cvalidt-termD)
  with c show ?thesis
  by (rule inter-guards-terminates)
qed
qed

```

```

lemma SplitGuards:
  assumes c:  $(c_1 \cap_g c_2) = Some\ c$ 
  assumes deriv-c1:  $\Gamma, \Theta \models_{t/F} P\ c_1\ Q, A$ 
  assumes deriv-c2:  $\Gamma, \Theta \models_{t/F} P\ c_2\ UNIV, UNIV$ 
  shows  $\Gamma, \Theta \models_{t/F} P\ c\ Q, A$ 
apply (rule hoaret-complete')
apply (rule SplitGuards-sound [OF - - c])
apply (rule hoaret-sound [OF deriv-c1])
apply (rule hoare-sound [OF deriv-c2])
done

```

```

lemma CombineStrip-sound:
  assumes valid:  $\Gamma, \Theta \models_{t/F} P\ c\ Q, A$ 
  assumes valid-strip:  $\Gamma, \Theta \models_{t/\{\}} P (strip-guards\ (-F)\ c)\ UNIV, UNIV$ 
  shows  $\Gamma, \Theta \models_{t/\{\}} P\ c\ Q, A$ 
proof (rule cvalidtI)
  fix s t
  assume ctxt:  $\forall (P, p, Q, A) \in \Theta. \Gamma \models_{t/\{\}} P (Call\ p)\ Q, A$ 
  hence ctxt':  $\forall (P, p, Q, A) \in \Theta. \Gamma \models_{t/\{\}} P (Call\ p)\ Q, A$ 
  by (auto simp add: validt-def)
  from ctxt have ctxt'':  $\forall (P, p, Q, A) \in \Theta. \Gamma \models_{t/F} P (Call\ p)\ Q, A$ 
  by (auto intro: valid-augment-Faults simp add: validt-def)
  assume exec:  $\Gamma \vdash \langle c, Normal\ s \rangle \Rightarrow t$ 
  assume P:  $s \in P$ 
  assume t-noFault:  $t \notin Fault\ \{\}$ 
  show  $t \in Normal\ \{ Q \cup Abrupt\ \} A$ 
  proof (cases t)
    case (Normal t')
    from cvalidt-postD [OF valid ctxt'' exec P] Normal
    show ?thesis
    by auto
  next

```

```

case (Abrupt  $t'$ )
from cvalidt-postD [OF valid ctxt'' exec P] Abrupt
show ?thesis
  by auto
next
case (Fault  $f$ )
show ?thesis
proof (cases  $f \in F$ )
  case True
  hence  $f \notin -F$  by simp
  with exec Fault
  have  $\Gamma \vdash \langle \text{strip-guards } (-F) \ c, \text{Normal } s \rangle \Rightarrow \text{Fault } f$ 
    by (auto intro: exec-to-exec-strip-guards-Fault)
  from cvalidD [OF valid-strip ctxt' this P] Fault
  have False
    by auto
  thus ?thesis ..
next
case False
with cvalidt-postD [OF valid ctxt'' exec P] Fault
show ?thesis
  by auto
qed
next
case Stuck
from cvalidt-postD [OF valid ctxt'' exec P] Stuck
show ?thesis
  by auto
qed
next
fix  $s$ 
assume  $\text{ctxt}: \forall (P, p, Q, A) \in \Theta. \Gamma \Vdash_{t/\{\}} P (\text{Call } p) \ Q, A$ 
hence  $\text{ctxt}' : \forall (P, p, Q, A) \in \Theta. \Gamma \Vdash_{t/F} P (\text{Call } p) \ Q, A$ 
  by (auto intro: valid-augment-Faults simp add: validt-def)
assume  $P: s \in P$ 
show  $\Gamma \vdash c \downarrow \text{Normal } s$ 
proof –
  from valid ctxt' P
  show  $\Gamma \vdash c \downarrow \text{Normal } s$ 
    by (rule cvalidt-termD)
qed
qed

lemma CombineStrip:
assumes  $\text{deriv}: \Gamma, \Theta \vdash_{t/F} P \ c \ Q, A$ 
assumes  $\text{deriv-strip}: \Gamma, \Theta \vdash_{/\{\}} P (\text{strip-guards } (-F) \ c) \ \text{UNIV}, \text{UNIV}$ 
shows  $\Gamma, \Theta \vdash_{t/\{\}} P \ c \ Q, A$ 
apply (rule hoaret-complete')

```

**apply** (*rule CombineStrip-sound*)  
**apply** (*iprover intro: hoaret-sound [OF deriv]*)  
**apply** (*iprover intro: hoare-sound [OF deriv-strip]*)  
**done**

**lemma** *GuardsFlip-sound:*

**assumes** *valid*:  $\Gamma, \Theta \models_{t/F} P \ c \ Q, A$

**assumes** *validFlip*:  $\Gamma, \Theta \models_{/_F} P \ c \ UNIV, UNIV$

**shows**  $\Gamma, \Theta \models_{t/\{\}} P \ c \ Q, A$

**proof** (*rule cvalidtI*)

**fix**  $s \ t$

**assume** *ctxt*:  $\forall (P, p, Q, A) \in \Theta. \Gamma \models_{t/\{\}} P \ (Call \ p) \ Q, A$

**from** *ctxt* **have** *ctxt'*:  $\forall (P, p, Q, A) \in \Theta. \Gamma \models_{t/F} P \ (Call \ p) \ Q, A$

**by** (*auto intro: valid-augment-Faults simp add: validt-def*)

**from** *ctxt* **have** *ctxtFlip*:  $\forall (P, p, Q, A) \in \Theta. \Gamma \models_{/_F} P \ (Call \ p) \ Q, A$

**by** (*auto intro: valid-augment-Faults simp add: validt-def*)

**assume** *exec*:  $\Gamma \vdash \langle c, Normal \ s \rangle \Rightarrow t$

**assume** *P*:  $s \in P$

**assume** *t-noFault*:  $t \notin Fault \ ' \ \{\}$

**show**  $t \in Normal \ ' \ Q \cup \ A \ Abrupt \ ' \ A$

**proof** (*cases t*)

**case** (*Normal t'*)

**from** *cvalidt-postD [OF valid ctxt' exec P] Normal*

**show** *?thesis*

**by** *auto*

**next**

**case** (*Abrupt t'*)

**from** *cvalidt-postD [OF valid ctxt' exec P] Abrupt*

**show** *?thesis*

**by** *auto*

**next**

**case** (*Fault f*)

**show** *?thesis*

**proof** (*cases f ∈ F*)

**case** *True*

**hence**  $f \notin \_F$  **by** *simp*

**with** *cvalidD [OF validFlip ctxtFlip exec P] Fault*

**have** *False*

**by** *auto*

**thus** *?thesis ..*

**next**

**case** *False*

**with** *cvalidt-postD [OF valid ctxt' exec P] Fault*

**show** *?thesis*

**by** *auto*

**qed**

**next**

**case** *Stuck*

```

    from cvalidt-postD [OF valid ctxt' exec P] Stuck
    show ?thesis
      by auto
  qed
next
  fix s
  assume ctxt:  $\forall (P, p, Q, A) \in \Theta. \Gamma \models_{t/\{\}} P (Call\ p)\ Q, A$ 
  hence ctxt':  $\forall (P, p, Q, A) \in \Theta. \Gamma \models_{t/F} P (Call\ p)\ Q, A$ 
    by (auto intro: valid-augment-Faults simp add: validt-def)
  assume P:  $s \in P$ 
  show  $\Gamma \vdash c \downarrow Normal\ s$ 
  proof -
    from valid ctxt' P
    show  $\Gamma \vdash c \downarrow Normal\ s$ 
      by (rule cvalidt-termD)
  qed
qed

```

**lemma** *GuardsFlip*:

```

  assumes deriv:  $\Gamma, \Theta \vdash_{t/F} P\ c\ Q, A$ 
  assumes derivFlip:  $\Gamma, \Theta \vdash_{-F} P\ c\ UNIV, UNIV$ 
  shows  $\Gamma, \Theta \vdash_{t/\{\}} P\ c\ Q, A$ 
  apply (rule hoaret-complete')
  apply (rule GuardsFlip-sound)
  apply (iprover intro: hoaret-sound [OF deriv])
  apply (iprover intro: hoare-sound [OF derivFlip])
  done

```

**lemma** *MarkGuardsI-sound*:

```

  assumes valid:  $\Gamma, \Theta \models_{t/\{\}} P\ c\ Q, A$ 
  shows  $\Gamma, \Theta \models_{t/\{\}} P\ mark-guards\ f\ c\ Q, A$ 
  proof (rule cvalidtI)
    fix s t
    assume ctxt:  $\forall (P, p, Q, A) \in \Theta. \Gamma \models_{t/\{\}} P (Call\ p)\ Q, A$ 
    assume exec:  $\Gamma \vdash \langle mark-guards\ f\ c, Normal\ s \rangle \Rightarrow t$ 
    from exec-mark-guards-to-exec [OF exec] obtain t' where
      exec-c:  $\Gamma \vdash \langle c, Normal\ s \rangle \Rightarrow t'$  and
      t'-noFault:  $\neg isFault\ t' \longrightarrow t' = t$ 
    by blast
    assume P:  $s \in P$ 
    assume t-noFault:  $t \notin Fault\ ' \{\}$ 
    show  $t \in Normal\ ' Q \cup Abrupt\ ' A$ 
    proof -
      from cvalidt-postD [OF valid [rule-format] ctxt exec-c P]
      have t'  $\in Normal\ ' Q \cup Abrupt\ ' A$ 
        by blast
      with t'-noFault

```



```

    show ?thesis
      by auto
  qed
next
  fix s
  assume ctxt:  $\forall (P, p, Q, A) \in \Theta. \Gamma \models_t / \{\} P (Call\ p)\ Q, A$ 
  assume P:  $s \in P$ 
  from cvalidt-termD [OF valid ctxt P]
  have  $\Gamma \vdash c \downarrow Normal\ s.$ 
  thus  $\Gamma \vdash mark-guards\ f\ c \downarrow Normal\ s$ 
    by (rule terminates-to-terminates-mark-guards)
  qed

lemma MarkGuardsI:
  assumes deriv:  $\Gamma, \Theta \vdash_t / \{\} P\ c\ Q, A$ 
  shows  $\Gamma, \Theta \vdash_t / \{\} P\ mark-guards\ f\ c\ Q, A$ 
  apply (rule hoaret-complete')
  apply (rule MarkGuardsI-sound)
  apply (iprover intro: hoaret-sound [OF deriv])
  done

lemma MarkGuardsD-sound:
  assumes valid:  $\Gamma, \Theta \models_t / \{\} P\ mark-guards\ f\ c\ Q, A$ 
  shows  $\Gamma, \Theta \vdash_t / \{\} P\ c\ Q, A$ 
  proof (rule cvalidtI)
    fix s t
    assume ctxt:  $\forall (P, p, Q, A) \in \Theta. \Gamma \models_t / \{\} P (Call\ p)\ Q, A$ 
    assume exec:  $\Gamma \vdash \langle c, Normal\ s \rangle \Rightarrow t$ 
    assume P:  $s \in P$ 
    assume t-noFault:  $t \notin Fault\ ' \{\}$ 
    show  $t \in Normal\ ' Q \cup Abrupt\ ' A$ 
    proof (cases isFault t)
      case True
        with exec-to-exec-mark-guards-Fault exec
        obtain f' where  $\Gamma \vdash \langle mark-guards\ f\ c, Normal\ s \rangle \Rightarrow Fault\ f'$ 
          by (fastforce elim: isFaultE)
        from cvalidt-postD [OF valid [rule-format] ctxt this P]
        have False
          by auto
        thus ?thesis ..
      case False
        from exec-to-exec-mark-guards [OF exec False]
        obtain f' where  $\Gamma \vdash \langle mark-guards\ f\ c, Normal\ s \rangle \Rightarrow t$ 
          by auto
        from cvalidt-postD [OF valid [rule-format] ctxt this P]
        show ?thesis
  
```

by *auto*  
 qed  
 next  
 fix  $s$   
 assume  $ctxt: \forall (P, p, Q, A) \in \Theta. \Gamma \models_{t/\{\}} P (Call\ p) Q, A$   
 assume  $P: s \in P$   
 from *cvalidt-termD* [OF *valid ctxt P*]  
 have  $\Gamma \vdash \text{mark-guards } f\ c \downarrow \text{Normal } s.$   
 thus  $\Gamma \vdash c \downarrow \text{Normal } s$   
 by (*rule terminates-mark-guards-to-terminates*)  
 qed

**lemma** *MarkGuardsD*:  
 assumes  $deriv: \Gamma, \Theta \vdash_{t/\{\}} P \text{ mark-guards } f\ c\ Q, A$   
 shows  $\Gamma, \Theta \vdash_{t/\{\}} P\ c\ Q, A$   
 apply (*rule hoaret-complete'*)  
 apply (*rule MarkGuardsD-sound*)  
 apply (*iprover intro: hoaret-sound* [OF *deriv*])  
 done

**lemma** *MergeGuardsI-sound*:  
 assumes  $valid: \Gamma, \Theta \models_{t/F} P\ c\ Q, A$   
 shows  $\Gamma, \Theta \models_{t/F} P \text{ merge-guards } c\ Q, A$   
**proof** (*rule cvalidtI*)  
 fix  $s\ t$   
 assume  $ctxt: \forall (P, p, Q, A) \in \Theta. \Gamma \models_{t/F} P (Call\ p) Q, A$   
 assume  $exec\text{-merge}: \Gamma \langle \text{merge-guards } c, \text{Normal } s \rangle \Rightarrow t$   
 from *exec-merge-guards-to-exec* [OF *exec-merge*]  
 have  $exec: \Gamma \langle c, \text{Normal } s \rangle \Rightarrow t.$   
 assume  $P: s \in P$   
 assume  $t\text{-notin-}F: t \notin \text{Fault } 'F$   
 from *cvalidt-postD* [OF *valid* [rule-format] *ctxt exec P t-notin-F*]  
 show  $t \in \text{Normal } 'Q \cup \text{Abrupt } 'A.$

next  
 fix  $s$   
 assume  $ctxt: \forall (P, p, Q, A) \in \Theta. \Gamma \models_{t/F} P (Call\ p) Q, A$   
 assume  $P: s \in P$   
 from *cvalidt-termD* [OF *valid ctxt P*]  
 have  $\Gamma \vdash c \downarrow \text{Normal } s.$   
 thus  $\Gamma \vdash \text{merge-guards } c \downarrow \text{Normal } s$   
 by (*rule terminates-to-terminates-merge-guards*)  
 qed

**lemma** *MergeGuardsI*:  
 assumes  $deriv: \Gamma, \Theta \vdash_{t/F} P\ c\ Q, A$   
 shows  $\Gamma, \Theta \vdash_{t/F} P \text{ merge-guards } c\ Q, A$   
 apply (*rule hoaret-complete'*)  
 apply (*rule MergeGuardsI-sound*)

**apply** (*iprover intro: hoaret-sound [OF deriv]*)  
**done**

**lemma** *MergeGuardsD-sound*:

**assumes** *valid*:  $\Gamma, \Theta \models_{t/F} P$  *merge-guards*  $c$   $Q, A$

**shows**  $\Gamma, \Theta \models_{t/F} P$   $c$   $Q, A$

**proof** (*rule cvalidtI*)

**fix**  $s$   $t$

**assume** *ctxt*:  $\forall (P, p, Q, A) \in \Theta. \Gamma \models_{t/F} P$  (*Call*  $p$ )  $Q, A$

**assume** *exec*:  $\Gamma \vdash \langle c, \text{Normal } s \rangle \Rightarrow t$

**from** *exec-to-exec-merge-guards* [*OF exec*]

**have** *exec-merge*:  $\Gamma \vdash \langle \text{merge-guards } c, \text{Normal } s \rangle \Rightarrow t$ .

**assume**  $P: s \in P$

**assume** *t-notin-F*:  $t \notin \text{Fault } F$

**from** *cvalidt-postD* [*OF valid [rule-format] ctxt exec-merge P t-notin-F*]

**show**  $t \in \text{Normal } Q \cup \text{Abrupt } A$ .

**next**

**fix**  $s$

**assume** *ctxt*:  $\forall (P, p, Q, A) \in \Theta. \Gamma \models_{t/F} P$  (*Call*  $p$ )  $Q, A$

**assume**  $P: s \in P$

**from** *cvalidt-termD* [*OF valid ctxt P*]

**have**  $\Gamma \vdash \text{merge-guards } c \downarrow \text{Normal } s$ .

**thus**  $\Gamma \vdash c \downarrow \text{Normal } s$

**by** (*rule terminates-merge-guards-to-terminates*)

**qed**

**lemma** *MergeGuardsD*:

**assumes** *deriv*:  $\Gamma, \Theta \vdash_{t/F} P$  *merge-guards*  $c$   $Q, A$

**shows**  $\Gamma, \Theta \vdash_{t/F} P$   $c$   $Q, A$

**apply** (*rule hoaret-complete'*)

**apply** (*rule MergeGuardsD-sound*)

**apply** (*iprover intro: hoaret-sound [OF deriv]*)

**done**

**lemma** *SubsetGuards-sound*:

**assumes**  $c-c'$ :  $c \subseteq_g c'$

**assumes** *valid*:  $\Gamma, \Theta \models_{t/\{\}} P$   $c'$   $Q, A$

**shows**  $\Gamma, \Theta \models_{t/\{\}} P$   $c$   $Q, A$

**proof** (*rule cvalidtI*)

**fix**  $s$   $t$

**assume** *ctxt*:  $\forall (P, p, Q, A) \in \Theta. \Gamma \models_{t/\{\}} P$  (*Call*  $p$ )  $Q, A$

**assume** *exec*:  $\Gamma \vdash \langle c, \text{Normal } s \rangle \Rightarrow t$

**from** *exec-to-exec-subseteq-guards* [*OF c-c' exec*] **obtain**  $t'$  **where**

*exec-c'*:  $\Gamma \vdash \langle c', \text{Normal } s \rangle \Rightarrow t'$  **and**

*t'-noFault*:  $\neg \text{isFault } t' \longrightarrow t' = t$

**by** *blast*

**assume**  $P: s \in P$

**assume**  $t\text{-noFault}$ :  $t \notin \text{Fault} \text{ ' } \{\}$   
**from**  $\text{cvalidt-postD}$  [ $OF$  valid [rule-format]  $\text{ctxt exec-c}' P$ ]  $t'\text{-noFault } t\text{-noFault}$   
**show**  $t \in \text{Normal} \text{ ' } Q \cup \text{Abrupt} \text{ ' } A$   
    **by** *auto*  
**next**  
**fix**  $s$   
**assume**  $\text{ctxt}$ :  $\forall (P, p, Q, A) \in \Theta. \Gamma \models_{t/\{\}} P (Call\ p) Q, A$   
**assume**  $P$ :  $s \in P$   
**from**  $\text{cvalidt-termD}$  [ $OF$  valid  $\text{ctxt } P$ ]  
**have**  $\text{termi-c}'$ :  $\Gamma \vdash c' \downarrow \text{Normal } s$ .  
**from**  $\text{cvalidt-postD}$  [ $OF$  valid  $\text{ctxt} - P$ ]  
**have**  $\text{noFault-c}'$ :  $\Gamma \vdash \langle c', \text{Normal } s \rangle \Rightarrow \notin \text{Fault} \text{ ' } UNIV$   
    **by** (*auto simp add: final-notin-def*)  
**from**  $\text{termi-c}'$   $c\text{-c}'$   $\text{noFault-c}'$   
**show**  $\Gamma \vdash c \downarrow \text{Normal } s$   
    **by** (*rule terminates-fewer-guards*)  
**qed**

**lemma** *SubsetGuards*:  
**assumes**  $c\text{-c}'$ :  $c \subseteq_g c'$   
**assumes**  $\text{deriv}$ :  $\Gamma, \Theta \vdash_{t/\{\}} P c' Q, A$   
**shows**  $\Gamma, \Theta \vdash_{t/\{\}} P c Q, A$   
**apply** (*rule hoaret-complete'*)  
**apply** (*rule SubsetGuards-sound* [ $OF$   $c\text{-c}'$ ])  
**apply** (*iprover intro: hoaret-sound* [ $OF$   $\text{deriv}$ ])  
**done**

**lemma** *NormalizeD-sound*:  
**assumes**  $\text{valid}$ :  $\Gamma, \Theta \models_{t/F} P (\text{normalize } c) Q, A$   
**shows**  $\Gamma, \Theta \models_{t/F} P c Q, A$

**proof** (*rule cvalidtI*)  
**fix**  $s\ t$   
**assume**  $\text{ctxt}$ :  $\forall (P, p, Q, A) \in \Theta. \Gamma \models_{t/F} P (Call\ p) Q, A$   
**assume**  $\text{exec}$ :  $\Gamma \vdash \langle c, \text{Normal } s \rangle \Rightarrow t$   
**hence**  $\text{exec-norm}$ :  $\Gamma \vdash \langle \text{normalize } c, \text{Normal } s \rangle \Rightarrow t$   
    **by** (*rule exec-to-exec-normalize*)  
**assume**  $P$ :  $s \in P$   
**assume**  $\text{noFault}$ :  $t \notin \text{Fault} \text{ ' } F$   
**from**  $\text{cvalidt-postD}$  [ $OF$  valid [rule-format]  $\text{ctxt exec-norm } P \text{noFault}$ ]  
**show**  $t \in \text{Normal} \text{ ' } Q \cup \text{Abrupt} \text{ ' } A$ .  
**next**  
**fix**  $s$   
**assume**  $\text{ctxt}$ :  $\forall (P, p, Q, A) \in \Theta. \Gamma \models_{t/F} P (Call\ p) Q, A$   
**assume**  $P$ :  $s \in P$   
**from**  $\text{cvalidt-termD}$  [ $OF$  valid  $\text{ctxt } P$ ]  
**have**  $\Gamma \vdash \text{normalize } c \downarrow \text{Normal } s$ .  
**thus**  $\Gamma \vdash c \downarrow \text{Normal } s$   
    **by** (*rule terminates-normalize-to-terminates*)

qed

**lemma** *NormalizeD*:

assumes *deriv*:  $\Gamma, \Theta \vdash_{t/F} P \text{ (normalize } c) Q, A$   
shows  $\Gamma, \Theta \vdash_{t/F} P c Q, A$   
apply (rule *hoaret-complete'*)  
apply (rule *NormalizeD-sound*)  
apply (*iprover intro*: *hoaret-sound* [*OF deriv*])  
done

**lemma** *NormalizeI-sound*:

assumes *valid*:  $\Gamma, \Theta \models_{t/F} P c Q, A$   
shows  $\Gamma, \Theta \vdash_{t/F} P \text{ (normalize } c) Q, A$   
**proof** (rule *cvalidtI*)  
fix *s t*  
assume *ctxt*:  $\forall (P, p, Q, A) \in \Theta. \Gamma \models_{t/F} P \text{ (Call } p) Q, A$   
assume  $\Gamma \langle \text{normalize } c, \text{Normal } s \rangle \Rightarrow t$   
hence *exec*:  $\Gamma \langle c, \text{Normal } s \rangle \Rightarrow t$   
by (rule *exec-normalize-to-exec*)  
assume *P*:  $s \in P$   
assume *noFault*:  $t \notin \text{Fault } ' F$   
from *cvalidt-postD* [*OF valid* [*rule-format*] *ctxt exec P noFault*]  
show  $t \in \text{Normal } ' Q \cup \text{Abrupt } ' A$ .  
**next**  
fix *s*  
assume *ctxt*:  $\forall (P, p, Q, A) \in \Theta. \Gamma \models_{t/F} P \text{ (Call } p) Q, A$   
assume *P*:  $s \in P$   
from *cvalidt-termD* [*OF valid ctxt P*]  
have  $\Gamma \vdash c \downarrow \text{Normal } s$ .  
thus  $\Gamma \vdash \text{normalize } c \downarrow \text{Normal } s$   
by (rule *terminates-to-terminates-normalize*)  
qed

**lemma** *NormalizeI*:

assumes *deriv*:  $\Gamma, \Theta \vdash_{t/F} P c Q, A$   
shows  $\Gamma, \Theta \vdash_{t/F} P \text{ (normalize } c) Q, A$   
apply (rule *hoaret-complete'*)  
apply (rule *NormalizeI-sound*)  
apply (*iprover intro*: *hoaret-sound* [*OF deriv*])  
done

### 10.3.5 Restricting the Procedure Environment

**lemma** *validt-restrict-to-validt*:

assumes *validt-c*:  $\Gamma \mid_M \models_{t/F} P c Q, A$   
shows  $\Gamma \models_{t/F} P c Q, A$   
**proof** –  
from *validt-c*

```

have valid-c:  $\Gamma \mid_M \Vdash_{/F} P \ c \ Q, A$  by (simp add: validt-def)
hence  $\Gamma \Vdash_{/F} P \ c \ Q, A$  by (rule valid-restrict-to-valid)
moreover
{
  fix s
  assume P:  $s \in P$ 
  have  $\Gamma \vdash c \downarrow \text{Normal } s$ 
  proof –
    from P validt-c have  $\Gamma \mid_M \vdash c \downarrow \text{Normal } s$ 
      by (auto simp add: validt-def)
    moreover
    from P valid-c
    have  $\Gamma \mid_M \vdash (c, \text{Normal } s) \Rightarrow \notin \{\text{Stuck}\}$ 
      by (auto simp add: valid-def final-notin-def)
    ultimately show ?thesis
      by (rule terminates-restrict-to-terminates)
  qed
}
ultimately show ?thesis
by (auto simp add: validt-def)
qed

```

```

lemma augment-procs:
assumes deriv-c:  $\Gamma \mid_M, \{\} \vdash_{t/F} P \ c \ Q, A$ 
shows  $\Gamma, \{\} \vdash_{t/F} P \ c \ Q, A$ 
  apply (rule hoaret-complete)
  apply (rule validt-restrict-to-validt)
  apply (insert hoaret-sound [OF deriv-c])
  by (simp add: cvalidt-def)

```

### 10.3.6 Miscellaneous

```

lemma augment-Faults:
assumes deriv-c:  $\Gamma, \{\} \vdash_{t/F} P \ c \ Q, A$ 
assumes F:  $F \subseteq F'$ 
shows  $\Gamma, \{\} \vdash_{t/F'} P \ c \ Q, A$ 
  apply (rule hoaret-complete)
  apply (rule validt-augment-Faults [OF - F])
  apply (insert hoaret-sound [OF deriv-c])
  by (simp add: cvalidt-def)

```

```

lemma TerminationPartial-sound:
assumes termination:  $\forall s \in P. \Gamma \vdash c \downarrow \text{Normal } s$ 
assumes partial-corr:  $\Gamma, \Theta \Vdash_{/F} P \ c \ Q, A$ 
shows  $\Gamma, \Theta \Vdash_{t/F} P \ c \ Q, A$ 
using termination partial-corr
by (auto simp add: cvalidt-def validt-def cvalid-def)

```

**lemma** *TerminationPartial*:  
**assumes** *partial-deriv*:  $\Gamma, \Theta \vdash_{/F} P \text{ c } Q, A$   
**assumes** *termination*:  $\forall s \in P. \Gamma \vdash c \downarrow \text{Normal } s$   
**shows**  $\Gamma, \Theta \vdash_{/F} P \text{ c } Q, A$   
**apply** (rule *hoaret-complete'*)  
**apply** (rule *TerminationPartial-sound* [*OF termination*])  
**apply** (rule *hoare-sound* [*OF partial-deriv*])  
**done**

**lemma** *TerminationPartialStrip*:  
**assumes** *partial-deriv*:  $\Gamma, \Theta \vdash_{/F} P \text{ c } Q, A$   
**assumes** *termination*:  $\forall s \in P. \text{strip } F' \Gamma \vdash \text{strip-guards } F' \text{ c } \downarrow \text{Normal } s$   
**shows**  $\Gamma, \Theta \vdash_{/F} P \text{ c } Q, A$   
**proof** –  
**from** *termination* **have**  $\forall s \in P. \Gamma \vdash c \downarrow \text{Normal } s$   
**by** (*auto intro: terminates-strip-guards-to-terminates*  
*terminates-strip-to-terminates*)  
**with** *partial-deriv*  
**show** *?thesis*  
**by** (rule *TerminationPartial*)  
**qed**

**lemma** *SplitTotalPartial*:  
**assumes** *termi*:  $\Gamma, \Theta \vdash_{/F} P \text{ c } Q', A'$   
**assumes** *part*:  $\Gamma, \Theta \vdash_{/F} P \text{ c } Q, A$   
**shows**  $\Gamma, \Theta \vdash_{/F} P \text{ c } Q, A$   
**proof** –  
**from** *hoaret-sound* [*OF termi*] *hoare-sound* [*OF part*]  
**have**  $\Gamma, \Theta \vdash_{/F} P \text{ c } Q, A$   
**by** (*fastforce simp add: cvalidt-def validt-def cvalid-def valid-def*)  
**thus** *?thesis*  
**by** (rule *hoaret-complete'*)  
**qed**

**lemma** *SplitTotalPartial'*:  
**assumes** *termi*:  $\Gamma, \Theta \vdash_{/UNIV} P \text{ c } Q', A'$   
**assumes** *part*:  $\Gamma, \Theta \vdash_{/F} P \text{ c } Q, A$   
**shows**  $\Gamma, \Theta \vdash_{/F} P \text{ c } Q, A$   
**proof** –  
**from** *hoaret-sound* [*OF termi*] *hoare-sound* [*OF part*]  
**have**  $\Gamma, \Theta \vdash_{/F} P \text{ c } Q, A$   
**by** (*fastforce simp add: cvalidt-def validt-def cvalid-def valid-def*)  
**thus** *?thesis*  
**by** (rule *hoaret-complete'*)  
**qed**

**end**

## 11 Derived Hoare Rules for Total Correctness

**theory** *HoareTotal* **imports** *HoareTotalProps* **begin**

**lemma** *conseq-no-aux*:

$$\llbracket \Gamma, \Theta \vdash_{t/F} P' \ c \ Q', A';$$

$$\forall s. s \in P \longrightarrow (s \in P' \wedge (Q' \subseteq Q) \wedge (A' \subseteq A)) \rrbracket$$

$$\implies$$

$$\Gamma, \Theta \vdash_{t/F} P \ c \ Q, A$$

**by** (*rule conseq* [**where**  $P' = \lambda Z. P'$  **and**  $Q' = \lambda Z. Q'$  **and**  $A' = \lambda Z. A'$ ]) *auto*

If for example a specification for a "procedure pointer" parameter is in the precondition we can extract it with this rule

**lemma** *conseq-exploit-pre*:

$$\llbracket \forall s \in P. \Gamma, \Theta \vdash_{t/F} (\{s\} \cap P) \ c \ Q, A \rrbracket$$

$$\implies$$

$$\Gamma, \Theta \vdash_{t/F} P \ c \ Q, A$$

**apply** (*rule Conseq*)

**apply** *clarify*

**apply** (*rule-tac*  $x = \{s\} \cap P$  **in** *exI*)

**apply** (*rule-tac*  $x = Q$  **in** *exI*)

**apply** (*rule-tac*  $x = A$  **in** *exI*)

**by** *simp*

**lemma** *conseq*:  $\llbracket \forall Z. \Gamma, \Theta \vdash_{t/F} (P' \ Z) \ c \ (Q' \ Z), (A' \ Z);$

$$\forall s. s \in P \longrightarrow (\exists Z. s \in P' \ Z \wedge (Q' \ Z \subseteq Q) \wedge (A' \ Z \subseteq A)) \rrbracket$$

$$\implies$$

$$\Gamma, \Theta \vdash_{t/F} P \ c \ Q, A$$

**by** (*rule Conseq'*) *blast*

**lemma** *Lem*:  $\llbracket \forall Z. \Gamma, \Theta \vdash_{t/F} (P' \ Z) \ c \ (Q' \ Z), (A' \ Z);$

$$P \subseteq \{s. \exists Z. s \in P' \ Z \wedge (Q' \ Z \subseteq Q) \wedge (A' \ Z \subseteq A)\} \rrbracket$$

$$\implies$$

$$\Gamma, \Theta \vdash_{t/F} P \ (lem \ x \ c) \ Q, A$$

**apply** (*unfold lem-def*)

**apply** (*erule conseq*)

**apply** *blast*

**done**

**lemma** *LemAnno*:

**assumes** *conseq*:  $P \subseteq \{s. \exists Z. s \in P' \ Z \wedge$

$$(\forall t. t \in Q' \ Z \longrightarrow t \in Q) \wedge (\forall t. t \in A' \ Z \longrightarrow t \in A)\}$$

**assumes** *lem*:  $\forall Z. \Gamma, \Theta \vdash_{t/F} (P' \ Z) \ c \ (Q' \ Z), (A' \ Z)$

**shows**  $\Gamma, \Theta \vdash_{t/F} P \ (lem \ x \ c) \ Q, A$

**apply** (*rule Lem* [*OF lem*])



**using** *conseq*  
**by** *blast*

**lemma** *LemAnnoNoAbrupt*:

**assumes** *conseq*:  $P \subseteq \{s. \exists Z. s \in P' Z \wedge (\forall t. t \in Q' Z \longrightarrow t \in Q)\}$

**assumes** *lem*:  $\forall Z. \Gamma, \Theta \vdash_{t/F} (P' Z) c (Q' Z), \{\}$

**shows**  $\Gamma, \Theta \vdash_{t/F} P (lem\ x\ c) Q, \{\}$

**apply** (*rule Lem [OF lem]*)

**using** *conseq*

**by** *blast*

**lemma** *TrivPost*:  $\forall Z. \Gamma, \Theta \vdash_{t/F} (P' Z) c (Q' Z), (A' Z)$

$\implies$

$\forall Z. \Gamma, \Theta \vdash_{t/F} (P' Z) c UNIV, UNIV$

**apply** (*rule allI*)

**apply** (*erule conseq*)

**apply** *auto*

**done**

**lemma** *TrivPostNoAbr*:  $\forall Z. \Gamma, \Theta \vdash_{t/F} (P' Z) c (Q' Z), \{\}$

$\implies$

$\forall Z. \Gamma, \Theta \vdash_{t/F} (P' Z) c UNIV, \{\}$

**apply** (*rule allI*)

**apply** (*erule conseq*)

**apply** *auto*

**done**

**lemma** *DynComConseq*:

**assumes**  $P \subseteq \{s. \exists P' Q' A'. \Gamma, \Theta \vdash_{t/F} P' (c\ s) Q', A' \wedge P \subseteq P' \wedge Q' \subseteq Q \wedge A' \subseteq A\}$

**shows**  $\Gamma, \Theta \vdash_{t/F} P\ DynCom\ c\ Q, A$

**using** *assms*

**apply**  $-$

**apply** (*rule hoaret.DynCom*)

**apply** *clarsimp*

**apply** (*rule hoaret.Conseq*)

**apply** *clarsimp*

**apply** *blast*

**done**

**lemma** *SpecAnno*:

**assumes** *consequence*:  $P \subseteq \{s. (\exists Z. s \in P' Z \wedge (Q' Z \subseteq Q) \wedge (A' Z \subseteq A))\}$

**assumes** *spec*:  $\forall Z. \Gamma, \Theta \vdash_{t/F} (P' Z) (c\ Z) (Q' Z), (A' Z)$

**assumes** *bdy-constant*:  $\forall Z. c\ Z = c\ undefined$

**shows**  $\Gamma, \Theta \vdash_{t/F} P (specAnno\ P' c\ Q' A') Q, A$

**proof**  $-$

**from** *spec bdy-constant*

**have**  $\forall Z. \Gamma, \Theta \vdash_{t/F} (P' Z) (c\ undefined) (Q' Z), (A' Z)$

**apply** –  
**apply** (*rule allI*)  
**apply** (*erule-tac x=Z in allE*)  
**apply** (*erule-tac x=Z in allE*)  
**apply** *simp*  
**done**  
**with** *consequence show ?thesis*  
**apply** (*simp add: specAnno-def*)  
**apply** (*erule conseq*)  
**apply** *blast*  
**done**  
**qed**

**lemma** *SpecAnno'*:

$\llbracket P \subseteq \{s. \exists Z. s \in P' Z \wedge$   
 $(\forall t. t \in Q' Z \longrightarrow t \in Q) \wedge (\forall t. t \in A' Z \longrightarrow t \in A)\};$   
 $\forall Z. \Gamma, \Theta \vdash_{t/F} (P' Z) (c Z) (Q' Z), (A' Z);$   
 $\forall Z. c Z = c \text{ undefined}$   
 $\rrbracket \Longrightarrow$   
 $\Gamma, \Theta \vdash_{t/F} P (\text{specAnno } P' c Q' A') Q, A$   
**apply** (*simp only: subset-iff [THEN sym]*)  
**apply** (*erule (1) SpecAnno*)  
**apply** *assumption*  
**done**

**lemma** *SpecAnnoNoAbrupt*:

$\llbracket P \subseteq \{s. \exists Z. s \in P' Z \wedge$   
 $(\forall t. t \in Q' Z \longrightarrow t \in Q)\};$   
 $\forall Z. \Gamma, \Theta \vdash_{t/F} (P' Z) (c Z) (Q' Z), \{\};$   
 $\forall Z. c Z = c \text{ undefined}$   
 $\rrbracket \Longrightarrow$   
 $\Gamma, \Theta \vdash_{t/F} P (\text{specAnno } P' c Q' (\lambda s. \{\})) Q, A$   
**apply** (*rule SpecAnno'*)  
**apply** *auto*  
**done**

**lemma** *Skip*:  $P \subseteq Q \Longrightarrow \Gamma, \Theta \vdash_{t/F} P \text{ Skip } Q, A$   
**by** (*rule hoaret.Skip [THEN conseqPre], simp*)

**lemma** *Basic*:  $P \subseteq \{s. (f s) \in Q\} \Longrightarrow \Gamma, \Theta \vdash_{t/F} P (\text{Basic } f) Q, A$   
**by** (*rule hoaret.Basic [THEN conseqPre]*)

**lemma** *BasicCond*:

$\llbracket P \subseteq \{s. (b s \longrightarrow f s \in Q) \wedge (\neg b s \longrightarrow g s \in Q)\} \rrbracket \Longrightarrow$   
 $\Gamma, \Theta \vdash_{t/F} P \text{ Basic } (\lambda s. \text{if } b s \text{ then } f s \text{ else } g s) Q, A$   
**apply** (*rule Basic*)

**apply** *auto*  
**done**

**lemma** *Spec*:  $P \subseteq \{s. (\forall t. (s,t) \in r \longrightarrow t \in Q) \wedge (\exists t. (s,t) \in r)\}$   
 $\implies \Gamma, \Theta \vdash_{t/F} P \text{ (Spec } r) \ Q, A$   
**by** (*rule hoaret.Spec [THEN conseqPre]*)

**lemma** *SpecIf*:  
 $\llbracket P \subseteq \{s. (b \ s \longrightarrow f \ s \in Q) \wedge (\neg \ b \ s \longrightarrow g \ s \in Q \wedge h \ s \in Q)\} \rrbracket \implies$   
 $\Gamma, \Theta \vdash_{t/F} P \text{ Spec (if-rel } b \ f \ g \ h) \ Q, A$   
**apply** (*rule Spec*)  
**apply** (*auto simp add: if-rel-def*)  
**done**

**lemma** *Seq* [*trans, intro?*]:  
 $\llbracket \Gamma, \Theta \vdash_{t/F} P \ c_1 \ R, A; \Gamma, \Theta \vdash_{t/F} R \ c_2 \ Q, A \rrbracket \implies \Gamma, \Theta \vdash_{t/F} P \text{ Seq } c_1 \ c_2 \ Q, A$   
**by** (*rule hoaret.Seq*)

**lemma** *SeqSwap*:  
 $\llbracket \Gamma, \Theta \vdash_{t/F} R \ c_2 \ Q, A; \Gamma, \Theta \vdash_{t/F} P \ c_1 \ R, A \rrbracket \implies \Gamma, \Theta \vdash_{t/F} P \text{ Seq } c_1 \ c_2 \ Q, A$   
**by** (*rule Seq*)

**lemma** *BSeq*:  
 $\llbracket \Gamma, \Theta \vdash_{t/F} P \ c_1 \ R, A; \Gamma, \Theta \vdash_{t/F} R \ c_2 \ Q, A \rrbracket \implies \Gamma, \Theta \vdash_{t/F} P \text{ (bseq } c_1 \ c_2) \ Q, A$   
**by** (*unfold bseq-def*) (*rule Seq*)

**lemma** *Cond*:  
**assumes** *wp*:  $P \subseteq \{s. (s \in b \longrightarrow s \in P_1) \wedge (s \notin b \longrightarrow s \in P_2)\}$   
**assumes** *deriv-c1*:  $\Gamma, \Theta \vdash_{t/F} P_1 \ c_1 \ Q, A$   
**assumes** *deriv-c2*:  $\Gamma, \Theta \vdash_{t/F} P_2 \ c_2 \ Q, A$   
**shows**  $\Gamma, \Theta \vdash_{t/F} P \text{ (Cond } b \ c_1 \ c_2) \ Q, A$   
**proof** (*rule hoaret.Cond [THEN conseqPre]*)  
**from** *deriv-c1*  
**show**  $\Gamma, \Theta \vdash_{t/F} (\{s. (s \in b \longrightarrow s \in P_1) \wedge (s \notin b \longrightarrow s \in P_2)\} \cap b) \ c_1 \ Q, A$   
**by** (*rule conseqPre*) *blast*  
**next**  
**from** *deriv-c2*  
**show**  $\Gamma, \Theta \vdash_{t/F} (\{s. (s \in b \longrightarrow s \in P_1) \wedge (s \notin b \longrightarrow s \in P_2)\} \cap -b) \ c_2 \ Q, A$   
**by** (*rule conseqPre*) *blast*  
**qed** (*insert wp*)

**lemma** *CondSwap*:  
 $\llbracket \Gamma, \Theta \vdash_{t/F} P_1 \ c_1 \ Q, A; \Gamma, \Theta \vdash_{t/F} P_2 \ c_2 \ Q, A;$   
 $P \subseteq \{s. (s \in b \longrightarrow s \in P_1) \wedge (s \notin b \longrightarrow s \in P_2)\} \rrbracket$   
 $\implies$   
 $\Gamma, \Theta \vdash_{t/F} P \text{ (Cond } b \ c_1 \ c_2) \ Q, A$

by (rule *Cond*)

**lemma** *Cond'*:

$\llbracket P \subseteq \{s. (b \subseteq P1) \wedge (- b \subseteq P2)\}; \Gamma, \Theta \vdash_{t/F} P1 \ c1 \ Q, A; \Gamma, \Theta \vdash_{t/F} P2 \ c2 \ Q, A \rrbracket$   
 $\implies$   
 $\Gamma, \Theta \vdash_{t/F} P \ (Cond \ b \ c1 \ c2) \ Q, A$   
 by (rule *CondSwap*) *blast+*

**lemma** *CondInv*:

**assumes** *wp*:  $P \subseteq Q$   
**assumes** *inv*:  $Q \subseteq \{s. (s \in b \longrightarrow s \in P_1) \wedge (s \notin b \longrightarrow s \in P_2)\}$   
**assumes** *deriv-c1*:  $\Gamma, \Theta \vdash_{t/F} P_1 \ c_1 \ Q, A$   
**assumes** *deriv-c2*:  $\Gamma, \Theta \vdash_{t/F} P_2 \ c_2 \ Q, A$   
**shows**  $\Gamma, \Theta \vdash_{t/F} P \ (Cond \ b \ c_1 \ c_2) \ Q, A$

**proof** –

**from** *wp inv*  
**have**  $P \subseteq \{s. (s \in b \longrightarrow s \in P_1) \wedge (s \notin b \longrightarrow s \in P_2)\}$   
 by *blast*  
**from** *Cond* [*OF this deriv-c1 deriv-c2*]  
**show** *?thesis* .

**qed**

**lemma** *CondInv'*:

**assumes** *wp*:  $P \subseteq I$   
**assumes** *inv*:  $I \subseteq \{s. (s \in b \longrightarrow s \in P_1) \wedge (s \notin b \longrightarrow s \in P_2)\}$   
**assumes** *wp'*:  $I \subseteq Q$   
**assumes** *deriv-c1*:  $\Gamma, \Theta \vdash_{t/F} P_1 \ c_1 \ I, A$   
**assumes** *deriv-c2*:  $\Gamma, \Theta \vdash_{t/F} P_2 \ c_2 \ I, A$   
**shows**  $\Gamma, \Theta \vdash_{t/F} P \ (Cond \ b \ c_1 \ c_2) \ Q, A$

**proof** –

**from** *CondInv* [*OF wp inv deriv-c1 deriv-c2*]  
**have**  $\Gamma, \Theta \vdash_{t/F} P \ (Cond \ b \ c_1 \ c_2) \ I, A$  .  
**from** *conseqPost* [*OF this wp' subset-refl*]  
**show** *?thesis* .

**qed**

**lemma** *switchNil*:

$P \subseteq Q \implies \Gamma, \Theta \vdash_{t/F} P \ (switch \ v \ []) \ Q, A$   
 by (*simp add: Skip*)

**lemma** *switchCons*:

$\llbracket P \subseteq \{s. (v \ s \in V \longrightarrow s \in P_1) \wedge (v \ s \notin V \longrightarrow s \in P_2)\};$   
 $\Gamma, \Theta \vdash_{t/F} P_1 \ c \ Q, A;$   
 $\Gamma, \Theta \vdash_{t/F} P_2 \ (switch \ v \ vs) \ Q, A \rrbracket$   
 $\implies \Gamma, \Theta \vdash_{t/F} P \ (switch \ v \ ((V, c) \# vs)) \ Q, A$   
 by (*simp add: Cond*)

**lemma** *Guard*:

$\llbracket P \subseteq g \cap R; \Gamma, \Theta \vdash_{t/F} R \ c \ Q, A \rrbracket$

$\implies \Gamma, \Theta \vdash_{t/F} P \ \text{Guard } f \ g \ c \ Q, A$

**apply** (*rule* *HoareTotalDef.Guard* [*THEN* *conseqPre*, of - - - *R*])

**apply** (*erule* *conseqPre*)

**apply** *auto*

**done**

**lemma** *GuardSwap*:

$\llbracket \Gamma, \Theta \vdash_{t/F} R \ c \ Q, A; P \subseteq g \cap R \rrbracket$

$\implies \Gamma, \Theta \vdash_{t/F} P \ \text{Guard } f \ g \ c \ Q, A$

**by** (*rule* *Guard*)

**lemma** *Guarantee*:

$\llbracket P \subseteq \{s. s \in g \longrightarrow s \in R\}; \Gamma, \Theta \vdash_{t/F} R \ c \ Q, A; f \in F \rrbracket$

$\implies \Gamma, \Theta \vdash_{t/F} P \ (\text{Guard } f \ g \ c) \ Q, A$

**apply** (*rule* *Guarantee* [*THEN* *conseqPre*, of - - - -  $\{s. s \in g \longrightarrow s \in R\}$ ])

**apply** *assumption*

**apply** (*erule* *conseqPre*)

**apply** *auto*

**done**

**lemma** *GuaranteeSwap*:

$\llbracket \Gamma, \Theta \vdash_{t/F} R \ c \ Q, A; P \subseteq \{s. s \in g \longrightarrow s \in R\}; f \in F \rrbracket$

$\implies \Gamma, \Theta \vdash_{t/F} P \ (\text{Guard } f \ g \ c) \ Q, A$

**by** (*rule* *Guarantee*)

**lemma** *GuardStrip*:

$\llbracket P \subseteq R; \Gamma, \Theta \vdash_{t/F} R \ c \ Q, A; f \in F \rrbracket$

$\implies \Gamma, \Theta \vdash_{t/F} P \ (\text{Guard } f \ g \ c) \ Q, A$

**apply** (*rule* *Guarantee* [*THEN* *conseqPre*])

**apply** *auto*

**done**

**lemma** *GuardStripSwap*:

$\llbracket \Gamma, \Theta \vdash_{t/F} R \ c \ Q, A; P \subseteq R; f \in F \rrbracket$

$\implies \Gamma, \Theta \vdash_{t/F} P \ (\text{Guard } f \ g \ c) \ Q, A$

**by** (*rule* *GuardStrip*)

**lemma** *GuaranteeStrip*:

$\llbracket P \subseteq R; \Gamma, \Theta \vdash_{t/F} R \ c \ Q, A; f \in F \rrbracket$

$\implies \Gamma, \Theta \vdash_{t/F} P \ (\text{guaranteeStrip } f \ g \ c) \ Q, A$

**by** (*unfold* *guaranteeStrip-def*) (*rule* *GuardStrip*)

**lemma** *GuaranteeStripSwap*:  
 $\llbracket \Gamma, \Theta \vdash_{t/F} R \ c \ Q, A; P \subseteq R; f \in F \rrbracket$   
 $\implies \Gamma, \Theta \vdash_{t/F} P \ (\text{guaranteeStrip } f \ g \ c) \ Q, A$   
**by** (*unfold guaranteeStrip-def*) (*rule GuardStrip*)

**lemma** *GuaranteeAsGuard*:  
 $\llbracket P \subseteq g \cap R; \Gamma, \Theta \vdash_{t/F} R \ c \ Q, A \rrbracket$   
 $\implies \Gamma, \Theta \vdash_{t/F} P \ \text{guaranteeStrip } f \ g \ c \ Q, A$   
**by** (*unfold guaranteeStrip-def*) (*rule Guard*)

**lemma** *GuaranteeAsGuardSwap*:  
 $\llbracket \Gamma, \Theta \vdash_{t/F} R \ c \ Q, A; P \subseteq g \cap R \rrbracket$   
 $\implies \Gamma, \Theta \vdash_{t/F} P \ \text{guaranteeStrip } f \ g \ c \ Q, A$   
**by** (*rule GuaranteeAsGuard*)

**lemma** *GuardsNil*:  
 $\Gamma, \Theta \vdash_{t/F} P \ c \ Q, A \implies$   
 $\Gamma, \Theta \vdash_{t/F} P \ (\text{guards } [] \ c) \ Q, A$   
**by** *simp*

**lemma** *GuardsCons*:  
 $\Gamma, \Theta \vdash_{t/F} P \ \text{Guard } f \ g \ (\text{guards } gs \ c) \ Q, A \implies$   
 $\Gamma, \Theta \vdash_{t/F} P \ (\text{guards } ((f, g) \# gs) \ c) \ Q, A$   
**by** *simp*

**lemma** *GuardsConsGuaranteeStrip*:  
 $\Gamma, \Theta \vdash_{t/F} P \ \text{guaranteeStrip } f \ g \ (\text{guards } gs \ c) \ Q, A \implies$   
 $\Gamma, \Theta \vdash_{t/F} P \ (\text{guards } (\text{guaranteeStripPair } f \ g \ # gs) \ c) \ Q, A$   
**by** (*simp add: guaranteeStripPair-def guaranteeStrip-def*)

**lemma** *While*:  
**assumes** *P-I*:  $P \subseteq I$   
**assumes** *deriv-body*:  
 $\forall \sigma. \Gamma, \Theta \vdash_{t/F} (\{\sigma\} \cap I \cap b) \ c \ (\{t. (t, \sigma) \in V\} \cap I), A$   
**assumes** *I-Q*:  $I \cap \neg b \subseteq Q$   
**assumes** *wf*: *wf*  $V$   
**shows**  $\Gamma, \Theta \vdash_{t/F} P \ (\text{whileAnno } b \ I \ V \ c) \ Q, A$   
**proof** –  
**from** *wf deriv-body P-I I-Q*  
**show** *?thesis*  
**apply** (*unfold whileAnno-def*)  
**apply** (*erule conseqPrePost [OF HoareTotalDef.While]*)  
**apply** *auto*  
**done**  
**qed**

**lemma** *WhileInvPost*:

**assumes** *P-I*:  $P \subseteq I$

**assumes** *termi-body*:

$\forall \sigma. \Gamma, \Theta \vdash_{t/UNIV} (\{\sigma\} \cap I \cap b) c (\{t. (t, \sigma) \in V\} \cap P), A$

**assumes** *deriv-body*:

$\Gamma, \Theta \vdash_{t/F} (I \cap b) c I, A$

**assumes** *I-Q*:  $I \cap \neg b \subseteq Q$

**assumes** *wf*:  $wf\ V$

**shows**  $\Gamma, \Theta \vdash_{t/F} P\ (whileAnno\ b\ I\ V\ c)\ Q, A$

**proof** –

**have**  $\forall \sigma. \Gamma, \Theta \vdash_{t/F} (\{\sigma\} \cap I \cap b) c (\{t. (t, \sigma) \in V\} \cap I), A$

**proof**

**fix**  $\sigma$

**from** *hoare-sound* [*OF deriv-body*] *hoaret-sound* [*OF termi-body*] [*rule-format*,  
*of*  $\sigma$ ]]

**have**  $\Gamma, \Theta \models_{t/F} (\{\sigma\} \cap I \cap b) c (\{t. (t, \sigma) \in V\} \cap I), A$

**by** (*fastforce simp add: cvalidt-def validt-def cvalid-def valid-def*)

**then**

**show**  $\Gamma, \Theta \vdash_{t/F} (\{\sigma\} \cap I \cap b) c (\{t. (t, \sigma) \in V\} \cap I), A$

**by** (*rule hoaret-complete'*)

**qed**

**from** *While* [*OF P-I this I-Q wf*]

**show** *?thesis* .

**qed**

**lemma**  $\Gamma, \Theta \vdash_{t/F} (P \cap b) c Q, A \implies \Gamma, \Theta \vdash_{t/F} (P \cap b) (Seq\ c\ (Guard\ f\ Q\ Skip))\ Q, A$   
**oops**

*J* will be instantiated by tactic with  $gs' \cap I$  for those guards that are not stripped.

**lemma** *WhileAnnoG*:

$\Gamma, \Theta \vdash_{t/F} P\ (guards\ gs$

$(whileAnno\ b\ J\ V\ (Seq\ c\ (guards\ gs\ Skip))))\ Q, A$

$\implies$

$\Gamma, \Theta \vdash_{t/F} P\ (whileAnnoG\ gs\ b\ I\ V\ c)\ Q, A$

**by** (*simp add: whileAnnoG-def whileAnno-def while-def*)

This form stems from *strip-guards F (whileAnnoG gs b I V c)*

**lemma** *WhileNoGuard'*:

**assumes** *P-I*:  $P \subseteq I$

**assumes** *deriv-body*:  $\forall \sigma. \Gamma, \Theta \vdash_{t/F} (\{\sigma\} \cap I \cap b) c (\{t. (t, \sigma) \in V\} \cap I), A$

**assumes** *I-Q*:  $I \cap \neg b \subseteq Q$

**assumes** *wf*:  $wf\ V$

**shows**  $\Gamma, \Theta \vdash_{t/F} P\ (whileAnno\ b\ I\ V\ (Seq\ c\ Skip))\ Q, A$

**apply** (*rule While [OF P-I - I-Q wf]*)

**apply** (*rule allI*)  
**apply** (*rule Seq*)  
**apply** (*rule deriv-body* [*rule-format*])  
**apply** (*rule hoaret.Skip*)  
**done**

**lemma** *WhileAnnoFix*:

**assumes** *consequence*:  $P \subseteq \{s. (\exists Z. s \in I Z \wedge (I Z \cap -b \subseteq Q))\}$   
**assumes** *bdy*:  $\forall Z \sigma. \Gamma, \Theta \vdash_{t/F} (\{\sigma\} \cap I Z \cap b) (c Z) (\{t. (t, \sigma) \in V Z\} \cap I Z), A$   
**assumes** *bdy-constant*:  $\forall Z. c Z = c \text{ undefined}$   
**assumes** *wf*:  $\forall Z. wf (V Z)$   
**shows**  $\Gamma, \Theta \vdash_{t/F} P (\text{whileAnnoFix } b \ I \ V \ c) \ Q, A$   
**proof** –  
**from** *bdy bdy-constant*  
**have** *bdy'*:  $\bigwedge Z. \forall \sigma. \Gamma, \Theta \vdash_{t/F} (\{\sigma\} \cap I Z \cap b) (c \text{ undefined})$   
 $(\{t. (t, \sigma) \in V Z\} \cap I Z), A$   
**apply** –  
**apply** (*erule-tac x=Z in allE*)  
**apply** (*erule-tac x=Z in allE*)  
**apply** *simp*  
**done**  
**have**  $\forall Z. \Gamma, \Theta \vdash_{t/F} (I Z) (\text{whileAnnoFix } b \ I \ V \ c) (I Z \cap -b), A$   
**apply** *rule*  
**apply** (*unfold whileAnnoFix-def*)  
**apply** (*rule hoaret.While*)  
**apply** (*rule wf* [*rule-format*])  
**apply** (*rule bdy'*)  
**done**  
**then**  
**show** *?thesis*  
**apply** (*rule conseq*)  
**using** *consequence*  
**by** *blast*  
**qed**

**lemma** *WhileAnnoFix'*:

**assumes** *consequence*:  $P \subseteq \{s. (\exists Z. s \in I Z \wedge$   
 $(\forall t. t \in I Z \cap -b \longrightarrow t \in Q))\}$   
**assumes** *bdy*:  $\forall Z \sigma. \Gamma, \Theta \vdash_{t/F} (\{\sigma\} \cap I Z \cap b) (c Z) (\{t. (t, \sigma) \in V Z\} \cap I Z), A$   
**assumes** *bdy-constant*:  $\forall Z. c Z = c \text{ undefined}$   
**assumes** *wf*:  $\forall Z. wf (V Z)$   
**shows**  $\Gamma, \Theta \vdash_{t/F} P (\text{whileAnnoFix } b \ I \ V \ c) \ Q, A$   
**apply** (*rule WhileAnnoFix* [*OF - bdy bdy-constant wf*])  
**using** *consequence by blast*

**lemma** *WhileAnnoGFix*:

**assumes** *whileAnnoFix*:  
 $\Gamma, \Theta \vdash_{t/F} P (\text{guards } gs$



$(\text{whileAnnoFix } b \ J \ V \ (\lambda Z. (\text{Seq } (c \ Z) \ (\text{guards } gs \ \text{Skip})))) \ Q, A$   
**shows**  $\Gamma, \Theta \vdash_{t/F} P \ (\text{whileAnnoGFix } gs \ b \ I \ V \ c) \ Q, A$   
**using** *whileAnnoFix*  
**by** (*simp add: whileAnnoGFix-def whileAnnoFix-def while-def*)

**lemma** *Bind:*

**assumes** *adapt*:  $P \subseteq \{s. s \in P' \ s\}$   
**assumes** *c*:  $\forall s. \Gamma, \Theta \vdash_{t/F} (P' \ s) \ (c \ (e \ s)) \ Q, A$   
**shows**  $\Gamma, \Theta \vdash_{t/F} P \ (\text{bind } e \ c) \ Q, A$   
**apply** (*rule conseq* [**where**  $P' = \lambda Z. \{s. s = Z \wedge s \in P' \ Z\}$  **and**  $Q' = \lambda Z. Q$  **and**  $A' = \lambda Z. A$ ])  
**apply** (*rule allI*)  
**apply** (*unfold bind-def*)  
**apply** (*rule HoareTotalDef.DynCom*)  
**apply** (*rule ballI*)  
**apply** *clarsimp*  
**apply** (*rule conseqPre*)  
**apply** (*rule c* [*rule-format*])  
**apply** *blast*  
**using** *adapt*  
**apply** *blast*  
**done**

**lemma** *Block-exn:*

**assumes** *adapt*:  $P \subseteq \{s. \text{init } s \in P' \ s\}$   
**assumes** *bdy*:  $\forall s. \Gamma, \Theta \vdash_{t/F} (P' \ s) \ \text{bdy } \{t. \text{return } s \ t \in R \ s \ t\}, \{t. \text{result-exn } (\text{return } s \ t) \ t \in A\}$   
**assumes** *c*:  $\forall s \ t. \Gamma, \Theta \vdash_{t/F} (R \ s \ t) \ (c \ s \ t) \ Q, A$   
**shows**  $\Gamma, \Theta \vdash_{t/F} P \ (\text{block-exn } \text{init } \text{bdy } \text{return } \text{result-exn } c) \ Q, A$   
**apply** (*rule conseq* [**where**  $P' = \lambda Z. \{s. s = Z \wedge \text{init } s \in P' \ Z\}$  **and**  $Q' = \lambda Z. Q$  **and**  $A' = \lambda Z. A$ ])  
**prefer** 2  
**using** *adapt*  
**apply** *blast*  
**apply** (*rule allI*)  
**apply** (*unfold block-exn-def*)  
**apply** (*rule HoareTotalDef.DynCom*)  
**apply** (*rule ballI*)  
**apply** *clarsimp*  
**apply** (*rule-tac*  $R = \{t. \text{return } Z \ t \in R \ Z \ t\}$  **in** *SeqSwap* )  
**apply** (*rule-tac*  $P' = \lambda Z'. \{t. t = Z' \wedge \text{return } Z \ t \in R \ Z \ t\}$  **and**  $Q' = \lambda Z'. Q$  **and**  $A' = \lambda Z'. A$  **in** *conseq*)  
**prefer** 2 **apply** *simp*  
**apply** (*rule allI*)  
**apply** (*rule HoareTotalDef.DynCom*)  
**apply** (*clarsimp*)  
**apply** (*rule SeqSwap*)  
**apply** (*rule c* [*rule-format*])

**apply** (*rule Basic*)  
**apply** *clarsimp*  
**apply** (*rule-tac*  $R=\{t. \text{result-exn} (\text{return } Z \ t) \ t \in A\}$  **in** *HoareTotalDef.Catch*)  
**apply** (*rule-tac*  $R=\{i. i \in P' \ Z\}$  **in** *Seq*)  
**apply** (*rule Basic*)  
**apply** *clarsimp*  
**apply** *simp*  
**apply** (*rule bdy* [*rule-format*])  
**apply** (*rule SeqSwap*)  
**apply** (*rule Throw*)  
**apply** (*rule Basic*)  
**apply** *simp*  
**done**

**lemma** *Block*:

**assumes** *adapt*:  $P \subseteq \{s. \text{init } s \in P' \ s\}$   
**assumes** *bdy*:  $\forall s. \Gamma, \Theta \vdash_t / F (P' \ s) \text{ bdy } \{t. \text{return } s \ t \in R \ s \ t\}, \{t. \text{return } s \ t \in A\}$   
**assumes** *c*:  $\forall s \ t. \Gamma, \Theta \vdash_t / F (R \ s \ t) (c \ s \ t) \ Q, A$   
**shows**  $\Gamma, \Theta \vdash_t / F P (\text{block init bdy return } c) \ Q, A$   
**using** *adapt bdy c unfolding block-def by (rule Block-exn)*

**lemma** *BlockSwap*:

**assumes** *c*:  $\forall s \ t. \Gamma, \Theta \vdash_t / F (R \ s \ t) (c \ s \ t) \ Q, A$   
**assumes** *bdy*:  $\forall s. \Gamma, \Theta \vdash_t / F (P' \ s) \text{ bdy } \{t. \text{return } s \ t \in R \ s \ t\}, \{t. \text{return } s \ t \in A\}$   
**assumes** *adapt*:  $P \subseteq \{s. \text{init } s \in P' \ s\}$   
**shows**  $\Gamma, \Theta \vdash_t / F P (\text{block init bdy return } c) \ Q, A$   
**using** *adapt bdy c*  
**by** (*rule Block*)

**lemma** *Block-exnSwap*:

**assumes** *c*:  $\forall s \ t. \Gamma, \Theta \vdash_t / F (R \ s \ t) (c \ s \ t) \ Q, A$   
**assumes** *bdy*:  $\forall s. \Gamma, \Theta \vdash_t / F (P' \ s) \text{ bdy } \{t. \text{return } s \ t \in R \ s \ t\}, \{t. \text{result-exn} (\text{return } s \ t) \ t \in A\}$   
**assumes** *adapt*:  $P \subseteq \{s. \text{init } s \in P' \ s\}$   
**shows**  $\Gamma, \Theta \vdash_t / F P (\text{block-exn init bdy return result-exn } c) \ Q, A$   
**using** *adapt bdy c*  
**by** (*rule Block-exn*)

**lemma** *Block-exnSpec*:

**assumes** *adapt*:  $P \subseteq \{s. \exists Z. \text{init } s \in P' \ Z \wedge$   
 $(\forall t. t \in Q' \ Z \longrightarrow \text{return } s \ t \in R \ s \ t) \wedge$   
 $(\forall t. t \in A' \ Z \longrightarrow \text{result-exn} (\text{return } s \ t) \ t \in A)\}$   
**assumes** *c*:  $\forall s \ t. \Gamma, \Theta \vdash_t / F (R \ s \ t) (c \ s \ t) \ Q, A$   
**assumes** *bdy*:  $\forall Z. \Gamma, \Theta \vdash_t / F (P' \ Z) \text{ bdy } (Q' \ Z), (A' \ Z)$   
**shows**  $\Gamma, \Theta \vdash_t / F P (\text{block-exn init bdy return result-exn } c) \ Q, A$   
**apply** (*rule conseq* [**where**  $P' = \lambda Z. \{s. \text{init } s \in P' \ Z \wedge$   
 $(\forall t. t \in Q' \ Z \longrightarrow \text{return } s \ t \in R \ s \ t) \wedge$

( $\forall t. t \in A' Z \longrightarrow \text{result-exn } (\text{return } s \ t) \ t \in A$ )} **and**

$Q' = \lambda Z. Q$  **and**  
 $A' = \lambda Z. A$ ])  
**prefer** 2  
**using** *adapt*  
**apply** *blast*  
**apply** (*rule allI*)  
**apply** (*unfold block-exn-def*)  
**apply** (*rule HoareTotalDef.DynCom*)  
**apply** (*rule ballI*)  
**apply** *clarsimp*  
**apply** (*rule-tac*  $R = \{t. \text{return } s \ t \in R \ s \ t\}$  **in** *SeqSwap* )  
**apply** (*rule-tac*  $P' = \lambda Z'. \{t. t = Z' \wedge \text{return } s \ t \in R \ s \ t\}$  **and**  
 $Q' = \lambda Z'. Q$  **and**  $A' = \lambda Z'. A$  **in** *conseq*)  
**prefer** 2 **apply** *simp*  
**apply** (*rule allI*)  
**apply** (*rule HoareTotalDef.DynCom*)  
**apply** (*clarsimp*)  
**apply** (*rule SeqSwap*)  
**apply** (*rule c [rule-format]*)  
**apply** (*rule Basic*)  
**apply** *clarsimp*  
**apply** (*rule-tac*  $R = \{t. \text{result-exn } (\text{return } s \ t) \ t \in A\}$  **in** *HoareTotalDef.Catch*)  
**apply** (*rule-tac*  $R = \{i. i \in P' Z\}$  **in** *Seq*)  
**apply** (*rule Basic*)  
**apply** *clarsimp*  
**apply** *simp*  
**apply** (*rule conseq [OF bdy]*)  
**apply** *clarsimp*  
**apply** *blast*  
**apply** (*rule SeqSwap*)  
**apply** (*rule Throw*)  
**apply** (*rule Basic*)  
**apply** *simp*  
**done**

**lemma** *BlockSpec*:

**assumes** *adapt*:  $P \subseteq \{s. \exists Z. \text{init } s \in P' Z \wedge$   
 $(\forall t. t \in Q' Z \longrightarrow \text{return } s \ t \in R \ s \ t) \wedge$   
 $(\forall t. t \in A' Z \longrightarrow \text{return } s \ t \in A)\}$   
**assumes** *c*:  $\forall s \ t. \Gamma, \Theta \vdash_{t/F} (R \ s \ t) \ (c \ s \ t) \ Q, A$   
**assumes** *bdy*:  $\forall Z. \Gamma, \Theta \vdash_{t/F} (P' Z) \ \text{bdy} \ (Q' Z), (A' Z)$   
**shows**  $\Gamma, \Theta \vdash_{t/F} P \ (\text{block } \text{init } \text{bdy } \text{return } c) \ Q, A$   
**using** *adapt c bdy unfolding block-def by (rule Block-exnSpec)*

**lemma** *Throw*:  $P \subseteq A \implies \Gamma, \Theta \vdash_{t/F} P \ \text{Throw} \ Q, A$   
**by** (*rule hoaret.Throw [THEN conseqPre]*)

**lemmas** *Catch* = *hoaret.Catch*

**lemma** *CatchSwap*:  $\llbracket \Gamma, \Theta \vdash_{t/F} R \ c_2 \ Q, A; \Gamma, \Theta \vdash_{t/F} P \ c_1 \ Q, R \rrbracket \implies \Gamma, \Theta \vdash_{t/F} P \ \text{Catch}$   
 $c_1 \ c_2 \ Q, A$   
 by (rule *hoaret.Catch*)

**lemma** *raise*:  $P \subseteq \{s. f \ s \in A\} \implies \Gamma, \Theta \vdash_{t/F} P \ \text{raise } f \ Q, A$   
 apply (simp add: *raise-def*)  
 apply (rule *Seq*)  
 apply (rule *Basic*)  
 apply (assumption)  
 apply (rule *Throw*)  
 apply (rule *subset-refl*)  
 done

**lemma** *condCatch*:  $\llbracket \Gamma, \Theta \vdash_{t/F} P \ c_1 \ Q, ((b \cap R) \cup (-b \cap A)); \Gamma, \Theta \vdash_{t/F} R \ c_2 \ Q, A \rrbracket$   
 $\implies \Gamma, \Theta \vdash_{t/F} P \ \text{condCatch } c_1 \ b \ c_2 \ Q, A$   
 apply (simp add: *condCatch-def*)  
 apply (rule *Catch*)  
 apply assumption  
 apply (rule *CondSwap*)  
 apply (assumption)  
 apply (rule *hoaret.Throw*)  
 apply blast  
 done

**lemma** *condCatchSwap*:  $\llbracket \Gamma, \Theta \vdash_{t/F} R \ c_2 \ Q, A; \Gamma, \Theta \vdash_{t/F} P \ c_1 \ Q, ((b \cap R) \cup (-b \cap A)) \rrbracket$   
 $\implies \Gamma, \Theta \vdash_{t/F} P \ \text{condCatch } c_1 \ b \ c_2 \ Q, A$   
 by (rule *condCatch*)

**lemma** *Proc-ernSpec*:

assumes *adapt*:  $P \subseteq \{s. \exists Z. \text{init } s \in P' \ Z \wedge$   
 $(\forall t. t \in Q' \ Z \longrightarrow \text{return } s \ t \in R \ s \ t) \wedge$   
 $(\forall t. t \in A' \ Z \longrightarrow \text{result-ern } (\text{return } s \ t) \ t \in A)\}$   
 assumes *c*:  $\forall s \ t. \Gamma, \Theta \vdash_{t/F} (R \ s \ t) \ (c \ s \ t) \ Q, A$   
 assumes *p*:  $\forall Z. \Gamma, \Theta \vdash_{t/F} (P' \ Z) \ \text{Call } p \ (Q' \ Z), (A' \ Z)$   
 shows  $\Gamma, \Theta \vdash_{t/F} P \ (\text{call-ern } \text{init } p \ \text{return } \text{result-ern } c) \ Q, A$   
 using *adapt c p*  
 apply (unfold *call-ern-def*)  
 by (rule *Block-ernSpec*)

**lemma** *ProcSpec*:

assumes *adapt*:  $P \subseteq \{s. \exists Z. \text{init } s \in P' \ Z \wedge$   
 $(\forall t. t \in Q' \ Z \longrightarrow \text{return } s \ t \in R \ s \ t) \wedge$   
 $(\forall t. t \in A' \ Z \longrightarrow \text{return } s \ t \in A)\}$   
 assumes *c*:  $\forall s \ t. \Gamma, \Theta \vdash_{t/F} (R \ s \ t) \ (c \ s \ t) \ Q, A$   
 assumes *p*:  $\forall Z. \Gamma, \Theta \vdash_{t/F} (P' \ Z) \ \text{Call } p \ (Q' \ Z), (A' \ Z)$

**shows**  $\Gamma, \Theta \vdash_{t/F} P$  (*call init p return c*)  $Q, A$   
**using** *adapt c p*  
**apply** (*unfold call-def*)  
**by** (*rule BlockSpec*)

**lemma** *Proc-exnSpec'*:

**assumes** *adapt*:  $P \subseteq \{s. \exists Z. \text{init } s \in P' Z \wedge$   
 $(\forall t \in Q' Z. \text{return } s t \in R s t) \wedge$   
 $(\forall t \in A' Z. \text{result-exn } (\text{return } s t) t \in A)\}$   
**assumes** *c*:  $\forall s t. \Gamma, \Theta \vdash_{t/F} (R s t) (c s t) Q, A$   
**assumes** *p*:  $\forall Z. \Gamma, \Theta \vdash_{t/F} (P' Z) \text{Call } p (Q' Z), (A' Z)$   
**shows**  $\Gamma, \Theta \vdash_{t/F} P$  (*call-exn init p return result-exn c*)  $Q, A$   
**apply** (*rule Proc-exnSpec [OF - c p]*)  
**apply** (*insert adapt*)  
**apply** *clarsimp*  
**apply** (*drule (1) subsetD*)  
**apply** (*clarsimp*)  
**apply** (*rule-tac x=Z in exI*)  
**apply** *blast*  
**done**

**lemma** *ProcSpec'*:

**assumes** *adapt*:  $P \subseteq \{s. \exists Z. \text{init } s \in P' Z \wedge$   
 $(\forall t \in Q' Z. \text{return } s t \in R s t) \wedge$   
 $(\forall t \in A' Z. \text{return } s t \in A)\}$   
**assumes** *c*:  $\forall s t. \Gamma, \Theta \vdash_{t/F} (R s t) (c s t) Q, A$   
**assumes** *p*:  $\forall Z. \Gamma, \Theta \vdash_{t/F} (P' Z) \text{Call } p (Q' Z), (A' Z)$   
**shows**  $\Gamma, \Theta \vdash_{t/F} P$  (*call init p return c*)  $Q, A$   
**using** *adapt c p unfolding call-call-exn by (rule Proc-exnSpec')*

**lemma** *Proc-exnSpecNoAbrupt*:

**assumes** *adapt*:  $P \subseteq \{s. \exists Z. \text{init } s \in P' Z \wedge$   
 $(\forall t. t \in Q' Z \longrightarrow \text{return } s t \in R s t)\}$   
**assumes** *c*:  $\forall s t. \Gamma, \Theta \vdash_{t/F} (R s t) (c s t) Q, A$   
**assumes** *p*:  $\forall Z. \Gamma, \Theta \vdash_{t/F} (P' Z) \text{Call } p (Q' Z), \{\}$   
**shows**  $\Gamma, \Theta \vdash_{t/F} P$  (*call-exn init p return result-exn c*)  $Q, A$   
**apply** (*rule Proc-exnSpec [OF - c p]*)  
**using** *adapt*  
**apply** *simp*  
**done**

**lemma** *ProcSpecNoAbrupt*:

**assumes** *adapt*:  $P \subseteq \{s. \exists Z. \text{init } s \in P' Z \wedge$   
 $(\forall t. t \in Q' Z \longrightarrow \text{return } s t \in R s t)\}$   
**assumes** *c*:  $\forall s t. \Gamma, \Theta \vdash_{t/F} (R s t) (c s t) Q, A$   
**assumes** *p*:  $\forall Z. \Gamma, \Theta \vdash_{t/F} (P' Z) \text{Call } p (Q' Z), \{\}$

**shows**  $\Gamma, \Theta \vdash_{t/F} P$  (*call init p return c*)  $Q, A$   
**apply** (*rule ProcSpec [OF - c p]*)  
**using** *adapt*  
**apply** *simp*  
**done**

**lemma** *FCall*:  
 $\Gamma, \Theta \vdash_{t/F} P$  (*call init p return* ( $\lambda s t. c$  (*result t*)))  $Q, A$   
 $\implies \Gamma, \Theta \vdash_{t/F} P$  (*fcall init p return result c*)  $Q, A$   
**by** (*simp add: fcall-def*)

**lemma** *ProcRec*:  
**assumes** *deriv-bodies*:  
 $\forall p \in \text{Procs.}$   
 $\forall \sigma Z. \Gamma, \Theta \cup (\bigcup q \in \text{Procs. } \bigcup Z.$   
 $\{ (P q Z \cap \{s. ((s, q), \sigma, p) \in r\}, q, Q q Z, A q Z) \}$   
 $\vdash_{t/F} (\{\sigma\} \cap P p Z) \text{ (the } (\Gamma p)) \text{ (} Q p Z), (A p Z)$   
**assumes** *wf: wf r*  
**assumes** *Procs-defined: Procs*  $\subseteq \text{dom } \Gamma$   
**shows**  $\forall p \in \text{Procs. } \forall Z.$   
 $\Gamma, \Theta \vdash_{t/F} (P p Z) \text{ Call } p \text{ (} Q p Z), (A p Z)$   
**by** (*intro strip*)  
*(rule HoareTotalDef.CallRec'*  
 $[OF - \text{Procs-defined wf deriv-bodies}]$ ,  
*simp-all*)

**lemma** *ProcRec'*:  
**assumes** *ctxt*:  
 $\Theta' = (\lambda \sigma p. \Theta \cup (\bigcup q \in \text{Procs.}$   
 $\bigcup Z. \{ (P q Z \cap \{s. ((s, q), \sigma, p) \in r\}, q, Q q Z, A q Z) \}))$   
**assumes** *deriv-bodies*:  
 $\forall p \in \text{Procs.}$   
 $\forall \sigma Z. \Gamma, \Theta' \sigma p \vdash_{t/F} (\{\sigma\} \cap P p Z) \text{ (the } (\Gamma p)) \text{ (} Q p Z), (A p Z)$   
**assumes** *wf: wf r*  
**assumes** *Procs-defined: Procs*  $\subseteq \text{dom } \Gamma$   
**shows**  $\forall p \in \text{Procs. } \forall Z. \Gamma, \Theta \vdash_{t/F} (P p Z) \text{ Call } p \text{ (} Q p Z), (A p Z)$   
**using** *ctxt deriv-bodies*  
**apply** *simp*  
**apply** (*erule ProcRec [OF - wf Procs-defined]*)  
**done**

**lemma** *ProcRecList*:  
**assumes** *deriv-bodies*:  
 $\forall p \in \text{set Procs.}$   
 $\forall \sigma Z. \Gamma, \Theta \cup (\bigcup q \in \text{set Procs. } \bigcup Z.$   
 $\{ (P q Z \cap \{s. ((s, q), \sigma, p) \in r\}, q, Q q Z, A q Z) \}$   
 $\vdash_{t/F} (\{\sigma\} \cap P p Z) \text{ (the } (\Gamma p)) \text{ (} Q p Z), (A p Z)$

**assumes** *wf*: *wf r*  
**assumes** *dist*: *distinct Procs*  
**assumes** *Procs-defined*: *set Procs*  $\subseteq$  *dom*  $\Gamma$   
**shows**  $\forall p \in \text{set } \text{Procs}. \forall Z.$   
 $\Gamma, \Theta \vdash_{t/F} (P \ p \ Z) \text{ Call } p \ (Q \ p \ Z), (A \ p \ Z)$   
**using** *deriv-bodies wf Procs-defined*  
**by** (*rule ProcRec*)

**lemma** *ProcRecSpecs*:

$\llbracket \forall \sigma. \forall (P, p, Q, A) \in \text{Specs}.$   
 $\Gamma, \Theta \cup ((\lambda(P, q, Q, A). (P \cap \{s. ((s, q), (\sigma, p)) \in r\}, q, Q, A)) \text{ 'Specs})$   
 $\vdash_{t/F} (\{\sigma\} \cap P) \text{ (the } (\Gamma \ p)) \ Q, A;$   
*wf r*;  
 $\forall (P, p, Q, A) \in \text{Specs}. p \in \text{dom } \Gamma \rrbracket$   
 $\implies \forall (P, p, Q, A) \in \text{Specs}. \Gamma, \Theta \vdash_{t/F} P \text{ (Call } p) \ Q, A$

**apply** (*rule ballI*)  
**apply** (*case-tac x*)  
**apply** (*rename-tac x P p Q A*)  
**apply** *simp*  
**apply** (*rule hoaret.CallRec*)  
**apply** *auto*  
**done**

**lemma** *ProcRec1*:

**assumes** *deriv-body*:  
 $\forall \sigma \ Z. \Gamma, \Theta \cup (\bigcup Z. \{(P \ Z \cap \{s. ((s, p), \sigma, p) \in r\}, p, Q \ Z, A \ Z)\})$   
 $\vdash_{t/F} (\{\sigma\} \cap P \ Z) \text{ (the } (\Gamma \ p)) \ (Q \ Z), (A \ Z)$   
**assumes** *wf*: *wf r*  
**assumes** *p-defined*:  $p \in \text{dom } \Gamma$   
**shows**  $\forall Z. \Gamma, \Theta \vdash_{t/F} (P \ Z) \text{ Call } p \ (Q \ Z), (A \ Z)$

**proof** –

**from** *deriv-body wf p-defined*  
**have**  $\forall p \in \{p\}. \forall Z. \Gamma, \Theta \vdash_{t/F} (P \ Z) \text{ Call } p \ (Q \ Z), (A \ Z)$   
**apply** –  
**apply** (*rule ProcRec [where A= $\lambda p. A$  and P= $\lambda p. P$  and Q= $\lambda p. Q$ ]*)  
**apply** *simp-all*  
**done**  
**thus** *?thesis*  
**by** *simp*

**qed**

**lemma** *ProcNoRec1*:

**assumes** *deriv-body*:  
 $\forall Z. \Gamma, \Theta \vdash_{t/F} (P \ Z) \text{ (the } (\Gamma \ p)) \ (Q \ Z), (A \ Z)$   
**assumes** *p-defined*:  $p \in \text{dom } \Gamma$   
**shows**  $\forall Z. \Gamma, \Theta \vdash_{t/F} (P \ Z) \text{ Call } p \ (Q \ Z), (A \ Z)$

**proof** –

**have**  $\forall \sigma \ Z. \Gamma, \Theta \vdash_{t/F} (\{\sigma\} \cap P \ Z) \text{ (the } (\Gamma \ p)) \ (Q \ Z), (A \ Z)$

**by** (*blast intro: conseqPre deriv-body [rule-format]*)  
**with** *p-defined* **have**  $\forall \sigma Z. \Gamma, \Theta \cup (\bigcup Z. \{(P Z \cap \{s. ((s,p), \sigma, p) \in \{\}\},$   
 $p, Q Z, A Z)\})$   
 $\vdash_{t/F} (\{\sigma\} \cap P Z) (the (\Gamma p)) (Q Z), (A Z)$   
**by** (*blast intro: hoaret-augment-context*)  
**from** *this*  
**show** *?thesis*  
**by** (*rule ProcRec1*) (*auto simp add: p-defined*)  
**qed**

**lemma** *ProcBody:*

**assumes** *WP*:  $P \subseteq P'$   
**assumes** *deriv-body*:  $\Gamma, \Theta \vdash_{t/F} P'$  *body*  $Q, A$   
**assumes** *body*:  $\Gamma p = Some\ body$   
**shows**  $\Gamma, \Theta \vdash_{t/F} P$  *Call*  $p Q, A$   
**apply** (*rule conseqPre [OF - WP]*)  
**apply** (*rule ProcNoRec1 [rule-format, where P= $\lambda Z. P'$  and Q= $\lambda Z. Q$  and A= $\lambda Z. A$ ]*)  
**apply** (*insert body*)  
**apply** *simp*  
**apply** (*rule hoaret-augment-context [OF deriv-body]*)  
**apply** *blast*  
**apply** *fastforce*  
**done**

**lemma** *CallBody:*

**assumes** *adapt*:  $P \subseteq \{s. init\ s \in P'\ s\}$   
**assumes** *bdy*:  $\forall s. \Gamma, \Theta \vdash_{t/F} (P'\ s)$  *body*  $\{t. return\ s\ t \in R\ s\ t\}, \{t. return\ s\ t \in A\}$   
**assumes** *c*:  $\forall s\ t. \Gamma, \Theta \vdash_{t/F} (R\ s\ t) (c\ s\ t) Q, A$   
**assumes** *body*:  $\Gamma p = Some\ body$   
**shows**  $\Gamma, \Theta \vdash_{t/F} P$  (*call init p return c*)  $Q, A$   
**apply** (*unfold call-def*)  
**apply** (*rule Block [OF adapt - c]*)  
**apply** (*rule allI*)  
**apply** (*rule ProcBody [where  $\Gamma = \Gamma$ , OF - bdy [rule-format] body]*)  
**apply** *simp*  
**done**

**lemma** *Call-exnBody:*

**assumes** *adapt*:  $P \subseteq \{s. init\ s \in P'\ s\}$   
**assumes** *bdy*:  $\forall s. \Gamma, \Theta \vdash_{t/F} (P'\ s)$  *body*  $\{t. return\ s\ t \in R\ s\ t\}, \{t. result-exn (return\ s\ t) t \in A\}$   
**assumes** *c*:  $\forall s\ t. \Gamma, \Theta \vdash_{t/F} (R\ s\ t) (c\ s\ t) Q, A$   
**assumes** *body*:  $\Gamma p = Some\ body$   
**shows**  $\Gamma, \Theta \vdash_{t/F} P$  (*call-exn init p return result-exn c*)  $Q, A$   
**apply** (*unfold call-exn-def*)  
**apply** (*rule Block-exn [OF adapt - c]*)  
**apply** (*rule allI*)



**apply** (rule *ProcBody* [where  $\Gamma=\Gamma$ , *OF* - bdy [rule-format] body])  
**apply simp**  
**done**

**lemmas** *ProcModifyReturn* = *HoareTotalProps.ProcModifyReturn*

**lemmas** *ProcModifyReturnSameFaults* = *HoareTotalProps.ProcModifyReturnSameFaults*

**lemmas** *Proc-exnModifyReturn* = *HoareTotalProps.Proc-exnModifyReturn*

**lemmas** *Proc-exnModifyReturnSameFaults* = *HoareTotalProps.Proc-exnModifyReturnSameFaults*

**lemma** *ProcModifyReturnNoAbr*:

**assumes** *spec*:  $\Gamma, \Theta \vdash_t /_F P$  (call init *p* return' *c*) *Q, A*

**assumes** *result-conform*:

$\forall s t. t \in \text{Modif} (init\ s) \longrightarrow (\text{return}'\ s\ t) = (\text{return}\ s\ t)$

**assumes** *modifies-spec*:

$\forall \sigma. \Gamma, \Theta \vdash /_{UNIV} \{\sigma\}$  Call *p* (Modif  $\sigma$ ), {}

**shows**  $\Gamma, \Theta \vdash_t /_F P$  (call init *p* return *c*) *Q, A*

**by** (rule *ProcModifyReturn* [*OF spec result-conform - modifies-spec*]) *simp*

**lemma** *Proc-exnModifyReturnNoAbr*:

**assumes** *spec*:  $\Gamma, \Theta \vdash_t /_F P$  (call-exn init *p* return' result-exn *c*) *Q, A*

**assumes** *result-conform*:

$\forall s t. t \in \text{Modif} (init\ s) \longrightarrow (\text{return}'\ s\ t) = (\text{return}\ s\ t)$

**assumes** *modifies-spec*:

$\forall \sigma. \Gamma, \Theta \vdash /_{UNIV} \{\sigma\}$  Call *p* (Modif  $\sigma$ ), {}

**shows**  $\Gamma, \Theta \vdash_t /_F P$  (call-exn init *p* return result-exn *c*) *Q, A*

**by** (rule *Proc-exnModifyReturn* [*OF spec result-conform - modifies-spec*]) *simp*

**lemma** *ProcModifyReturnNoAbrSameFaults*:

**assumes** *spec*:  $\Gamma, \Theta \vdash_t /_F P$  (call init *p* return' *c*) *Q, A*

**assumes** *result-conform*:

$\forall s t. t \in \text{Modif} (init\ s) \longrightarrow (\text{return}'\ s\ t) = (\text{return}\ s\ t)$

**assumes** *modifies-spec*:

$\forall \sigma. \Gamma, \Theta \vdash /_F \{\sigma\}$  Call *p* (Modif  $\sigma$ ), {}

**shows**  $\Gamma, \Theta \vdash_t /_F P$  (call init *p* return *c*) *Q, A*

**by** (rule *ProcModifyReturnSameFaults* [*OF spec result-conform - modifies-spec*])  
*simp*

**lemma** *Proc-exnModifyReturnNoAbrSameFaults*:

**assumes** *spec*:  $\Gamma, \Theta \vdash_t /_F P$  (call-exn init *p* return' result-exn *c*) *Q, A*

**assumes** *result-conform*:

$\forall s t. t \in \text{Modif} (init\ s) \longrightarrow (\text{return}'\ s\ t) = (\text{return}\ s\ t)$

**assumes** *modifies-spec*:

$\forall \sigma. \Gamma, \Theta \vdash /_F \{\sigma\}$  Call *p* (Modif  $\sigma$ ), {}

**shows**  $\Gamma, \Theta \vdash_t /_F P$  (call-exn init *p* return result-exn *c*) *Q, A*

**by** (rule *Proc-exnModifyReturnSameFaults* [*OF spec result-conform - modifies-spec*])

*simp*

**lemma** *DynProc-exn*:

**assumes** *adapt*:  $P \subseteq \{s. \exists Z. \text{init } s \in P' s Z \wedge$   
 $(\forall t. t \in Q' s Z \longrightarrow \text{return } s t \in R s t) \wedge$   
 $(\forall t. t \in A' s Z \longrightarrow \text{result-exn } (\text{return } s t) t \in A)\}$

**assumes** *c*:  $\forall s t. \Gamma, \Theta \vdash_{t/F} (R s t) (c s t) Q, A$

**assumes** *p*:  $\forall s \in P. \forall Z. \Gamma, \Theta \vdash_{t/F} (P' s Z) \text{Call } (p s) (Q' s Z), (A' s Z)$

**shows**  $\Gamma, \Theta \vdash_{t/F} P \text{dynCall-exn } f \text{UNIV init } p \text{return result-exn } c Q, A$

**apply** (*rule conseq* [**where**  $P' = \lambda Z. \{s. s = Z \wedge s \in P\}$   
**and**  $Q' = \lambda Z. Q$  **and**  $A' = \lambda Z. A$ ])

**prefer** 2

**using** *adapt*

**apply** *blast*

**apply** (*rule allI*)

**apply** (*unfold dynCall-exn-def maybe-guard-UNIV call-exn-def block-exn-def guards.simps*)

**apply** (*rule HoareTotalDef.DynCom*)

**apply** *clarsimp*

**apply** (*rule HoareTotalDef.DynCom*)

**apply** *clarsimp*

**apply** (*frule in-mono* [*rule-format*, *OF adapt*])

**apply** *clarsimp*

**apply** (*rename-tac* *Z'*)

**apply** (*rule-tac*  $R = Q' Z Z'$  **in** *Seq*)

**apply** (*rule CatchSwap*)

**apply** (*rule SeqSwap*)

**apply** (*rule Throw*)

**apply** (*rule subset-refl*)

**apply** (*rule Basic*)

**apply** (*rule subset-refl*)

**apply** (*rule-tac*  $R = \{i. i \in P' Z Z'\}$  **in** *Seq*)

**apply** (*rule Basic*)

**apply** *clarsimp*

**apply** *simp*

**apply** (*rule-tac*  $Q' = Q' Z Z'$  **and**  $A' = A' Z Z'$  **in** *conseqPost*)

**using** *p*

**apply** *clarsimp*

**apply** *simp*

**apply** *clarsimp*

**apply** (*rule-tac*  $P' = \lambda Z''. \{t. t = Z'' \wedge \text{return } Z t \in R Z t\}$  **and**  
 $Q' = \lambda Z''. Q$  **and**  $A' = \lambda Z''. A$  **in** *conseq*)

**prefer** 2 **apply** *simp*

**apply** (*rule allI*)

**apply** (*rule HoareTotalDef.DynCom*)

**apply** *clarsimp*

**apply** (*rule SeqSwap*)

**apply** (*rule c* [*rule-format*])

**apply** (*rule Basic*)

**apply** *clarsimp*

done

**lemma** *DynProc-exn-guards-cons*:

**assumes**  $p: \Gamma, \Theta \vdash_{t/F} P \text{ dynCall-exn } f \text{ UNIV } \text{init } p \text{ return result-exn } c \ Q, A$

**shows**  $\Gamma, \Theta \vdash_{t/F} (g \cap P) \text{ dynCall-exn } f \ g \ \text{init } p \ \text{return result-exn } c \ Q, A$

**using**  $p$  **apply** (*clarsimp simp add: dynCall-exn-def maybe-guard-def*)

**apply** (*rule Guard*)

**apply** (*rule subset-refl*)

**apply** *assumption*

done

**lemma** *DynProc*:

**assumes** *adapt*:  $P \subseteq \{s. \exists Z. \text{init } s \in P' \ s \ Z \wedge$   
 $(\forall t. t \in Q' \ s \ Z \longrightarrow \text{return } s \ t \in R \ s \ t) \wedge$   
 $(\forall t. t \in A' \ s \ Z \longrightarrow \text{return } s \ t \in A)\}$

**assumes**  $c: \forall s \ t. \Gamma, \Theta \vdash_{t/F} (R \ s \ t) \ (c \ s \ t) \ Q, A$

**assumes**  $p: \forall s \in P. \forall Z. \Gamma, \Theta \vdash_{t/F} (P' \ s \ Z) \ \text{Call } (p \ s) \ (Q' \ s \ Z), (A' \ s \ Z)$

**shows**  $\Gamma, \Theta \vdash_{t/F} P \text{ dynCall } \text{init } p \ \text{return } c \ Q, A$

**using** *adapt c p* **unfolding** *dynCall-dynCall-exn* **by** (*rule DynProc-exn*)

**lemma** *DynProc-exn'*:

**assumes** *adapt*:  $P \subseteq \{s. \exists Z. \text{init } s \in P' \ s \ Z \wedge$   
 $(\forall t \in Q' \ s \ Z. \text{return } s \ t \in R \ s \ t) \wedge$   
 $(\forall t \in A' \ s \ Z. \text{result-exn } (\text{return } s \ t) \ t \in A)\}$

**assumes**  $c: \forall s \ t. \Gamma, \Theta \vdash_{t/F} (R \ s \ t) \ (c \ s \ t) \ Q, A$

**assumes**  $p: \forall s \in P. \forall Z. \Gamma, \Theta \vdash_{t/F} (P' \ s \ Z) \ \text{Call } (p \ s) \ (Q' \ s \ Z), (A' \ s \ Z)$

**shows**  $\Gamma, \Theta \vdash_{t/F} P \text{ dynCall-exn } f \ \text{UNIV } \text{init } p \ \text{return result-exn } c \ Q, A$

**proof** –

**from** *adapt* **have**  $P \subseteq \{s. \exists Z. \text{init } s \in P' \ s \ Z \wedge$   
 $(\forall t. t \in Q' \ s \ Z \longrightarrow \text{return } s \ t \in R \ s \ t) \wedge$   
 $(\forall t. t \in A' \ s \ Z \longrightarrow \text{result-exn } (\text{return } s \ t) \ t \in A)\}$

**by** *blast*

**from** *this c p* **show** *?thesis*

**by** (*rule DynProc-exn*)

qed

**lemma** *DynProc'*:

**assumes** *adapt*:  $P \subseteq \{s. \exists Z. \text{init } s \in P' \ s \ Z \wedge$   
 $(\forall t \in Q' \ s \ Z. \text{return } s \ t \in R \ s \ t) \wedge$   
 $(\forall t \in A' \ s \ Z. \text{return } s \ t \in A)\}$

**assumes**  $c: \forall s \ t. \Gamma, \Theta \vdash_{t/F} (R \ s \ t) \ (c \ s \ t) \ Q, A$

**assumes**  $p: \forall s \in P. \forall Z. \Gamma, \Theta \vdash_{t/F} (P' \ s \ Z) \ \text{Call } (p \ s) \ (Q' \ s \ Z), (A' \ s \ Z)$

**shows**  $\Gamma, \Theta \vdash_{t/F} P \text{ dynCall } \text{init } p \ \text{return } c \ Q, A$

**using** *adapt c p* **unfolding** *dynCall-dynCall-exn* **by** (*rule DynProc-exn'*)

**lemma** *DynProc-exnStaticSpec*:

**assumes** *adapt*:  $P \subseteq \{s. s \in S \wedge (\exists Z. \text{init } s \in P' \ Z \wedge$   
 $(\forall \tau. \tau \in Q' \ Z \longrightarrow \text{return } s \ \tau \in R \ s \ \tau) \wedge$

$(\forall \tau. \tau \in A' Z \longrightarrow \text{result-exn } (\text{return } s \tau) \tau \in A))\}$

**assumes**  $c: \forall s t. \Gamma, \Theta \vdash_{t/F} (R s t) (c s t) Q, A$   
**assumes**  $spec: \forall s \in S. \forall Z. \Gamma, \Theta \vdash_{t/F} (P' Z) \text{Call } (p s) (Q' Z), (A' Z)$   
**shows**  $\Gamma, \Theta \vdash_{t/F} P (\text{dynCall-exn } f \text{UNIV } \text{init } p \text{return result-exn } c) Q, A$   
**proof** –  
**from** *adapt* **have**  $P\text{-}S: P \subseteq S$   
**by** *blast*  
**have**  $\Gamma, \Theta \vdash_{t/F} (P \cap S) (\text{dynCall-exn } f \text{UNIV } \text{init } p \text{return result-exn } c) Q, A$   
**apply** (*rule DynProc-exn* [**where**  $P' = \lambda s Z. P' Z$  **and**  $Q' = \lambda s Z. Q' Z$   
**and**  $A' = \lambda s Z. A' Z, OF - c]$ )  
**apply** *clarsimp*  
**apply** (*frule in-mono* [*rule-format*, *OF adapt*])  
**apply** *clarsimp*  
**using** *spec*  
**apply** *clarsimp*  
**done**  
**thus** *?thesis*  
**by** (*rule conseqPre*) (*insert P-S,blast*)  
**qed**

**lemma** *DynProcStaticSpec*:

**assumes**  $adapt: P \subseteq \{s. s \in S \wedge (\exists Z. \text{init } s \in P' Z \wedge$   
 $(\forall \tau. \tau \in Q' Z \longrightarrow \text{return } s \tau \in R s \tau) \wedge$   
 $(\forall \tau. \tau \in A' Z \longrightarrow \text{return } s \tau \in A))\}$   
**assumes**  $c: \forall s t. \Gamma, \Theta \vdash_{t/F} (R s t) (c s t) Q, A$   
**assumes**  $spec: \forall s \in S. \forall Z. \Gamma, \Theta \vdash_{t/F} (P' Z) \text{Call } (p s) (Q' Z), (A' Z)$   
**shows**  $\Gamma, \Theta \vdash_{t/F} P (\text{dynCall } \text{init } p \text{return } c) Q, A$   
**using**  $adapt$   $c$  *spec* **unfolding** *dynCall-dynCall-exn* **by** (*rule DynProc-exnStaticSpec*)

**lemma** *DynProc-exnProcPar*:

**assumes**  $adapt: P \subseteq \{s. p s = q \wedge (\exists Z. \text{init } s \in P' Z \wedge$   
 $(\forall \tau. \tau \in Q' Z \longrightarrow \text{return } s \tau \in R s \tau) \wedge$   
 $(\forall \tau. \tau \in A' Z \longrightarrow \text{result-exn } (\text{return } s \tau) \tau \in A))\}$   
**assumes**  $c: \forall s t. \Gamma, \Theta \vdash_{t/F} (R s t) (c s t) Q, A$   
**assumes**  $spec: \forall Z. \Gamma, \Theta \vdash_{t/F} (P' Z) \text{Call } q (Q' Z), (A' Z)$   
**shows**  $\Gamma, \Theta \vdash_{t/F} P (\text{dynCall-exn } f \text{UNIV } \text{init } p \text{return result-exn } c) Q, A$   
**apply** (*rule DynProc-exnStaticSpec* [**where**  $S = \{s. p s = q\}$ , *simplified*, *OF adapt*  
 $c]$ )  
**using** *spec*  
**apply** *simp*  
**done**

**lemma** *DynProcProcPar*:

**assumes**  $adapt: P \subseteq \{s. p s = q \wedge (\exists Z. \text{init } s \in P' Z \wedge$   
 $(\forall \tau. \tau \in Q' Z \longrightarrow \text{return } s \tau \in R s \tau) \wedge$   
 $(\forall \tau. \tau \in A' Z \longrightarrow \text{return } s \tau \in A))\}$   
**assumes**  $c: \forall s t. \Gamma, \Theta \vdash_{t/F} (R s t) (c s t) Q, A$

**assumes** *spec*:  $\forall Z. \Gamma, \Theta \vdash_{t/F} (P' Z) \text{ Call } q (Q' Z), (A' Z)$   
**shows**  $\Gamma, \Theta \vdash_{t/F} P (\text{dynCall init } p \text{ return } c) Q, A$   
**apply** (*rule DynProcStaticSpec* [**where**  $S = \{s. p s = q\}$ , *simplified*, *OF adapt c*])  
**using** *spec*  
**apply** *simp*  
**done**

**lemma** *DynProc-exnProcParNoAbrupt*:

**assumes** *adapt*:  $P \subseteq \{s. p s = q \wedge (\exists Z. \text{init } s \in P' Z \wedge$   
 $(\forall \tau. \tau \in Q' Z \longrightarrow \text{return } s \tau \in R s \tau))\}$

**assumes** *c*:  $\forall s t. \Gamma, \Theta \vdash_{t/F} (R s t) (c s t) Q, A$

**assumes** *spec*:  $\forall Z. \Gamma, \Theta \vdash_{t/F} (P' Z) \text{ Call } q (Q' Z), \{\}$

**shows**  $\Gamma, \Theta \vdash_{t/F} P (\text{dynCall-exn } f \text{ UNIV init } p \text{ return result-exn } c) Q, A$

**proof** –

**have**  $P \subseteq \{s. p s = q \wedge (\exists Z. \text{init } s \in P' Z \wedge$   
 $(\forall t. t \in Q' Z \longrightarrow \text{return } s t \in R s t) \wedge$   
 $(\forall t. t \in \{\} \longrightarrow \text{result-exn } (\text{return } s t) t \in A))\}$

(**is**  $P \subseteq ?P'$ )

**proof**

**fix** *s*

**assume**  $P: s \in P$

**with** *adapt* **obtain** *Z* **where**

*Pre*:  $p s = q \wedge \text{init } s \in P' Z$  **and**

*adapt-Norm*:  $\forall \tau. \tau \in Q' Z \longrightarrow \text{return } s \tau \in R s \tau$

**by** *blast*

**from** *adapt-Norm*

**have**  $\forall t. t \in Q' Z \longrightarrow \text{return } s t \in R s t$

**by** *auto*

**then**

**show**  $s \in ?P'$

**using** *Pre* **by** *blast*

**qed**

**note**  $P = \text{this}$

**show** *?thesis*

**apply** –

**apply** (*rule DynProc-exnStaticSpec* [**where**  $S = \{s. p s = q\}$ , *simplified*, *OF P*  
*c*])

**apply** (*insert spec*)

**apply** *auto*

**done**

**qed**

**lemma** *DynProcProcParNoAbrupt*:

**assumes** *adapt*:  $P \subseteq \{s. p s = q \wedge (\exists Z. \text{init } s \in P' Z \wedge$   
 $(\forall \tau. \tau \in Q' Z \longrightarrow \text{return } s \tau \in R s \tau))\}$

**assumes** *c*:  $\forall s t. \Gamma, \Theta \vdash_{t/F} (R s t) (c s t) Q, A$

**assumes** *spec*:  $\forall Z. \Gamma, \Theta \vdash_{t/F} (P' Z) \text{ Call } q (Q' Z), \{\}$

**shows**  $\Gamma, \Theta \vdash_{t/F} P (\text{dynCall init } p \text{ return } c) Q, A$

**using** *adapt c spec unfolding dynCall-dynCall-exn* **by** (rule *DynProc-exnProcParNoAbrupt*)

**lemma** *DynProc-exnModifyReturnNoAbr*:

**assumes** *to-prove*:  $\Gamma, \Theta \vdash_{t/F} P$  (*dynCall-exn* *f g* *init p* *return'* *result-exn c*) *Q, A*

**assumes** *ret-nrm-modif*:  $\forall s t. t \in (\text{Modif } (\text{init } s))$   
 $\longrightarrow \text{return}' s t = \text{return } s t$

**assumes** *modif-clause*:

$\forall s \in P. \forall \sigma. \Gamma, \Theta \vdash_{UNIV} \{\sigma\} \text{Call } (p s) (\text{Modif } \sigma), \{\}$

**shows**  $\Gamma, \Theta \vdash_{t/F} P$  (*dynCall-exn* *f g* *init p* *return* *result-exn c*) *Q, A*

**proof** –

**from** *ret-nrm-modif*

**have**  $\forall s t. t \in (\text{Modif } (\text{init } s))$   
 $\longrightarrow \text{return}' s t = \text{return } s t$

**by** *iprover*

**then**

**have** *ret-nrm-modif'*:  $\forall s t. t \in (\text{Modif } (\text{init } s))$   
 $\longrightarrow \text{return}' s t = \text{return } s t$

**by** *simp*

**have** *ret-abr-modif'*:  $\forall s t. t \in \{\}$

$\longrightarrow \text{result-exn } (\text{return}' s t) t = \text{result-exn } (\text{return } s t) t$

**by** *simp*

**from** *to-prove* *ret-nrm-modif'* *ret-abr-modif'* *modif-clause* **show** *?thesis*

**by** (rule *dynProc-exnModifyReturn*)

**qed**

**lemma** *DynProcModifyReturnNoAbr*:

**assumes** *to-prove*:  $\Gamma, \Theta \vdash_{t/F} P$  (*dynCall* *init p* *return'* *c*) *Q, A*

**assumes** *ret-nrm-modif*:  $\forall s t. t \in (\text{Modif } (\text{init } s))$   
 $\longrightarrow \text{return}' s t = \text{return } s t$

**assumes** *modif-clause*:

$\forall s \in P. \forall \sigma. \Gamma, \Theta \vdash_{UNIV} \{\sigma\} \text{Call } (p s) (\text{Modif } \sigma), \{\}$

**shows**  $\Gamma, \Theta \vdash_{t/F} P$  (*dynCall* *init p* *return c*) *Q, A*

**using** *to-prove* *ret-nrm-modif* *modif-clause* **unfolding** *dynCall-dynCall-exn*

**by** (rule *DynProc-exnModifyReturnNoAbr*)

**lemma** *ProcDyn-exnModifyReturnNoAbrSameFaults*:

**assumes** *to-prove*:  $\Gamma, \Theta \vdash_{t/F} P$  (*dynCall-exn* *f g* *init p* *return'* *result-exn c*) *Q, A*

**assumes** *ret-nrm-modif*:  $\forall s t. t \in (\text{Modif } (\text{init } s))$   
 $\longrightarrow \text{return}' s t = \text{return } s t$

**assumes** *modif-clause*:

$\forall s \in P. \forall \sigma. \Gamma, \Theta \vdash_{F} \{\sigma\} (\text{Call } (p s)) (\text{Modif } \sigma), \{\}$

**shows**  $\Gamma, \Theta \vdash_{t/F} P$  (*dynCall-exn* *f g* *init p* *return* *result-exn c*) *Q, A*

**proof** –

**from** *ret-nrm-modif*

**have**  $\forall s t. t \in (\text{Modif } (\text{init } s))$   
 $\longrightarrow \text{return}' s t = \text{return } s t$

**by** *iprover*

**then**

**have**  $ret\text{-}nrm\text{-}modif'$ :  $\forall s t. t \in (Modif (init s))$   
 $\longrightarrow return' s t = return s t$   
**by** *simp*  
**have**  $ret\text{-}abr\text{-}modif'$ :  $\forall s t. t \in \{\}$   
 $\longrightarrow result\text{-}exn (return' s t) t = result\text{-}exn (return s t) t$   
**by** *simp*  
**from** *to-prove*  $ret\text{-}nrm\text{-}modif'$   $ret\text{-}abr\text{-}modif'$  *modif-clause* **show** *?thesis*  
**by** (*rule dynProc-exnModifyReturnSameFaults*)  
**qed**

**lemma** *ProcDynModifyReturnNoAbrSameFaults*:  
**assumes** *to-prove*:  $\Gamma, \Theta \vdash_t /_F P (dynCall init p return' c) Q, A$   
**assumes**  $ret\text{-}nrm\text{-}modif'$ :  $\forall s t. t \in (Modif (init s))$   
 $\longrightarrow return' s t = return s t$   
**assumes** *modif-clause*:  
 $\forall s \in P. \forall \sigma. \Gamma, \Theta \vdash /_F \{\sigma\} (Call (p s)) (Modif \sigma), \{\}$   
**shows**  $\Gamma, \Theta \vdash_t /_F P (dynCall init p return c) Q, A$   
**using** *to-prove*  $ret\text{-}nrm\text{-}modif'$  *modif-clause* **unfolding** *dynCall-dynCall-exn*  
**by** (*rule ProcDyn-exnModifyReturnNoAbrSameFaults*)

**lemma** *Proc-exnProcParModifyReturn*:  
**assumes**  $q: P \subseteq \{s. p s = q\} \cap P'$   
— *DynProcProcPar* introduces the same constraint as first conjunction in  $P'$ , so the *veg* can simplify it.  
**assumes** *to-prove*:  $\Gamma, \Theta \vdash_t /_F P' (dynCall\text{-}exn f g init p return' result\text{-}exn c) Q, A$   
**assumes**  $ret\text{-}nrm\text{-}modif'$ :  $\forall s t. t \in (Modif (init s))$   
 $\longrightarrow return' s t = return s t$   
**assumes**  $ret\text{-}abr\text{-}modif'$ :  $\forall s t. t \in (ModifAbr (init s))$   
 $\longrightarrow result\text{-}exn (return' s t) t = result\text{-}exn (return s t) t$   
**assumes** *modif-clause*:  
 $\forall \sigma. \Gamma, \Theta \vdash /_{UNIV} \{\sigma\} (Call q) (Modif \sigma), (ModifAbr \sigma)$   
**shows**  $\Gamma, \Theta \vdash_t /_F P (dynCall\text{-}exn f g init p return result\text{-}exn c) Q, A$   
**proof** —  
**from** *to-prove* **have**  $\Gamma, \Theta \vdash_t /_F (\{s. p s = q\} \cap P') (dynCall\text{-}exn f g init p return' result\text{-}exn c) Q, A$   
**by** (*rule conseqPre*) *blast*  
**from** *this*  $ret\text{-}nrm\text{-}modif'$   
 $ret\text{-}abr\text{-}modif'$   
**have**  $\Gamma, \Theta \vdash_t /_F (\{s. p s = q\} \cap P') (dynCall\text{-}exn f g init p return result\text{-}exn c) Q, A$   
**by** (*rule dynProc-exnModifyReturn*) (*insert modif-clause, auto*)  
**from** *this*  $q$  **show** *?thesis*  
**by** (*rule conseqPre*)  
**qed**

**lemma** *ProcProcParModifyReturn*:  
**assumes**  $q: P \subseteq \{s. p s = q\} \cap P'$

— *DynProcProcPar* introduces the same constraint as first conjunction in  $P'$ , so the vcg can simplify it.

**assumes** *to-prove*:  $\Gamma, \Theta \vdash_{t/F} P' \text{ (dynCall init p return' c) } Q, A$

**assumes** *ret-nrm-modif*:  $\forall s t. t \in (\text{Modif (init s)})$   
 $\longrightarrow \text{return' s t} = \text{return s t}$

**assumes** *ret-abr-modif*:  $\forall s t. t \in (\text{ModifAbr (init s)})$   
 $\longrightarrow \text{return' s t} = \text{return s t}$

**assumes** *modif-clause*:

$\forall \sigma. \Gamma, \Theta \vdash_{UNIV} \{\sigma\} \text{ (Call q) (Modif } \sigma), (\text{ModifAbr } \sigma)$

**shows**  $\Gamma, \Theta \vdash_{t/F} P \text{ (dynCall init p return c) } Q, A$

**using** *q to-prove ret-nrm-modif ret-abr-modif modif-clause unfolding dynCall-dynCall-exn*  
**by** (*rule Proc-exnProcParModifyReturn*)

**lemma** *Proc-exnProcParModifyReturnSameFaults*:

**assumes** *q*:  $P \subseteq \{s. p s = q\} \cap P'$

— *DynProcProcPar* introduces the same constraint as first conjunction in  $P'$ , so the vcg can simplify it.

**assumes** *to-prove*:  $\Gamma, \Theta \vdash_{t/F} P' \text{ (dynCall-exn f g init p return' result-exn c) } Q, A$

**assumes** *ret-nrm-modif*:  $\forall s t. t \in (\text{Modif (init s)})$   
 $\longrightarrow \text{return' s t} = \text{return s t}$

**assumes** *ret-abr-modif*:  $\forall s t. t \in (\text{ModifAbr (init s)})$   
 $\longrightarrow \text{result-exn (return' s t) t} = \text{result-exn (return s t) t}$

**assumes** *modif-clause*:

$\forall \sigma. \Gamma, \Theta \vdash_{t/F} \{\sigma\} \text{ Call q (Modif } \sigma), (\text{ModifAbr } \sigma)$

**shows**  $\Gamma, \Theta \vdash_{t/F} P \text{ (dynCall-exn f g init p return result-exn c) } Q, A$

**proof** —

**from** *to-prove*

**have**  $\Gamma, \Theta \vdash_{t/F} (\{s. p s = q\} \cap P') \text{ (dynCall-exn f g init p return' result-exn c) } Q, A$

**by** (*rule conseqPre*) *blast*

**from** *this ret-nrm-modif*

*ret-abr-modif*

**have**  $\Gamma, \Theta \vdash_{t/F} (\{s. p s = q\} \cap P') \text{ (dynCall-exn f g init p return result-exn c) } Q, A$

**by** (*rule dynProc-exnModifyReturnSameFaults*) (*insert modif-clause, auto*)

**from** *this q show ?thesis*

**by** (*rule conseqPre*)

**qed**

**lemma** *ProcProcParModifyReturnSameFaults*:

**assumes** *q*:  $P \subseteq \{s. p s = q\} \cap P'$

— *DynProcProcPar* introduces the same constraint as first conjunction in  $P'$ , so the vcg can simplify it.

**assumes** *to-prove*:  $\Gamma, \Theta \vdash_{t/F} P' \text{ (dynCall init p return' c) } Q, A$

**assumes** *ret-nrm-modif*:  $\forall s t. t \in (\text{Modif (init s)})$   
 $\longrightarrow \text{return' s t} = \text{return s t}$

**assumes** *ret-abr-modif*:  $\forall s t. t \in (\text{ModifAbr (init s)})$



$\longrightarrow \text{return}' s t = \text{return} s t$

**assumes** *modif-clause*:

$\forall \sigma. \Gamma, \Theta \vdash_{/F} \{\sigma\} \text{ Call } q (\text{Modif } \sigma), (\text{ModifAbr } \sigma)$

**shows**  $\Gamma, \Theta \vdash_{/F} P (\text{dynCall init } p \text{ return } c) Q, A$

**using**  $q$  to-prove *ret-nrm-modif* *ret-abr-modif* *modif-clause* **unfolding** *dynCall-dynCall-exn*  
**by** (rule *Proc-exnProcParModifyReturnSameFaults*)

**lemma** *Proc-exnProcParModifyReturnNoAbr*:

**assumes**  $q: P \subseteq \{s. p s = q\} \cap P'$

— *DynProcProcParNoAbrupt* introduces the same constraint as first conjunction in  $P'$ , so the vcg can simplify it.

**assumes** to-prove:  $\Gamma, \Theta \vdash_{/F} P' (\text{dynCall-exn } f g \text{ init } p \text{ return}' \text{ result-exn } c) Q, A$

**assumes** *ret-nrm-modif*:  $\forall s t. t \in (\text{Modif } (\text{init } s))$

$\longrightarrow \text{return}' s t = \text{return} s t$

**assumes** *modif-clause*:

$\forall \sigma. \Gamma, \Theta \vdash_{/UNIV} \{\sigma\} (\text{Call } q) (\text{Modif } \sigma), \{\}$

**shows**  $\Gamma, \Theta \vdash_{/F} P (\text{dynCall-exn } f g \text{ init } p \text{ return } \text{result-exn } c) Q, A$

**proof** —

**from** to-prove **have**  $\Gamma, \Theta \vdash_{/F} (\{s. p s = q\} \cap P') (\text{dynCall-exn } f g \text{ init } p \text{ return}' \text{ result-exn } c) Q, A$

**by** (rule *conseqPre*) *blast*

**from** *this* *ret-nrm-modif*

**have**  $\Gamma, \Theta \vdash_{/F} (\{s. p s = q\} \cap P') (\text{dynCall-exn } f g \text{ init } p \text{ return } \text{result-exn } c) Q, A$

**by** (rule *DynProc-exnModifyReturnNoAbr*) (*insert modif-clause, auto*)

**from** *this*  $q$  **show** *?thesis*

**by** (rule *conseqPre*)

**qed**

**lemma** *ProcProcParModifyReturnNoAbr*:

**assumes**  $q: P \subseteq \{s. p s = q\} \cap P'$

— *DynProcProcParNoAbrupt* introduces the same constraint as first conjunction in  $P'$ , so the vcg can simplify it.

**assumes** to-prove:  $\Gamma, \Theta \vdash_{/F} P' (\text{dynCall init } p \text{ return}' c) Q, A$

**assumes** *ret-nrm-modif*:  $\forall s t. t \in (\text{Modif } (\text{init } s))$

$\longrightarrow \text{return}' s t = \text{return} s t$

**assumes** *modif-clause*:

$\forall \sigma. \Gamma, \Theta \vdash_{/UNIV} \{\sigma\} (\text{Call } q) (\text{Modif } \sigma), \{\}$

**shows**  $\Gamma, \Theta \vdash_{/F} P (\text{dynCall init } p \text{ return } c) Q, A$

**using**  $q$  to-prove *ret-nrm-modif* *modif-clause* **unfolding** *dynCall-dynCall-exn*

**by** (rule *Proc-exnProcParModifyReturnNoAbr*)

**lemma** *Proc-exnProcParModifyReturnNoAbrSameFaults*:

**assumes**  $q: P \subseteq \{s. p s = q\} \cap P'$

— *DynProcProcParNoAbrupt* introduces the same constraint as first conjunction in  $P'$ , so the vcg can simplify it.

**assumes** to-prove:  $\Gamma, \Theta \vdash_{/F} P' (\text{dynCall-exn } f g \text{ init } p \text{ return}' \text{ result-exn } c) Q, A$

**assumes** *ret-nrm-modif*:  $\forall s t. t \in (\text{Modif } (\text{init } s))$   
 $\longrightarrow \text{return}' s t = \text{return } s t$   
**assumes** *modif-clause*:  
 $\forall \sigma. \Gamma, \Theta \vdash_{/F} \{\sigma\} (\text{Call } q) (\text{Modif } \sigma), \{\}$   
**shows**  $\Gamma, \Theta \vdash_{/F} P (\text{dynCall-exn } f g \text{ init } p \text{ return result-exn } c) Q, A$   
**proof** –  
**from** *to-prove* **have**  
 $\Gamma, \Theta \vdash_{/F} (\{s. p s = q\} \cap P') (\text{dynCall-exn } f g \text{ init } p \text{ return}' \text{ result-exn } c) Q, A$   
**by** (*rule conseqPre*) *blast*  
**from** *this ret-nrm-modif*  
**have**  $\Gamma, \Theta \vdash_{/F} (\{s. p s = q\} \cap P') (\text{dynCall-exn } f g \text{ init } p \text{ return result-exn } c)$   
 $Q, A$   
**by** (*rule ProcDyn-exnModifyReturnNoAbrSameFaults*) (*insert modif-clause, auto*)  
**from** *this q show ?thesis*  
**by** (*rule conseqPre*)  
**qed**

**lemma** *ProcProcParModifyReturnNoAbrSameFaults*:  
**assumes**  $q: P \subseteq \{s. p s = q\} \cap P'$   
– *DynProcProcParNoAbrupt* introduces the same constraint as first conjunction in  $P'$ , so the vcg can simplify it.  
**assumes** *to-prove*:  $\Gamma, \Theta \vdash_{/F} P' (\text{dynCall } \text{init } p \text{ return}' c) Q, A$   
**assumes** *ret-nrm-modif*:  $\forall s t. t \in (\text{Modif } (\text{init } s))$   
 $\longrightarrow \text{return}' s t = \text{return } s t$   
**assumes** *modif-clause*:  
 $\forall \sigma. \Gamma, \Theta \vdash_{/F} \{\sigma\} (\text{Call } q) (\text{Modif } \sigma), \{\}$   
**shows**  $\Gamma, \Theta \vdash_{/F} P (\text{dynCall } \text{init } p \text{ return } c) Q, A$   
**using**  $q$  *to-prove ret-nrm-modif modif-clause unfolding dynCall-dynCall-exn*  
**by** (*rule Proc-exnProcParModifyReturnNoAbrSameFaults*)

**lemma** *MergeGuards-iff*:  $\Gamma, \Theta \vdash_{/F} P \text{ merge-guards } c Q, A = \Gamma, \Theta \vdash_{/F} P c Q, A$   
**by** (*auto intro: MergeGuardsI MergeGuardsD*)

**lemma** *CombineStrip'*:  
**assumes** *deriv*:  $\Gamma, \Theta \vdash_{/F} P c' Q, A$   
**assumes** *deriv-strip-triv*:  $\Gamma, \{\} \vdash_{/\{\}} P c'' \text{UNIV}, \text{UNIV}$   
**assumes**  $c'': c'' = \text{mark-guards False } (\text{strip-guards } (-F) c')$   
**assumes**  $c: \text{merge-guards } c = \text{merge-guards } (\text{mark-guards False } c')$   
**shows**  $\Gamma, \Theta \vdash_{/\{\}} P c Q, A$   
**proof** –  
**from** *deriv-strip-triv* **have** *deriv-strip*:  $\Gamma, \Theta \vdash_{/\{\}} P c'' \text{UNIV}, \text{UNIV}$   
**by** (*auto intro: hoare-augment-context*)  
**from** *deriv-strip* [*simplified c''*]  
**have**  $\Gamma, \Theta \vdash_{/\{\}} P (\text{strip-guards } (-F) c') \text{UNIV}, \text{UNIV}$   
**by** (*rule HoarePartialProps.MarkGuardsD*)  
**with** *deriv*

**have**  $\Gamma, \Theta \vdash_t / \{\}$   $P \ c' \ Q, A$   
**by** (*rule CombineStrip*)  
**hence**  $\Gamma, \Theta \vdash_t / \{\}$   $P \ \text{mark-guards False} \ c' \ Q, A$   
**by** (*rule MarkGuardsI*)  
**hence**  $\Gamma, \Theta \vdash_t / \{\}$   $P \ \text{merge-guards} \ (\text{mark-guards False} \ c') \ Q, A$   
**by** (*rule MergeGuardsI*)  
**hence**  $\Gamma, \Theta \vdash_t / \{\}$   $P \ \text{merge-guards} \ c \ Q, A$   
**by** (*simp add: c*)  
**thus** *?thesis*  
**by** (*rule MergeGuardsD*)  
**qed**

**lemma** *CombineStrip''*:  
**assumes** *deriv*:  $\Gamma, \Theta \vdash_t / \{\text{True}\} \ P \ c' \ Q, A$   
**assumes** *deriv-strip-triv*:  $\Gamma, \{\} \vdash_t / \{\} \ P \ c'' \ \text{UNIV}, \text{UNIV}$   
**assumes**  $c''$ :  $c'' = \text{mark-guards False} \ (\text{strip-guards} \ (\{\text{False}\}) \ c')$   
**assumes**  $c$ :  $\text{merge-guards} \ c = \text{merge-guards} \ (\text{mark-guards False} \ c')$   
**shows**  $\Gamma, \Theta \vdash_t / \{\}$   $P \ c \ Q, A$   
**apply** (*rule CombineStrip' [OF deriv deriv-strip-triv - c]*)  
**apply** (*insert c''*)  
**apply** (*subgoal-tac - \{True\} = \{False\}*)  
**apply** *auto*  
**done**

**lemma** *AsmUN*:  
 $(\bigcup Z. \{(P \ Z, p, Q \ Z, A \ Z)\}) \subseteq \Theta$   
 $\implies$   
 $\forall Z. \Gamma, \Theta \vdash_t / F \ (P \ Z) \ (\text{Call } p) \ (Q \ Z), (A \ Z)$   
**by** (*blast intro: hoaret.Asm*)

**lemma** *hoaret-to-hoarep'*:  
 $\forall Z. \Gamma, \{\} \vdash_t / F \ (P \ Z) \ p \ (Q \ Z), (A \ Z) \implies \forall Z. \Gamma, \{\} \vdash_t / F \ (P \ Z) \ p \ (Q \ Z), (A \ Z)$   
**by** (*iprover intro: total-to-partial*)

**lemma** *augment-context'*:  
 $\llbracket \Theta \subseteq \Theta'; \forall Z. \Gamma, \Theta \vdash_t / F \ (P \ Z) \ p \ (Q \ Z), (A \ Z) \rrbracket$   
 $\implies \forall Z. \Gamma, \Theta' \vdash_t / F \ (P \ Z) \ p \ (Q \ Z), (A \ Z)$   
**by** (*iprover intro: hoaret-augment-context*)

**lemma** *augment-emptyFaults*:  
 $\llbracket \forall Z. \Gamma, \{\} \vdash_t / \{\} \ (P \ Z) \ p \ (Q \ Z), (A \ Z) \rrbracket \implies$   
 $\forall Z. \Gamma, \{\} \vdash_t / F \ (P \ Z) \ p \ (Q \ Z), (A \ Z)$   
**by** (*blast intro: augment-Faults*)

**lemma** *augment-FaultsUNIV*:

$\llbracket \forall Z. \Gamma, \{\} \vdash_{t/F} (P Z) p (Q Z), (A Z) \rrbracket \Longrightarrow$   
 $\forall Z. \Gamma, \{\} \vdash_{t/UNIV} (P Z) p (Q Z), (A Z)$   
**by** (*blast intro: augment-Faults*)

**lemma** *PostConjI* [*trans*]:  
 $\llbracket \Gamma, \Theta \vdash_{t/F} P c Q, A; \Gamma, \Theta \vdash_{t/F} P c R, B \rrbracket \Longrightarrow \Gamma, \Theta \vdash_{t/F} P c (Q \cap R), (A \cap B)$   
**by** (*rule PostConjI*)

**lemma** *PostConjI'* :  
 $\llbracket \Gamma, \Theta \vdash_{t/F} P c Q, A; \Gamma, \Theta \vdash_{t/F} P c Q, A \rrbracket \Longrightarrow \Gamma, \Theta \vdash_{t/F} P c R, B$   
 $\Longrightarrow \Gamma, \Theta \vdash_{t/F} P c (Q \cap R), (A \cap B)$   
**by** (*rule PostConjI iprover+*)

**lemma** *PostConjE* [*consumes 1*]:  
**assumes** *conj*:  $\Gamma, \Theta \vdash_{t/F} P c (Q \cap R), (A \cap B)$   
**assumes** *E*:  $\llbracket \Gamma, \Theta \vdash_{t/F} P c Q, A; \Gamma, \Theta \vdash_{t/F} P c R, B \rrbracket \Longrightarrow S$   
**shows** *S*

**proof** –  
**from** *conj* **have**  $\Gamma, \Theta \vdash_{t/F} P c Q, A$  **by** (*rule conseqPost*) *blast+*  
**moreover**  
**from** *conj* **have**  $\Gamma, \Theta \vdash_{t/F} P c R, B$  **by** (*rule conseqPost*) *blast+*  
**ultimately show** *S*  
**by** (*rule E*)  
**qed**

### 11.0.1 Rules for Single-Step Proof

We are now ready to introduce a set of Hoare rules to be used in single-step structured proofs in Isabelle/Isar.

Assertions of Hoare Logic may be manipulated in calculational proofs, with the inclusion expressed in terms of sets or predicates. Reversed order is supported as well.

**lemma** *annotateI* [*trans*]:  
 $\llbracket \Gamma, \Theta \vdash_{t/F} P \text{ anno } Q, A; c = \text{anno} \rrbracket \Longrightarrow \Gamma, \Theta \vdash_{t/F} P c Q, A$   
**by** (*simp*)

**lemma** *annotate-normI*:  
**assumes** *deriv-anno*:  $\Gamma, \Theta \vdash_{t/F} P \text{ anno } Q, A$   
**assumes** *norm-eq*: *normalize c = normalize anno*  
**shows**  $\Gamma, \Theta \vdash_{t/F} P c Q, A$   
**proof** –  
**from** *HoareTotalProps.NormalizeI* [*OF deriv-anno*] *norm-eq*  
**have**  $\Gamma, \Theta \vdash_{t/F} P \text{ normalize } c Q, A$   
**by** *simp*  
**from** *NormalizeD* [*OF this*]  
**show** *?thesis* .

qed

**lemma** *annotateWhile*:

$\llbracket \Gamma, \Theta \vdash_{t/F} P \text{ (whileAnnoG gs b I V c) } Q, A \rrbracket \Longrightarrow \Gamma, \Theta \vdash_{t/F} P \text{ (while gs b c) } Q, A$   
by (*simp add: whileAnnoG-def*)

**lemma** *reannotateWhile*:

$\llbracket \Gamma, \Theta \vdash_{t/F} P \text{ (whileAnnoG gs b I V c) } Q, A \rrbracket \Longrightarrow \Gamma, \Theta \vdash_{t/F} P \text{ (whileAnnoG gs b J V c) } Q, A$   
by (*simp add: whileAnnoG-def*)

**lemma** *reannotateWhileNoGuard*:

$\llbracket \Gamma, \Theta \vdash_{t/F} P \text{ (whileAnno b I V c) } Q, A \rrbracket \Longrightarrow \Gamma, \Theta \vdash_{t/F} P \text{ (whileAnno b J V c) } Q, A$   
by (*simp add: whileAnno-def*)

**lemma** [*trans*]:  $P' \subseteq P \Longrightarrow \Gamma, \Theta \vdash_{t/F} P c Q, A \Longrightarrow \Gamma, \Theta \vdash_{t/F} P' c Q, A$   
by (*rule conseqPre*)

**lemma** [*trans*]:  $Q \subseteq Q' \Longrightarrow \Gamma, \Theta \vdash_{t/F} P c Q, A \Longrightarrow \Gamma, \Theta \vdash_{t/F} P c Q', A$   
by (*rule conseqPost*) *blast+*

**lemma** [*trans*]:

$\Gamma, \Theta \vdash_{t/F} \{s. P s\} c Q, A \Longrightarrow (\bigwedge s. P' s \longrightarrow P s) \Longrightarrow \Gamma, \Theta \vdash_{t/F} \{s. P' s\} c Q, A$   
by (*rule conseqPre*) *auto*

**lemma** [*trans*]:

$(\bigwedge s. P' s \longrightarrow P s) \Longrightarrow \Gamma, \Theta \vdash_{t/F} \{s. P s\} c Q, A \Longrightarrow \Gamma, \Theta \vdash_{t/F} \{s. P' s\} c Q, A$   
by (*rule conseqPre*) *auto*

**lemma** [*trans*]:

$\Gamma, \Theta \vdash_{t/F} P c \{s. Q s\}, A \Longrightarrow (\bigwedge s. Q s \longrightarrow Q' s) \Longrightarrow \Gamma, \Theta \vdash_{t/F} P c \{s. Q' s\}, A$   
by (*rule conseqPost*) *auto*

**lemma** [*trans*]:

$(\bigwedge s. Q s \longrightarrow Q' s) \Longrightarrow \Gamma, \Theta \vdash_{t/F} P c \{s. Q s\}, A \Longrightarrow \Gamma, \Theta \vdash_{t/F} P c \{s. Q' s\}, A$   
by (*rule conseqPost*) *auto*

**lemma** [*intro?*]:  $\Gamma, \Theta \vdash_{t/F} P \text{ Skip } P, A$

by (*rule Skip*) *auto*

**lemma** *CondInt* [*trans, intro?*]:

$\llbracket \Gamma, \Theta \vdash_{t/F} (P \cap b) c1 Q, A; \Gamma, \Theta \vdash_{t/F} (P \cap - b) c2 Q, A \rrbracket$   
 $\Longrightarrow$   
 $\Gamma, \Theta \vdash_{t/F} P \text{ (Cond b c1 c2) } Q, A$   
by (*rule Cond*) *auto*

```

lemma CondConj [trans, intro?]:
  [[ $\Gamma, \Theta \vdash_{t/F} \{s. P s \wedge b s\}$  c1 Q, A;  $\Gamma, \Theta \vdash_{t/F} \{s. P s \wedge \neg b s\}$  c2 Q, A]
   $\implies$ 
   $\Gamma, \Theta \vdash_{t/F} \{s. P s\}$  (Cond  $\{s. b s\}$  c1 c2) Q, A
  by (rule Cond) auto
end

```

## 12 Auxiliary Definitions/Lemmas to Facilitate Hoare Logic

```

theory Hoare imports HoarePartial HoareTotal begin

```

```

syntax

```

```

-hoarep-emptyFaults::
  [( $'s, 'p, 'f$ ) body, ('s, 'p) quadruple set,
   ( $'f$  set, 's assn, ('s, 'p, 'f) com, 's assn, 's assn]  $\implies$  bool
   ( $\langle (\exists -, -/ \vdash (-) / -, -) \rangle$  [61, 60, 1000, 20, 1000, 1000]60)

-hoarep-emptyCtx::
  [( $'s, 'p, 'f$ ) body, 'f set, 's assn, ('s, 'p, 'f) com, 's assn, 's assn]  $\implies$  bool
   ( $\langle (\exists -/ \vdash \prime_- (-) / -, -) \rangle$  [61, 60, 1000, 20, 1000, 1000]60)

-hoarep-emptyCtx-emptyFaults::
  [( $'s, 'p, 'f$ ) body, 's assn, ('s, 'p, 'f) com, 's assn, 's assn]  $\implies$  bool
   ( $\langle (\exists -/ \vdash (-) / -, -) \rangle$  [61, 1000, 20, 1000, 1000]60)

-hoarep-noAbr::
  [( $'s, 'p, 'f$ ) body, ('s, 'p) quadruple set, 'f set,
   ( $'s$  assn, ('s, 'p, 'f) com, 's assn]  $\implies$  bool
   ( $\langle (\exists -, -/ \vdash \prime_- (-) / -, -) \rangle$  [61, 60, 60, 1000, 20, 1000]60)

-hoarep-noAbr-emptyFaults::
  [( $'s, 'p, 'f$ ) body, ('s, 'p) quadruple set, 's assn, ('s, 'p, 'f) com, 's assn]  $\implies$  bool
   ( $\langle (\exists -, -/ \vdash (-) / -, -) \rangle$  [61, 60, 1000, 20, 1000]60)

-hoarep-emptyCtx-noAbr::
  [( $'s, 'p, 'f$ ) body, 'f set, 's assn, ('s, 'p, 'f) com, 's assn]  $\implies$  bool
   ( $\langle (\exists -/ \vdash \prime_- (-) / -, -) \rangle$  [61, 60, 1000, 20, 1000]60)

-hoarep-emptyCtx-noAbr-emptyFaults::
  [( $'s, 'p, 'f$ ) body, 's assn, ('s, 'p, 'f) com, 's assn]  $\implies$  bool
   ( $\langle (\exists -/ \vdash (-) / -, -) \rangle$  [61, 1000, 20, 1000]60)

```

-hoaret-emptyFaults::  
 [(*'s','p','f'*) body,(*'s','p'*) quadruple set,  
   '*s* assn,(*'s','p','f'*) com, '*s* assn,'*s* assn] => bool  
 (⟨(∃-,/|\_ ( - / - / - / -)⟩ [61,60,1000,20,1000,1000]60)

-hoaret-emptyCtx::  
 [(*'s','p','f'*) body,'*f* set,'*s* assn,(*'s','p','f'*) com, '*s* assn,'*s* assn] => bool  
 (⟨(∃-/|\_ ( - / - / - / -)⟩ [61,60,1000,20,1000,1000]60)

-hoaret-emptyCtx-emptyFaults::  
 [(*'s','p','f'*) body,'*s* assn,(*'s','p','f'*) com, '*s* assn,'*s* assn] => bool  
 (⟨(∃-/|\_ ( - / - / - / -)⟩ [61,1000,20,1000,1000]60)

-hoaret-noAbr::  
 [(*'s','p','f'*) body,'*f* set, (*'s','p'*) quadruple set,  
   '*s* assn,(*'s','p','f'*) com, '*s* assn] => bool  
 (⟨(∃-,/|\_ ( - / - / - / -)⟩ [61,60,60,1000,20,1000]60)

-hoaret-noAbr-emptyFaults::  
 [(*'s','p','f'*) body,(*'s','p'*) quadruple set,'*s* assn,(*'s','p','f'*) com, '*s* assn] => bool  
 (⟨(∃-,/|\_ ( - / - / - / -)⟩ [61,60,1000,20,1000]60)

-hoaret-emptyCtx-noAbr::  
 [(*'s','p','f'*) body,'*f* set,'*s* assn,(*'s','p','f'*) com, '*s* assn] => bool  
 (⟨(∃-/|\_ ( - / - / - / -)⟩ [61,60,1000,20,1000]60)

-hoaret-emptyCtx-noAbr-emptyFaults::  
 [(*'s','p','f'*) body,'*s* assn,(*'s','p','f'*) com, '*s* assn] => bool  
 (⟨(∃-/|\_ ( - / - / - / -)⟩ [61,1000,20,1000]60)

## **syntax (ASCII)**

-hoarep-emptyFaults::  
 [(*'s','p','f'*) body,(*'s','p'*) quadruple set,  
   '*s* assn,(*'s','p','f'*) com, '*s* assn,'*s* assn] => bool  
 (⟨(∃-,/|\_ ( - / - / - / -)⟩ [61,60,1000,20,1000,1000]60)

-hoarep-emptyCtx::  
 [(*'s','p','f'*) body,'*f* set,'*s* assn,(*'s','p','f'*) com, '*s* assn,'*s* assn] => bool  
 (⟨(∃-/|\_ ( - / - / - / -)⟩ [61,60,1000,20,1000,1000]60)

-hoarep-emptyCtx-emptyFaults::  
 [(*'s','p','f'*) body,'*s* assn,(*'s','p','f'*) com, '*s* assn,'*s* assn] => bool  
 (⟨(∃-/|\_ ( - / - / - / -)⟩ [61,1000,20,1000,1000]60)

-hoarep-noAbr::  
 [(*'s','p','f'*) body,(*'s','p'*) quadruple set,'*f* set,  
   '*s* assn,(*'s','p','f'*) com, '*s* assn] => bool

$\langle \langle 3, -, - / - / - (-) / - \rangle \rangle [61, 60, 60, 1000, 20, 1000] 60$

*-hoarep-noAbr-emptyFaults::*

$[(s, p, f) \text{ body}, (s, p) \text{ quadruple set}, s \text{ assn}, (s, p, f) \text{ com}, s \text{ assn}] \Rightarrow \text{bool}$   
 $\langle \langle 3, -, - / - / - (-) / - \rangle \rangle [61, 60, 1000, 20, 1000] 60$

*-hoarep-emptyCtx-noAbr::*

$[(s, p, f) \text{ body}, f \text{ set}, s \text{ assn}, (s, p, f) \text{ com}, s \text{ assn}] \Rightarrow \text{bool}$   
 $\langle \langle 3, - / - / - (-) / - \rangle \rangle [61, 60, 1000, 20, 1000] 60$

*-hoarep-emptyCtx-noAbr-emptyFaults::*

$[(s, p, f) \text{ body}, s \text{ assn}, (s, p, f) \text{ com}, s \text{ assn}] \Rightarrow \text{bool}$   
 $\langle \langle 3, - / - / - (-) / - \rangle \rangle [61, 1000, 20, 1000] 60$

*-hoaret-emptyFault::*

$[(s, p, f) \text{ body}, (s, p) \text{ quadruple set},$   
 $s \text{ assn}, (s, p, f) \text{ com}, s \text{ assn}, s \text{ assn}] \Rightarrow \text{bool}$   
 $\langle \langle 3, -, - / - t (-) / -, - \rangle \rangle [61, 60, 1000, 20, 1000, 1000] 60$

*-hoaret-emptyCtx::*

$[(s, p, f) \text{ body}, f \text{ set}, s \text{ assn}, (s, p, f) \text{ com}, s \text{ assn}, s \text{ assn}] \Rightarrow \text{bool}$   
 $\langle \langle 3, - / - t (-) / -, - \rangle \rangle [61, 60, 1000, 20, 1000, 1000] 60$

*-hoaret-emptyCtx-emptyFaults::*

$[(s, p, f) \text{ body}, s \text{ assn}, (s, p, f) \text{ com}, s \text{ assn}, s \text{ assn}] \Rightarrow \text{bool}$   
 $\langle \langle 3, - / - t (-) / -, - \rangle \rangle [61, 1000, 20, 1000, 1000] 60$

*-hoaret-noAbr::*

$[(s, p, f) \text{ body}, (s, p) \text{ quadruple set}, f \text{ set},$   
 $s \text{ assn}, (s, p, f) \text{ com}, s \text{ assn}] \Rightarrow \text{bool}$   
 $\langle \langle 3, -, - / - t (-) / - \rangle \rangle [61, 60, 60, 1000, 20, 1000] 60$

*-hoaret-noAbr-emptyFaults::*

$[(s, p, f) \text{ body}, (s, p) \text{ quadruple set}, s \text{ assn}, (s, p, f) \text{ com}, s \text{ assn}] \Rightarrow \text{bool}$   
 $\langle \langle 3, -, - / - t (-) / - \rangle \rangle [61, 60, 1000, 20, 1000] 60$

*-hoaret-emptyCtx-noAbr::*

$[(s, p, f) \text{ body}, f \text{ set}, s \text{ assn}, (s, p, f) \text{ com}, s \text{ assn}] \Rightarrow \text{bool}$   
 $\langle \langle 3, - / - t (-) / - \rangle \rangle [61, 60, 1000, 20, 1000] 60$

*-hoaret-emptyCtx-noAbr-emptyFaults::*

$[(s, p, f) \text{ body}, s \text{ assn}, (s, p, f) \text{ com}, s \text{ assn}] \Rightarrow \text{bool}$   
 $\langle \langle 3, - / - t (-) / - \rangle \rangle [61, 1000, 20, 1000] 60$

## translations

$\Gamma \vdash P \text{ c } Q, A \quad == \quad \Gamma \vdash / \{ \} P \text{ c } Q, A$   
 $\Gamma \vdash /_F P \text{ c } Q, A \quad == \quad \Gamma, \{ \} \vdash /_F P \text{ c } Q, A$



$$\begin{aligned} \Gamma, \Theta \vdash P c Q & == \Gamma, \Theta \vdash_{/\{\}} P c Q \\ \Gamma, \Theta \vdash_{/F} P c Q & == \Gamma, \Theta \vdash_{/F} P c Q, \{\} \\ \Gamma, \Theta \vdash P c Q, A & == \Gamma, \Theta \vdash_{/\{\}} P c Q, A \end{aligned}$$

$$\begin{aligned} \Gamma \vdash P c Q & == \Gamma \vdash_{/\{\}} P c Q \\ \Gamma \vdash_{/F} P c Q & == \Gamma, \{\} \vdash_{/F} P c Q \\ \Gamma \vdash_{/F} P c Q & \leq \Gamma \vdash_{/F} P c Q, \{\} \\ \Gamma \vdash P c Q & \leq \Gamma \vdash P c Q, \{\} \end{aligned}$$

$$\begin{aligned} \Gamma \vdash_t P c Q, A & == \Gamma \vdash_t_{/\{\}} P c Q, A \\ \Gamma \vdash_{t/F} P c Q, A & == \Gamma, \{\} \vdash_{t/F} P c Q, A \end{aligned}$$

$$\begin{aligned} \Gamma, \Theta \vdash_t P c Q & == \Gamma, \Theta \vdash_t_{/\{\}} P c Q \\ \Gamma, \Theta \vdash_{t/F} P c Q & == \Gamma, \Theta \vdash_{t/F} P c Q, \{\} \\ \Gamma, \Theta \vdash_t P c Q, A & == \Gamma, \Theta \vdash_t_{/\{\}} P c Q, A \end{aligned}$$

$$\begin{aligned} \Gamma \vdash_t P c Q & == \Gamma \vdash_t_{/\{\}} P c Q \\ \Gamma \vdash_{t/F} P c Q & == \Gamma, \{\} \vdash_{t/F} P c Q \\ \Gamma \vdash_{t/F} P c Q & \leq \Gamma \vdash_{t/F} P c Q, \{\} \\ \Gamma \vdash_t P c Q & \leq \Gamma \vdash_t P c Q, \{\} \end{aligned}$$

**term**  $\Gamma \vdash P c Q$   
**term**  $\Gamma \vdash P c Q, A$

**term**  $\Gamma \vdash_{/F} P c Q$   
**term**  $\Gamma \vdash_{/F} P c Q, A$

**term**  $\Gamma, \Theta \vdash P c Q$   
**term**  $\Gamma, \Theta \vdash_{/F} P c Q$

**term**  $\Gamma, \Theta \vdash P c Q, A$   
**term**  $\Gamma, \Theta \vdash_{/F} P c Q, A$

**term**  $\Gamma \vdash_t P c Q$   
**term**  $\Gamma \vdash_t P c Q, A$

**term**  $\Gamma \vdash_{t/F} P c Q$   
**term**  $\Gamma \vdash_{t/F} P c Q, A$

**term**  $\Gamma, \Theta \vdash_t P c Q$   
**term**  $\Gamma, \Theta \vdash_{t/F} P c Q$

**term**  $\Gamma, \Theta \vdash P \text{ c } Q, A$   
**term**  $\Gamma, \Theta \vdash_{t/F} P \text{ c } Q, A$

**locale** *hoare* =  
**fixes**  $\Gamma :: ('s, 'p, 'f) \text{ body}$

**primrec** *assoc* ::  $('a \times 'b) \text{ list} \Rightarrow 'a \Rightarrow 'b$   
**where**  
*assoc* []  $x = \text{undefined}$  |  
*assoc* ( $p \# ps$ )  $x = (\text{if } \text{fst } p = x \text{ then } (\text{snd } p) \text{ else } \text{assoc } ps \ x)$

**lemma** *conjE-simp*:  $(P \wedge Q \Longrightarrow \text{PROP } R) \equiv (P \Longrightarrow Q \Longrightarrow \text{PROP } R)$   
**by** *rule simp-all*

**lemma** *CollectInt-iff*:  $\{s. P \ s\} \cap \{s. Q \ s\} = \{s. P \ s \wedge Q \ s\}$   
**by** *auto*

**lemma** *Compl-Collect*:  $\neg(\text{Collect } b) = \{x. \neg(b \ x)\}$   
**by** *fastforce*

**lemma** *Collect-False*:  $\{s. \text{False}\} = \{\}$   
**by** *simp*

**lemma** *Collect-True*:  $\{s. \text{True}\} = \text{UNIV}$   
**by** *simp*

**lemma** *triv-All-eq*:  $\forall x. P \equiv P$   
**by** *simp*

**lemma** *triv-Ex-eq*:  $\exists x. P \equiv P$   
**by** *simp*

**lemma** *Ex-True*:  $\exists b. b$   
**by** *blast*

**lemma** *Ex-False*:  $\exists b. \neg b$   
**by** *blast*

**definition** *mex* ::  $('a \Rightarrow \text{bool}) \Rightarrow \text{bool}$   
**where** *mex*  $P = \text{Ex } P$

**definition** *meq* ::  $'a \Rightarrow 'a \Rightarrow \text{bool}$   
**where** *meq*  $s \ Z = (s = Z)$

**lemma** *subset-unII*:  $A \subseteq B \Longrightarrow A \subseteq B \cup C$   
**by** *blast*

**lemma** *subset-unI2*:  $A \subseteq C \implies A \subseteq B \cup C$

**by** *blast*

**lemma** *split-paired-UN*:  $(\bigcup p. (P p)) = (\bigcup a b. (P (a,b)))$

**by** *auto*

**lemma** *in-insert-hd*:  $f \in \text{insert } f X$

**by** *simp*

**lemma** *lookup-Some-in-dom*:  $\Gamma p = \text{Some } bdy \implies p \in \text{dom } \Gamma$

**by** *auto*

**lemma** *unit-object*:  $(\forall u::\text{unit}. P u) = P ()$

**by** *auto*

**lemma** *unit-ex*:  $(\exists u::\text{unit}. P u) = P ()$

**by** *auto*

**lemma** *unit-meta*:  $(\bigwedge (u::\text{unit}). \text{PROP } P u) \equiv \text{PROP } P ()$

**by** *auto*

**lemma** *unit-UN*:  $(\bigcup z::\text{unit}. P z) = P ()$

**by** *auto*

**lemma** *subset-singleton-insert1*:  $y = x \implies \{y\} \subseteq \text{insert } x A$

**by** *auto*

**lemma** *subset-singleton-insert2*:  $\{y\} \subseteq A \implies \{y\} \subseteq \text{insert } x A$

**by** *auto*

**lemma** *in-Specs-simp*:  $(\forall x \in \bigcup Z. \{(P Z, p, Q Z, A Z)\}. \text{Prop } x) =$   
 $(\forall Z. \text{Prop } (P Z, p, Q Z, A Z))$

**by** *auto*

**lemma** *in-set-Un-simp*:  $(\forall x \in A \cup B. P x) = ((\forall x \in A. P x) \wedge (\forall x \in B. P x))$

**by** *auto*

**lemma** *split-all-conj*:  $(\forall x. P x \wedge Q x) = ((\forall x. P x) \wedge (\forall x. Q x))$

**by** *blast*

**lemma** *image-Un-single-simp*:  $f ` (\bigcup Z. \{P Z\}) = (\bigcup Z. \{f (P Z)\})$

**by** *auto*

**lemma** *measure-lex-prod-def'*:

$f <*\text{mlex}*> r \equiv (\{(x,y). (x,y) \in \text{measure } f \vee f x=f y \wedge (x,y) \in r\})$

**by** (*auto simp add: mlex-prod-def inv-image-def*)

**lemma** *in-measure-iff*:  $(x,y) \in \text{measure } f = (f x < f y)$   
**by** (*simp add: measure-def inv-image-def*)

**lemma** *in-lex-iff*:  
 $((a,b),(x,y)) \in r < *lex* > s = ((a,x) \in r \vee (a=x \wedge (b,y) \in s))$   
**by** (*simp add: lex-prod-def*)

**lemma** *in-mlex-iff*:  
 $(x,y) \in f < *mlex* > r = (f x < f y \vee (f x = f y \wedge (x,y) \in r))$   
**by** (*simp add: measure-lex-prod-def' in-measure-iff*)

**lemma** *in-inv-image-iff*:  $(x,y) \in \text{inv-image } r f = ((f x, f y) \in r)$   
**by** (*simp add: inv-image-def*)

This is actually the same as *wf-mlex*. However, this basic proof took me so long that I'm not willing to delete it.

**lemma** *wf-measure-lex-prod* [*simp,intro*]:  
**assumes** *wf-r*:  $wf\ r$   
**shows**  $wf\ (f < *mlex* > r)$   
**proof** (*rule ccontr*)  
**assume**  $\neg wf\ (f < *mlex* > r)$   
**then**  
**obtain** *g* **where**  $\forall i. (g\ (Suc\ i), g\ i) \in f < *mlex* > r$   
**by** (*auto simp add: wf-iff-no-infinite-down-chain*)  
**hence** *g*:  $\forall i. (g\ (Suc\ i), g\ i) \in \text{measure } f \vee$   
 $f\ (g\ (Suc\ i)) = f\ (g\ i) \wedge (g\ (Suc\ i), g\ i) \in r$   
**by** (*simp add: measure-lex-prod-def'*)  
**hence** *le-g*:  $\forall i. f\ (g\ (Suc\ i)) \leq f\ (g\ i)$   
**by** (*auto simp add: in-measure-iff order-le-less*)  
**have**  $wf\ (\text{measure } f)$   
**by** *simp*  
**hence**  $\forall Q. (\exists x. x \in Q) \longrightarrow (\exists z \in Q. \forall y. (y, z) \in \text{measure } f \longrightarrow y \notin Q)$   
**by** (*simp add: wf-eq-minimal*)  
**from** *this* [*rule-format, of g 'UNIV'*]  
**have**  $\exists z. z \in \text{range } g \wedge (\forall y. (y, z) \in \text{measure } f \longrightarrow y \notin \text{range } g)$   
**by** *auto*  
**then obtain** *z* **where**  
 $z: z \in \text{range } g$  **and**  
 $\text{min-}z: \forall y. f\ y < f\ z \longrightarrow y \notin \text{range } g$   
**by** (*auto simp add: in-measure-iff*)  
**from** *z* **obtain** *k* **where**  
 $k: z = g\ k$   
**by** *auto*  
**have**  $\forall i. k \leq i \longrightarrow f\ (g\ i) = f\ (g\ k)$   
**proof** (*intro allI impI*)  
**fix** *i*  
**assume**  $k \leq i$  **then show**  $f\ (g\ i) = f\ (g\ k)$   
**proof** (*induct i*)

```

case 0
have  $k \leq 0$  by fact hence  $k = 0$  by simp
thus  $f (g 0) = f (g k)$ 
  by simp
next
case (Suc n)
have k-Suc-n:  $k \leq \text{Suc } n$  by fact
then show  $f (g (\text{Suc } n)) = f (g k)$ 
proof (cases  $k = \text{Suc } n$ )
  case True
  thus ?thesis by simp
next
  case False
  with k-Suc-n
  have  $k \leq n$ 
  by simp
  with Suc.hyps
  have n-k:  $f (g n) = f (g k)$  by simp
  from le-g have le:  $f (g (\text{Suc } n)) \leq f (g n)$ 
  by simp
  show ?thesis
  proof (cases  $f (g (\text{Suc } n)) = f (g n)$ )
    case True with n-k show ?thesis by simp
  next
    case False
    with le have  $f (g (\text{Suc } n)) < f (g n)$ 
    by simp
    with n-k k have  $f (g (\text{Suc } n)) < f z$ 
    by simp
    with min-z have  $g (\text{Suc } n) \notin \text{range } g$ 
    by blast
    hence False by simp
    thus ?thesis
    by simp
  qed
qed
qed
qed
with k [symmetric] have  $\forall i. k \leq i \longrightarrow f (g i) = f z$ 
  by simp
hence  $\forall i. k \leq i \longrightarrow f (g (\text{Suc } i)) = f (g i)$ 
  by simp
with g have  $\forall i. k \leq i \longrightarrow (g (\text{Suc } i), (g i)) \in r$ 
  by (auto simp add: in-measure-iff order-less-le )
hence  $\forall i. (g (\text{Suc } (i+k)), (g (i+k))) \in r$ 
  by simp
then
have  $\exists f. \forall i. (f (\text{Suc } i), f i) \in r$ 
  by - (rule exI [where  $x = \lambda i. g (i+k)$ ], simp)

```

**with** *wf-r* **show** *False*  
**by** (*simp add: wf-iff-no-infinite-down-chain*)  
**qed**

**lemmas** *all-imp-to-ex = all-simps (5)*

**lemma** *all-imp-eq-triv*:  $(\forall x. x = k \longrightarrow Q) = Q$   
 $(\forall x. k = x \longrightarrow Q) = Q$

**by** *auto*

**end**

## 13 State Space Template

**theory** *StateSpace* **imports** *Hoare*  
**begin**

**record** *'g state = globals::'g*

**definition**

*upd-globals:: ('g  $\Rightarrow$  'g)  $\Rightarrow$  ('g,'z) state-scheme  $\Rightarrow$  ('g,'z) state-scheme*

**where**

*upd-globals upd s = s(globals := upd (globals s))*

**named-theorems** *state-simp*

**lemma** *upd-globals-conv [state-simp]*: *upd-globals f = ( $\lambda s. s(globals := f (globals s))$ )*

**by** (*rule ext*) (*simp add: upd-globals-def*)

**record** (*'g, 'l*) *state-locals = 'g state +*  
*locals :: 'l*

**type-synonym** (*'g, 'n, 'val*) *stateSP = ('g, 'n  $\Rightarrow$  'val) state-locals*

**type-synonym** (*'g, 'n, 'val, 'x*) *stateSP-scheme = ('g, 'n  $\Rightarrow$  'val, 'x) state-locals-scheme*

**end**

## 14 Alternative Small Step Semantics

**theory** *AlternativeSmallStep* **imports** *HoareTotalDef*  
**begin**

This is the small-step semantics, which is described and used in my PhD-

thesis [9]. It decomposes the statement into a list of statements and finally executes the head. So the redex is always the head of the list. The equivalence between termination (based on the big-step semantics) and the absence of infinite computations in this small-step semantics follows the same lines of reasoning as for the new small-step semantics. However, it is technically more involved since the configurations are more complicated. That's why I switched to the new small-step semantics in the "main trunk". I keep this alternative version and the important proofs in this theory, so that one can compare both approaches.

**14.1 Small-Step Computation:**  $\Gamma \vdash (cs, css, s) \rightarrow (cs', css', s')$

**type-synonym**  $(s, p, f)$  continuation =  $(s, p, f)$  com list  $\times$   $(s, p, f)$  com list

**type-synonym**  $(s, p, f)$  config =  
 $(s, p, f)$  com list  $\times$   $(s, p, f)$  continuation list  $\times$   $(s, f)$  xstate

**inductive** step:: $[(s, p, f)$  body,  $(s, p, f)$  config,  $(s, p, f)$  config]  $\Rightarrow$  bool  
 $(\vdash (- \rightarrow / -) \rangle [81, 81, 81] 100)$

**for**  $\Gamma :: (s, p, f)$  body

**where**

*Skip*:  $\Gamma \vdash (\text{Skip} \# cs, css, \text{Normal } s) \rightarrow (cs, css, \text{Normal } s)$

| *Guard*:  $s \in g \implies \Gamma \vdash (\text{Guard } f g c \# cs, css, \text{Normal } s) \rightarrow (c \# cs, css, \text{Normal } s)$

| *GuardFault*:  $s \notin g \implies \Gamma \vdash (\text{Guard } f g c \# cs, css, \text{Normal } s) \rightarrow (cs, css, \text{Fault } f)$

| *FaultProp*:  $\Gamma \vdash (c \# cs, css, \text{Fault } f) \rightarrow (cs, css, \text{Fault } f)$

| *FaultPropBlock*:  $\Gamma \vdash ([], (nrms, abrs) \# css, \text{Fault } f) \rightarrow (nrms, css, \text{Fault } f)$

| *AbruptProp*:  $\Gamma \vdash (c \# cs, css, \text{Abrupt } s) \rightarrow (cs, css, \text{Abrupt } s)$

| *ExitBlockNormal*:

$\Gamma \vdash ([], (nrms, abrs) \# css, \text{Normal } s) \rightarrow (nrms, css, \text{Normal } s)$

| *ExitBlockAbrupt*:

$\Gamma \vdash ([], (nrms, abrs) \# css, \text{Abrupt } s) \rightarrow (abrs, css, \text{Normal } s)$

| *Basic*:  $\Gamma \vdash (\text{Basic } f \# cs, css, \text{Normal } s) \rightarrow (cs, css, \text{Normal } (f s))$

| *Spec*:  $(s, t) \in r \implies \Gamma \vdash (\text{Spec } r \# cs, css, \text{Normal } s) \rightarrow (cs, css, \text{Normal } t)$

| *SpecStuck*:  $\forall t. (s, t) \notin r \implies \Gamma \vdash (\text{Spec } r \# cs, css, \text{Normal } s) \rightarrow (cs, css, \text{Stuck})$

| *Seq*:  $\Gamma \vdash (\text{Seq } c_1 c_2 \# cs, css, \text{Normal } s) \rightarrow (c_1 \# c_2 \# cs, css, \text{Normal } s)$

| *CondTrue*:  $s \in b \implies \Gamma \vdash (\text{Cond } b c_1 c_2 \# cs, css, \text{Normal } s) \rightarrow (c_1 \# cs, css, \text{Normal } s)$

| *CondFalse*:  $s \notin b \implies \Gamma \vdash (\text{Cond } b c_1 c_2 \# cs, css, \text{Normal } s) \rightarrow (c_2 \# cs, css, \text{Normal } s)$

$| \textit{WhileTrue}: \llbracket s \in b \rrbracket$   
 $\implies$   
 $\Gamma \vdash (\textit{While } b \ c \# cs, css, Normal \ s) \rightarrow (c \# \textit{While } b \ c \# cs, css, Normal \ s)$   
 $| \textit{WhileFalse}: \llbracket s \notin b \rrbracket$   
 $\implies$   
 $\Gamma \vdash (\textit{While } b \ c \# cs, css, Normal \ s) \rightarrow (cs, css, Normal \ s)$

$| \textit{Call}: \Gamma \ p = \textit{Some } bdy \implies$   
 $\Gamma \vdash (\textit{Call } p \# cs, css, Normal \ s) \rightarrow ([bdy], (cs, \textit{Throw} \# cs) \# css, Normal \ s)$

$| \textit{CallUndefined}: \Gamma \ p = \textit{None} \implies$   
 $\Gamma \vdash (\textit{Call } p \# cs, css, Normal \ s) \rightarrow (cs, css, \textit{Stuck})$

$| \textit{StuckProp}: \Gamma \vdash (c \# cs, css, \textit{Stuck}) \rightarrow (cs, css, \textit{Stuck})$   
 $| \textit{StuckPropBlock}: \Gamma \vdash (\llbracket \_, (nrms, abrs) \rrbracket \# css, \textit{Stuck}) \rightarrow (nrms, css, \textit{Stuck})$

$| \textit{DynCom}: \Gamma \vdash (\textit{DynCom } c \# cs, css, Normal \ s) \rightarrow (c \ s \# cs, css, Normal \ s)$

$| \textit{Throw}: \Gamma \vdash (\textit{Throw} \# cs, css, Normal \ s) \rightarrow (cs, css, \textit{Abrupt } s)$   
 $| \textit{Catch}: \Gamma \vdash (\textit{Catch } c_1 \ c_2 \# cs, css, Normal \ s) \rightarrow ([c_1], (cs, c_2 \# cs) \# css, Normal \ s)$

**lemmas** *step-induct = step.induct [of - (c, css, s) (c', css', s'), split-format (complete), case-names*  
*Skip Guard GuardFault FaultProp FaultPropBlock AbruptProp ExitBlockNormal*  
*ExitBlockAbrupt*  
*Basic Spec SpecStuck Seq CondTrue CondFalse WhileTrue WhileFalse Call Cal-*  
*lUndefined*  
*StuckProp StuckPropBlock DynCom Throw Catch, induct set]*

**inductive-cases** *step-elim-cases [cases set]:*

$\Gamma \vdash (c \# cs, css, \textit{Fault } f) \rightarrow u$   
 $\Gamma \vdash (\llbracket \_, css, \textit{Fault } f \rrbracket) \rightarrow u$   
 $\Gamma \vdash (c \# cs, css, \textit{Stuck}) \rightarrow u$   
 $\Gamma \vdash (\llbracket \_, css, \textit{Stuck} \rrbracket) \rightarrow u$   
 $\Gamma \vdash (c \# cs, css, \textit{Abrupt } s) \rightarrow u$   
 $\Gamma \vdash (\llbracket \_, css, \textit{Abrupt } s \rrbracket) \rightarrow u$   
 $\Gamma \vdash (\llbracket \_, css, \textit{Normal } s \rrbracket) \rightarrow u$   
 $\Gamma \vdash (\textit{Skip} \# cs, css, s) \rightarrow u$   
 $\Gamma \vdash (\textit{Guard } f \ g \ c \# cs, css, s) \rightarrow u$   
 $\Gamma \vdash (\textit{Basic } f \# cs, css, s) \rightarrow u$   
 $\Gamma \vdash (\textit{Spec } r \# cs, css, s) \rightarrow u$   
 $\Gamma \vdash (\textit{Seq } c_1 \ c_2 \# cs, css, s) \rightarrow u$   
 $\Gamma \vdash (\textit{Cond } b \ c_1 \ c_2 \# cs, css, s) \rightarrow u$   
 $\Gamma \vdash (\textit{While } b \ c \# cs, css, s) \rightarrow u$   
 $\Gamma \vdash (\textit{Call } p \# cs, css, s) \rightarrow u$   
 $\Gamma \vdash (\textit{DynCom } c \# cs, css, s) \rightarrow u$   
 $\Gamma \vdash (\textit{Throw} \# cs, css, s) \rightarrow u$   
 $\Gamma \vdash (\textit{Catch } c_1 \ c_2 \# cs, css, s) \rightarrow u$



**inductive-cases** *step-Normal-elim-cases* [cases set]:

$$\begin{aligned} & \Gamma \vdash (c \# cs, css, Fault\ f) \rightarrow u \\ & \Gamma \vdash (\square, css, Fault\ f) \rightarrow u \\ & \Gamma \vdash (c \# cs, css, Stuck) \rightarrow u \\ & \Gamma \vdash (\square, css, Stuck) \rightarrow u \\ & \Gamma \vdash (\square, (nrms, abrs) \# css, Normal\ s) \rightarrow u \\ & \Gamma \vdash (\square, (nrms, abrs) \# css, Abrupt\ s) \rightarrow u \\ & \Gamma \vdash (Skip \# cs, css, Normal\ s) \rightarrow u \\ & \Gamma \vdash (Guard\ f\ g\ c \# cs, css, Normal\ s) \rightarrow u \\ & \Gamma \vdash (Basic\ f \# cs, css, Normal\ s) \rightarrow u \\ & \Gamma \vdash (Spec\ r \# cs, css, Normal\ s) \rightarrow u \\ & \Gamma \vdash (Seq\ c1\ c2 \# cs, css, Normal\ s) \rightarrow u \\ & \Gamma \vdash (Cond\ b\ c1\ c2 \# cs, css, Normal\ s) \rightarrow u \\ & \Gamma \vdash (While\ b\ c \# cs, css, Normal\ s) \rightarrow u \\ & \Gamma \vdash (Call\ p \# cs, css, Normal\ s) \rightarrow u \\ & \Gamma \vdash (DynCom\ c \# cs, css, Normal\ s) \rightarrow u \\ & \Gamma \vdash (Throw \# cs, css, Normal\ s) \rightarrow u \\ & \Gamma \vdash (Catch\ c1\ c2 \# cs, css, Normal\ s) \rightarrow u \end{aligned}$$

**abbreviation**

$$step\text{-}rtrancl :: [(s, p, f)\ body, (s, p, f)\ config, (s, p, f)\ config] \Rightarrow bool$$

$$(\langle \vdash (- \rightarrow^* / -) \rangle [81, 81, 81]\ 100)$$

**where**

$$\Gamma \vdash cs0 \rightarrow^* cs1 \quad == (step\ \Gamma)^{**}\ cs0\ cs1$$

**abbreviation**

$$step\text{-}trancl :: [(s, p, f)\ body, (s, p, f)\ config, (s, p, f)\ config] \Rightarrow bool$$

$$(\langle \vdash (- \rightarrow^+ / -) \rangle [81, 81, 81]\ 100)$$

**where**

$$\Gamma \vdash cs0 \rightarrow^+ cs1 \quad == (step\ \Gamma)^{++}\ cs0\ cs1$$

### 14.1.1 Structural Properties of Small Step Computations

**lemma** *Fault-app-steps*:  $\Gamma \vdash (cs @ xs, css, Fault\ f) \rightarrow^* (xs, css, Fault\ f)$

**proof** (*induct cs*)

**case** *Nil* **thus** ?*case by simp*

**next**

**case** (*Cons c cs*)

**have**  $\Gamma \vdash (c \# cs @ xs, css, Fault\ f) \rightarrow^* (xs, css, Fault\ f)$

**proof** –

**have**  $\Gamma \vdash (c \# cs @ xs, css, Fault\ f) \rightarrow (cs @ xs, css, Fault\ f)$

**by** (*rule step.FaultProp*)

**also**

**have**  $\Gamma \vdash (cs @ xs, css, Fault\ f) \rightarrow^* (xs, css, Fault\ f)$

**by** (*rule Cons.hyps*)

**finally show** ?*thesis* .

**qed**

```

    thus ?case
      by simp
qed

lemma Stuck-app-steps:  $\Gamma \vdash (cs@xs, css, Stuck) \rightarrow^* (xs, css, Stuck)$ 
proof (induct cs)
  case Nil thus ?case by simp
next
  case (Cons c cs)
  have  $\Gamma \vdash (c\#cs@xs, css, Stuck) \rightarrow^* (xs, css, Stuck)$ 
  proof -
    have  $\Gamma \vdash (c\#cs@xs, css, Stuck) \rightarrow (cs@xs, css, Stuck)$ 
      by (rule step.StuckProp)
    also
    have  $\Gamma \vdash (cs@xs, css, Stuck) \rightarrow^* (xs, css, Stuck)$ 
      by (rule Cons.hyps)
    finally show ?thesis .
  qed
  thus ?case
    by simp
qed

```

We can only append commands inside a block, if execution does not enter or exit a block.

```

lemma app-step:
  assumes step:  $\Gamma \vdash (cs, css, s) \rightarrow (cs', css', t)$ 
  shows  $css = css' \implies \Gamma \vdash (cs@xs, css, s) \rightarrow (cs'@xs, css', t)$ 
using step
apply induct
apply (simp-all del: fun-upd-apply, (blast intro: step.intros)+)
done

```

We can append whole blocks, without interfering with the actual block. Outer blocks do not influence execution of inner blocks.

```

lemma app-css-step:
  assumes step:  $\Gamma \vdash (cs, css, s) \rightarrow (cs', css', t)$ 
  shows  $\Gamma \vdash (cs, css@xs, s) \rightarrow (cs', css'@xs, t)$ 
using step
apply induct
apply (simp-all del: fun-upd-apply, (blast intro: step.intros)+)
done

```

```

ML <
  ML-Thms.bind-thm (trancl-induct3, Split-Rule.split-rule @ {context}
    (Rule-Insts.read-instantiate @ {context}
      [(((a, 0), Position.none), (ax, ay, az)),
        (((b, 0), Position.none), (bx, by, bz))] []
      @ {thm tranclp-induct}));
  >

```

**lemma** *app-css-steps*:  
**assumes** *step*:  $\Gamma \vdash (cs, css, s) \rightarrow^+ (cs', css', t)$   
**shows**  $\Gamma \vdash (cs, css @ xs, s) \rightarrow^+ (cs', css' @ xs, t)$   
**apply**(*rule* *trancl-induct3* [*OF step*])  
**apply** (*rule* *app-css-step* [*THEN* *tranclp.r-into-trancl* [*of step*  $\Gamma$ ]], *assumption*)  
**apply**(*blast intro:app-css-step tranclp-trans*)  
**done**

**lemma** *step-Cons'*:  
**assumes** *step*:  $\Gamma \vdash (ccs, css, s) \rightarrow (cs', css', t)$   
**shows**  
 $\bigwedge c \text{ cs. } ccs = c \# cs \implies \exists css''. \text{ css}' = css'' @ css \wedge$   
     (*if*  $css'' = []$  *then*  $\exists p. \text{ cs}' = p @ cs$   
     *else* ( $\exists pnorm \text{ pabr. } css'' = [(pnorm @ cs, pabr @ cs)]$ ))  
**using** *step*  
**by** *induct force+*

**lemma** *step-Cons*:  
**assumes** *step*:  $\Gamma \vdash (c \# cs, css, s) \rightarrow (cs', css', t)$   
**shows**  $\exists pcss. \text{ css}' = pcss @ css \wedge$   
     (*if*  $pcss = []$  *then*  $\exists ps. \text{ cs}' = ps @ cs$   
     *else* ( $\exists pcs\text{-normal } pcs\text{-abrupt. } pcss = [(pcs\text{-normal} @ cs, pcs\text{-abrupt} @ cs)]$ ))  
**using** *step-Cons'* [*OF step*]  
**by** *blast*

**lemma** *step-Nil'*:  
**assumes** *step*:  $\Gamma \vdash (cs, asscss, s) \rightarrow (cs', css', t)$   
**shows**  
 $\bigwedge ass. [cs = []; \text{ asscss} = ass @ css; \text{ ass} \neq Nil] \implies$   
      $\text{ css}' = tl \text{ ass} @ css \wedge$   
     (*case* *s* *of*  
         *Abrupt* *s'*  $\Rightarrow \text{ cs}' = snd \text{ (hd ass)} \wedge t = Normal \text{ s}'$   
         | *-*  $\Rightarrow \text{ cs}' = fst \text{ (hd ass)} \wedge t = s$ )  
**using** *step*  
**by** (*induct*) (*fastforce simp add: neq-Nil-conv*) $+$

**lemma** *step-Nil*:  
**assumes** *step*:  $\Gamma \vdash ([], ass @ css, s) \rightarrow (cs', css', t)$   
**assumes** *ass-not-Nil*:  $ass \neq []$   
**shows**  $\text{ css}' = tl \text{ ass} @ css \wedge$   
     (*case* *s* *of*  
         *Abrupt* *s'*  $\Rightarrow \text{ cs}' = snd \text{ (hd ass)} \wedge t = Normal \text{ s}'$   
         | *-*  $\Rightarrow \text{ cs}' = fst \text{ (hd ass)} \wedge t = s$ )  
**using** *step-Nil'* [*OF step - - ass-not-Nil*]  
**by** *simp*

**lemma** *step-Nil''*:

**assumes** *step*:  $\Gamma \vdash (\[], (pcs\text{-}normal, pcs\text{-}abrupt) \# pcss @ css, s) \rightarrow (cs', pcss @ css, t)$   
**shows** (*case s of*  
 $Abrupt\ s' \Rightarrow cs' = pcs\text{-}abrupt \wedge t = Normal\ s'$   
 $\mid - \Rightarrow cs' = pcs\text{-}normal \wedge t = s$ )  
**using** *step-Nil'* [*OF step, where*  $ass = (pcs\text{-}normal, pcs\text{-}abrupt) \# pcss$  **and**  $css = css$ ]  
**by** (*auto split: xstate.splits*)

**lemma** *drop-suffix-css-step'*:  
**assumes** *step*:  $\Gamma \vdash (cs, cssxs, s) \rightarrow (cs', css'xs, t)$   
**shows**  $\bigwedge css\ css'\ xs. \llbracket cssxs = css @ xs; css'xs = css' @ xs \rrbracket$   
 $\implies \Gamma \vdash (cs, css, s) \rightarrow (cs', css', t)$   
**using** *step*  
**apply** *induct*  
**apply** (*fastforce intro: step.intros*) +  
**done**

**lemma** *drop-suffix-css-step*:  
**assumes** *step*:  $\Gamma \vdash (cs, pcss @ css, s) \rightarrow (cs', pcss' @ css, t)$   
**shows**  $\Gamma \vdash (cs, pcss, s) \rightarrow (cs', pcss', t)$   
**using** *step by* (*blast intro: drop-suffix-css-step'*)

**lemma** *drop-suffix-hd-css-step'*:  
**assumes** *step*:  $\Gamma \vdash (pcs, css, s) \rightarrow (cs', css'css, t)$   
**shows**  $\bigwedge p\ ps\ cs\ pnorm\ pabr. \llbracket pcs = p \# ps @ cs; css'css = (pnorm @ cs, pabr @ cs) \# css \rrbracket$   
 $\implies \Gamma \vdash (p \# ps, css, s) \rightarrow (cs', (pnorm, pabr) \# css, t)$   
**using** *step*  
**by** *induct* (*force intro: step.intros*) +

**lemma** *drop-suffix-hd-css-step''*:  
**assumes** *step*:  $\Gamma \vdash (p \# ps @ cs, css, s) \rightarrow (cs', (pnorm @ cs, pabr @ cs) \# css, t)$   
**shows**  $\Gamma \vdash (p \# ps, css, s) \rightarrow (cs', (pnorm, pabr) \# css, t)$   
**using** *drop-suffix-hd-css-step'* [*OF step*]  
**by** *auto*

**lemma** *drop-suffix-hd-css-step*:  
**assumes** *step*:  $\Gamma \vdash (p \# ps @ cs, css, s) \rightarrow (cs', [(pnorm @ ps @ cs, pabr @ ps @ cs)] @ css, t)$   
**shows**  $\Gamma \vdash (p \# ps, css, s) \rightarrow (cs', [(pnorm @ ps, pabr @ ps)] @ css, t)$   
**proof** –  
**from** *step drop-suffix-hd-css-step''* [*of - p ps cs css s cs' pnorm @ ps pabr @ ps t*]  
**show** *?thesis*  
**by** *auto*  
**qed**

**lemma** *drop-suffix'*:  
**assumes** *step*:  $\Gamma \vdash (csss, css, s) \rightarrow (cs'xs, css', t)$   
**shows**  $\bigwedge xs\ cs\ cs'. \llbracket csss = css'; csss = cs @ xs; cs'xs = cs' @ xs; cs \neq [] \rrbracket$   
 $\implies \Gamma \vdash (cs, css, s) \rightarrow (cs', css, t)$   
**using** *step*  
**apply** *induct*

**apply** (*fastforce intro: step.intros simp add: neq-Nil-conv*)+  
**done**

**lemma** *drop-suffix*:

**assumes** *step*:  $\Gamma \vdash (c\#cs@xs,css,s) \rightarrow (cs'@xs,css,t)$   
**shows**  $\Gamma \vdash (c\#cs,css,s) \rightarrow (cs',css,t)$   
**by**(*rule drop-suffix'* [*OF step - -*]) *auto*

**lemma** *drop-suffix-same-css-step*:

**assumes** *step*:  $\Gamma \vdash (cs@xs,css,s) \rightarrow (cs'@xs,css,t)$   
**assumes** *not-Nil*:  $cs \neq []$   
**shows**  $\Gamma \vdash (cs,xss,s) \rightarrow (cs',xss,t)$

**proof** –

**from** *drop-suffix'* [*OF step - - not-Nil*]  
**have**  $\Gamma \vdash (cs,css,s) \rightarrow (cs',css,t)$   
**by** *auto*  
**with** *drop-suffix-css-step* [*of - cs [] css s cs' [] t*]  
**have**  $\Gamma \vdash (cs, [], s) \rightarrow (cs', [], t)$   
**by** *auto*  
**from** *app-css-step* [*OF this*]  
**show** *?thesis*  
**by** *auto*

**qed**

**lemma** *Cons-change-css-step*:

**assumes** *step*:  $\Gamma \vdash (cs,css,s) \rightarrow (cs',css'@css,t)$   
**shows**  $\Gamma \vdash (cs,xss,s) \rightarrow (cs',css'@xss,t)$

**proof** –

**from** *step*  
*drop-suffix-css-step* [**where**  $cs=cs$  **and**  $pcss=[]$  **and**  $css=css$  **and**  $s=s$   
**and**  $cs'=cs'$  **and**  $pcss'=css'$  **and**  $t=t$ ]  
**have**  $\Gamma \vdash (cs, [], s) \rightarrow (cs', css', t)$   
**by** *auto*  
**from** *app-css-step* [**where**  $xs=xss$ , *OF this*]  
**show** *?thesis*  
**by** *auto*

**qed**

**lemma** *Nil-change-css-step*:

**assumes** *step*:  $\Gamma \vdash ([] , ass@css,s) \rightarrow (cs',ass'@css,t)$   
**assumes** *ass-not-Nil*:  $ass \neq []$   
**shows**  $\Gamma \vdash ([] , ass@xss,s) \rightarrow (cs',ass'@xss,t)$

**proof** –

**from** *step* *drop-suffix-css-step* [*of - [] ass css s cs' ass' t*]  
**have**  $\Gamma \vdash ([] , ass, s) \rightarrow (cs', ass', t)$   
**by** *auto*  
**from** *app-css-step* [**where**  $xs=xss$ , *OF this*]  
**show** *?thesis*  
**by** *auto*

qed

### 14.1.2 Equivalence between Big and Small-Step Semantics

```
lemma exec-impl-steps:
  assumes exec:  $\Gamma \vdash \langle c, s \rangle \Rightarrow t$ 
  shows  $\bigwedge cs\ css. \Gamma \vdash (c \# cs, css, s) \rightarrow^* (cs, css, t)$ 
using exec
proof (induct)
  case Skip thus ?case by (blast intro: step.Skip)
next
  case Guard thus ?case by (blast intro: step.Guard rtranclp-trans)
next
  case GuardFault thus ?case by (blast intro: step.GuardFault)
next
  case FaultProp thus ?case by (blast intro: step.FaultProp)
next
  case Basic thus ?case by (blast intro: step.Basic)
next
  case Spec thus ?case by (blast intro: step.Spec)
next
  case SpecStuck thus ?case by (blast intro: step.SpecStuck)
next
  case Seq thus ?case by (blast intro: step.Seq rtranclp-trans)
next
  case CondTrue thus ?case by (blast intro: step.CondTrue rtranclp-trans)
next
  case CondFalse thus ?case by (blast intro: step.CondFalse rtranclp-trans)
next
  case WhileTrue thus ?case by (blast intro: step.WhileTrue rtranclp-trans)
next
  case WhileFalse thus ?case by (blast intro: step.WhileFalse)
next
  case (Call p bdy s s' cs css)
  have bdy:  $\Gamma \vdash p = \text{Some } bdy$  by fact
  have steps-body:  $\Gamma \vdash ([bdy], (cs, Throw \# cs) \# css, Normal\ s) \rightarrow^*$ 
     $([], (cs, Throw \# cs) \# css, s')$  by fact
  show ?case
  proof (cases s')
    case (Normal s'')
    note steps-body
    also from Normal have  $\Gamma \vdash ([], (cs, Throw \# cs) \# css, s') \rightarrow (cs, css, s')$ 
      by (auto intro: step.intros)
    finally show ?thesis
      using bdy
      by (blast intro: step.Call rtranclp-trans)
  next
  case (Abrupt s'')
  with steps-body
```

```

have  $\Gamma \vdash ([bdy], (cs, Throw \# cs) \# css, Normal \ s) \rightarrow^*$ 
       $([], (cs, Throw \# cs) \# css, Abrupt \ s')$  by simp
also have  $\Gamma \vdash ([], (cs, Throw \# cs) \# css, Abrupt \ s') \rightarrow (Throw \# cs, css, Normal \ s')$ 
      by (rule ExitBlockAbrupt)
also have  $\Gamma \vdash (Throw \# cs, css, Normal \ s') \rightarrow (cs, css, Abrupt \ s')$ 
      by (rule Throw)
finally show ?thesis
      using bdy Abrupt
      by (auto intro: step.Call rtranclp-trans)
next
  case Fault
  note steps-body
  also from Fault have  $\Gamma \vdash ([], (cs, Throw \# cs) \# css, s') \rightarrow (cs, css, s')$ 
    by (auto intro: step.intros)
  finally show ?thesis
    using bdy
    by (blast intro: step.Call rtranclp-trans)
next
  case Stuck
  note steps-body
  also from Stuck have  $\Gamma \vdash ([], (cs, Throw \# cs) \# css, s') \rightarrow (cs, css, s')$ 
    by (auto intro: step.intros)
  finally show ?thesis
    using bdy
    by (blast intro: step.Call rtranclp-trans)
qed
next
  case (CallUndefined p s cs css)
  have undef:  $\Gamma \ p = None$  by fact
  hence  $\Gamma \vdash (Call \ p \ \# \ cs, \ css, \ Normal \ s) \rightarrow (cs, \ css, \ Stuck)$ 
    by (rule step.CallUndefined)
  thus ?case ..
next
  case StuckProp thus ?case by (blast intro: step.StuckProp rtrancl-trans)
next
  case DynCom thus ?case by (blast intro: step.DynCom rtranclp-trans)
next
  case Throw thus ?case by (blast intro: step.Throw)
next
  case AbruptProp thus ?case by (blast intro: step.AbruptProp)
next
  case (CatchMatch c1 s s' c2 s'' cs css)
  have steps-c1:  $\Gamma \vdash ([c_1], (cs, c_2 \# cs) \# css, Normal \ s) \rightarrow^*$ 
     $([], (cs, c_2 \# cs) \# css, Abrupt \ s')$  by fact
  also
  have  $\Gamma \vdash ([], (cs, c_2 \# cs) \# css, Abrupt \ s') \rightarrow (c_2 \# cs, css, Normal \ s')$ 
    by (rule ExitBlockAbrupt)
  also
  have steps-c2:  $\Gamma \vdash (c_2 \# cs, css, Normal \ s') \rightarrow^* (cs, css, s')$  by fact

```

```

finally
show  $\Gamma \vdash (\text{Catch } c_1 \ c_2 \ \# \ cs, \ css, \ \text{Normal } s) \rightarrow^* (cs, \ css, \ s')$ 
  by (blast intro: step.Catch rtranclp-trans)
next
case (CatchMiss  $c_1 \ s \ s' \ c_2 \ cs \ css$ )
assume notAbr:  $\neg \text{isAbr } s'$ 
have steps-c1:  $\Gamma \vdash ([c_1], (cs, c_2 \# cs) \# css, \ \text{Normal } s) \rightarrow^* ([], (cs, c_2 \# cs) \# css, \ s')$  by
fact
show  $\Gamma \vdash (\text{Catch } c_1 \ c_2 \ \# \ cs, \ css, \ \text{Normal } s) \rightarrow^* (cs, \ css, \ s')$ 
proof (cases s')
  case (Normal w)
  with steps-c1
  have  $\Gamma \vdash ([c_1], (cs, c_2 \# cs) \# css, \ \text{Normal } s) \rightarrow^* ([], (cs, c_2 \# cs) \# css, \ \text{Normal } w)$ 
    by simp
  also
  have  $\Gamma \vdash ([], (cs, c_2 \# cs) \# css, \ \text{Normal } w) \rightarrow (cs, \ css, \ \text{Normal } w)$ 
    by (rule ExitBlockNormal)
  finally show ?thesis using Normal
    by (auto intro: step.Catch rtranclp-trans)
next
case Abrupt with notAbr show ?thesis by simp
next
case (Fault f)
with steps-c1
have  $\Gamma \vdash ([c_1], (cs, c_2 \# cs) \# css, \ \text{Normal } s) \rightarrow^* ([], (cs, c_2 \# cs) \# css, \ \text{Fault } f)$ 
  by simp
also
have  $\Gamma \vdash ([], (cs, c_2 \# cs) \# css, \ \text{Fault } f) \rightarrow (cs, \ css, \ \text{Fault } f)$ 
  by (rule FaultPropBlock)
finally show ?thesis using Fault
  by (auto intro: step.Catch rtranclp-trans)
next
case Stuck
with steps-c1
have  $\Gamma \vdash ([c_1], (cs, c_2 \# cs) \# css, \ \text{Normal } s) \rightarrow^* ([], (cs, c_2 \# cs) \# css, \ \text{Stuck})$ 
  by simp
also
have  $\Gamma \vdash ([], (cs, c_2 \# cs) \# css, \ \text{Stuck}) \rightarrow (cs, \ css, \ \text{Stuck})$ 
  by (rule StuckPropBlock)
finally show ?thesis using Stuck
  by (auto intro: step.Catch rtranclp-trans)
qed
qed

```

```

inductive execs::[('s, 'p, 'f) body, ('s, 'p, 'f) com list,
  ('s, 'p, 'f) continuation list,
  ('s, 'f) xstate, ('s, 'f) xstate]  $\Rightarrow$  bool
(\vdash \langle -, - \rangle \Rightarrow \rightarrow [50, 50, 50, 50, 50] 50)

```



**for**  $\Gamma :: ('s, 'p, 'f)$  *body*  
**where**  
 $Nil: \Gamma \vdash \langle [], [], s \rangle \Rightarrow s$   
  
| *ExitBlockNormal*:  $\Gamma \vdash \langle nrms, css, Normal\ s \rangle \Rightarrow t$   
 $\Rightarrow$   
 $\Gamma \vdash \langle [], (nrms, abrs) \# css, Normal\ s \rangle \Rightarrow t$   
  
| *ExitBlockAbrupt*:  $\Gamma \vdash \langle abrs, css, Normal\ s \rangle \Rightarrow t$   
 $\Rightarrow$   
 $\Gamma \vdash \langle [], (nrms, abrs) \# css, Abrupt\ s \rangle \Rightarrow t$   
  
| *ExitBlockFault*:  $\Gamma \vdash \langle nrms, css, Fault\ f \rangle \Rightarrow t$   
 $\Rightarrow$   
 $\Gamma \vdash \langle [], (nrms, abrs) \# css, Fault\ f \rangle \Rightarrow t$   
  
| *ExitBlockStuck*:  $\Gamma \vdash \langle nrms, css, Stuck \rangle \Rightarrow t$   
 $\Rightarrow$   
 $\Gamma \vdash \langle [], (nrms, abrs) \# css, Stuck \rangle \Rightarrow t$   
  
| *Cons*:  $\llbracket \Gamma \vdash \langle c, s \rangle \Rightarrow t; \Gamma \vdash \langle cs, css, t \rangle \Rightarrow u \rrbracket$   
 $\Rightarrow$   
 $\Gamma \vdash \langle c \# cs, css, s \rangle \Rightarrow u$

**inductive-cases** *execs-elim-cases* [*cases set*]:

$\Gamma \vdash \langle [], css, s \rangle \Rightarrow t$   
 $\Gamma \vdash \langle c \# cs, css, s \rangle \Rightarrow t$

**ML** ‹

$ML$ -*Thms.bind-thm* (*converse-rtrancl-induct3*, *Split-Rule.split-rule* @{*context*}  
(*Rule-Insts.read-instantiate* @{*context*}  
 $\llbracket (((a, 0), Position.none), (cs, css, s)),$   
 $\llbracket ((b, 0), Position.none), (cs', css', t) \rrbracket \rrbracket$  []  
@{*thm converse-rtranclp-induct*});

›

**lemma** *execs-Fault-end*:

**assumes** *execs*:  $\Gamma \vdash \langle cs, css, s \rangle \Rightarrow t$  **shows**  $s = Fault\ f \Longrightarrow t = Fault\ f$   
**using** *execs*  
**by** (*induct*) (*auto dest: Fault-end*)

**lemma** *execs-Stuck-end*:

**assumes** *execs*:  $\Gamma \vdash \langle cs, css, s \rangle \Rightarrow t$  **shows**  $s = Stuck \Longrightarrow t = Stuck$   
**using** *execs*  
**by** (*induct*) (*auto dest: Stuck-end*)

**theorem** *steps-impl-exec*:

**assumes**  $steps: \Gamma \vdash (cs, css, s) \rightarrow^* ([], [], t)$   
**shows**  $\Gamma \vdash \langle cs, css, s \rangle \Rightarrow t$   
**using**  $steps$   
**proof** (*induct rule: converse-rtrancl-induct3 [consumes 1]*)  
**show**  $\Gamma \vdash \langle [], [], t \rangle \Rightarrow t$  **by** (*rule execs.Nil*)  
**next**  
**fix**  $cs\ css\ s\ cs'\ css'\ w$   
**assume**  $step: \Gamma \vdash (cs, css, s) \rightarrow (cs', css', w)$   
**assume**  $execs: \Gamma \vdash \langle cs', css', w \rangle \Rightarrow t$   
**from**  $step$   
**show**  $\Gamma \vdash \langle cs, css, s \rangle \Rightarrow t$   
**proof** (*cases*)  
**case** (*Catch c1 c2 cs s*)  
**with**  $execs$  **obtain**  $t'$  **where**  
 $exec-c1: \Gamma \vdash \langle c1, Normal\ s \rangle \Rightarrow t'$  **and**  
 $execs-rest: \Gamma \vdash \langle [], (cs, c2 \# cs) \# css, t' \rangle \Rightarrow t$   
**by** (*clarsimp elim!: execs-elim-cases*)  
**have**  $\Gamma \vdash \langle Catch\ c1\ c2\ \# cs, css, Normal\ s \rangle \Rightarrow t$   
**proof** (*cases t'*)  
**case** (*Normal t''*)  
**with**  $exec-c1$  **have**  $\Gamma \vdash \langle Catch\ c1\ c2, Normal\ s \rangle \Rightarrow t'$   
**by** (*auto intro: exec.CatchMiss*)  
**moreover**  
**from**  $execs-rest\ Normal$  **have**  $\Gamma \vdash \langle cs, css, t' \rangle \Rightarrow t$   
**by** (*cases auto*)  
**ultimately show** *?thesis*  
**by** (*rule execs.Cons*)  
**next**  
**case** (*Abrupt t''*)  
**from**  $execs-rest\ Abrupt$  **have**  $\Gamma \vdash \langle c2 \# cs, css, Normal\ t'' \rangle \Rightarrow t$   
**by** (*cases auto*)  
**then obtain**  $v$  **where**  
 $exec-c2: \Gamma \vdash \langle c2, Normal\ t'' \rangle \Rightarrow v$  **and**  
 $rest: \Gamma \vdash \langle cs, css, v \rangle \Rightarrow t$   
**by** *cases*  
**from**  $exec-c1\ Abrupt\ exec-c2$   
**have**  $\Gamma \vdash \langle Catch\ c1\ c2, Normal\ s \rangle \Rightarrow v$   
**by** – (*rule exec.CatchMatch, auto*)  
**from** *this rest*  
**show** *?thesis*  
**by** (*rule execs.Cons*)  
**next**  
**case** (*Fault f*)  
**with**  $exec-c1$  **have**  $\Gamma \vdash \langle Catch\ c1\ c2, Normal\ s \rangle \Rightarrow Fault\ f$   
**by** (*auto intro: exec.intros*)  
**moreover from**  $execs-rest\ Fault$  **have**  $\Gamma \vdash \langle cs, css, Fault\ f \rangle \Rightarrow t$   
**by** (*cases auto*)  
**ultimately show** *?thesis*  
**by** (*rule execs.Cons*)

```

next
  case Stuck
  with exec-c1 have  $\Gamma \langle \text{Catch } c1 \ c2, \text{Normal } s \rangle \Rightarrow \text{Stuck}$ 
    by (auto intro: exec.intros)
  moreover from execs-rest Stuck have  $\Gamma \langle cs, css, \text{Stuck} \rangle \Rightarrow t$ 
    by (cases auto)
  ultimately show ?thesis
    by (rule execs.Cons)
qed
with Catch show ?thesis by simp
next
  case (Call p bdy cs s)
  have bdy:  $\Gamma \ p = \text{Some } bdy$  by fact
  from Call execs obtain t' where
    exec-body:  $\Gamma \langle bdy, \text{Normal } s \rangle \Rightarrow t'$  and
    execs-rest:
       $\Gamma \langle [], (cs, \text{Throw} \# cs) \# css, t' \rangle \Rightarrow t$ 
    by (clarsimp elim!: execs-elim-cases)
  have  $\Gamma \langle \text{Call } p \ \# \ cs, css, \text{Normal } s \rangle \Rightarrow t$ 
  proof (cases t')
    case (Normal t'')
    with exec-body bdy
    have  $\Gamma \langle \text{Call } p, \text{Normal } s \rangle \Rightarrow \text{Normal } t''$ 
      by (auto intro: exec.intros)
    moreover
    from execs-rest Normal
    have  $\Gamma \langle cs, css, \text{Normal } t'' \rangle \Rightarrow t$ 
      by cases auto
    ultimately show ?thesis by (rule execs.Cons)
  next
  case (Abrupt t'')
  with exec-body bdy
  have  $\Gamma \langle \text{Call } p, \text{Normal } s \rangle \Rightarrow \text{Abrupt } t''$ 
    by (auto intro: exec.intros)
  moreover
  from execs-rest Abrupt have
     $\Gamma \langle \text{Throw } \# \ cs, css, \text{Normal } t'' \rangle \Rightarrow t$ 
    by (cases auto)
  then obtain v where v:  $\Gamma \langle \text{Throw}, \text{Normal } t'' \rangle \Rightarrow v$   $\Gamma \langle cs, css, v \rangle \Rightarrow t$ 
    by (clarsimp elim!: execs-elim-cases)
  moreover from v have v=Abrupt t''
    by (auto elim: exec-Normal-elim-cases)
  ultimately
  show ?thesis by (auto intro: execs.Cons)
next
  case (Fault f)
  with exec-body bdy have  $\Gamma \langle \text{Call } p, \text{Normal } s \rangle \Rightarrow \text{Fault } f$ 
    by (auto intro: exec.intros)
  moreover from execs-rest Fault have  $\Gamma \langle cs, css, \text{Fault } f \rangle \Rightarrow t$ 

```

by (cases) (auto elim: execs-elim-cases dest: Fault-end)  
 ultimately  
 show ?thesis by (rule execs.Cons)  
 next  
 case Stuck  
 with exec-body bdy have  $\Gamma \vdash \langle \text{Call } p, \text{Normal } s \rangle \Rightarrow \text{Stuck}$   
 by (auto intro: exec.intros)  
 moreover from execs-rest Stuck have  $\Gamma \vdash \langle cs, css, \text{Stuck} \rangle \Rightarrow t$   
 by (cases) (auto elim: execs-elim-cases dest: Stuck-end)  
 ultimately  
 show ?thesis by (rule execs.Cons)  
 qed  
 with Call show ?thesis by simp  
 qed (insert execs,  
 (blast intro: execs.intros exec.intros elim!: execs-elim-cases)+)  
 qed

**theorem steps-impl-exec:**  
 assumes steps:  $\Gamma \vdash ([c], [], s) \rightarrow^* ([], [], t)$   
 shows  $\Gamma \vdash \langle c, s \rangle \Rightarrow t$   
 using steps-impl-exec [OF steps]  
 by (blast elim: execs-elim-cases)

**corollary steps-eq-exec:**  $\Gamma \vdash ([c], [], s) \rightarrow^* ([], [], t) = \Gamma \vdash \langle c, s \rangle \Rightarrow t$   
 by (blast intro: steps-impl-exec exec-impl-steps)

## 14.2 Infinite Computations: $\text{inf } \Gamma \text{ cs css } s$

**definition inf ::**  
 $[(\text{'s}, \text{'p}, \text{'f}) \text{ body}, (\text{'s}, \text{'p}, \text{'f}) \text{ com list}, (\text{'s}, \text{'p}, \text{'f}) \text{ continuation list}, (\text{'s}, \text{'f}) \text{ xstate}]$   
 $\Rightarrow \text{bool}$   
**where**  $\text{inf } \Gamma \text{ cs css } s = (\exists f. f \ 0 = (cs, css, s) \wedge (\forall i. \Gamma \vdash f \ i \rightarrow f(\text{Suc } i)))$

**lemma not-infI:**  $\llbracket \bigwedge f. \llbracket f \ 0 = (cs, css, s); \bigwedge i. \Gamma \vdash f \ i \rightarrow f(\text{Suc } i) \rrbracket \implies \text{False} \rrbracket$   
 $\implies \neg \text{inf } \Gamma \text{ cs css } s$   
 by (auto simp add: inf-def)

## 14.3 Equivalence of Termination and Absence of Infinite Computations

**inductive terminatess::**  $[(\text{'s}, \text{'p}, \text{'f}) \text{ body}, (\text{'s}, \text{'p}, \text{'f}) \text{ com list},$   
 $(\text{'s}, \text{'p}, \text{'f}) \text{ continuation list}, (\text{'s}, \text{'f}) \text{ xstate}] \Rightarrow \text{bool}$   
 $(\langle \vdash -, - \Downarrow - \rangle \rightarrow [60, 20, 60] \ 89)$

**for**  $\Gamma :: (\text{'s}, \text{'p}, \text{'f}) \text{ body}$

**where**

$\text{Nil}: \Gamma \vdash [], [] \Downarrow s$

$| \text{ExitBlockNormal}: \Gamma \vdash \text{nrms}, \text{css} \Downarrow \text{Normal } s$   
 $\implies$

$$\Gamma \vdash [], (nrms, abrs) \# css \Downarrow Normal\ s$$

| *ExitBlockAbrupt*:  $\Gamma \vdash abrs, css \Downarrow Normal\ s$   
 $\implies$   
 $\Gamma \vdash [], (nrms, abrs) \# css \Downarrow Abrupt\ s$

| *ExitBlockFault*:  $\Gamma \vdash nrms, css \Downarrow Fault\ f$   
 $\implies$   
 $\Gamma \vdash [], (nrms, abrs) \# css \Downarrow Fault\ f$

| *ExitBlockStuck*:  $\Gamma \vdash nrms, css \Downarrow Stuck$   
 $\implies$   
 $\Gamma \vdash [], (nrms, abrs) \# css \Downarrow Stuck$

| *Cons*:  $[\Gamma \vdash c \Downarrow s; (\forall t. \Gamma \vdash \langle c, s \rangle \Rightarrow t \longrightarrow \Gamma \vdash cs, css \Downarrow t)]$   
 $\implies$   
 $\Gamma \vdash c \# cs, css \Downarrow s$

**inductive-cases** *terminatess-elim-cases* [cases set]:

$$\Gamma \vdash [], css \Downarrow t$$

$$\Gamma \vdash c \# cs, css \Downarrow t$$

**lemma** *terminatess-Fault*:  $\bigwedge cs. \Gamma \vdash cs, css \Downarrow Fault\ f$

**proof** (*induct css*)

**case** *Nil*

**show**  $\Gamma \vdash cs, [] \Downarrow Fault\ f$

**proof** (*induct cs*)

**case** *Nil* **show**  $\Gamma \vdash [], [] \Downarrow Fault\ f$  **by** (*rule terminatess.Nil*)

**next**

**case** (*Cons c cs*)

**thus** ?*case*

**by** (*auto intro: terminatess.intros terminatess.intros dest: Fault-end*)

**qed**

**next**

**case** (*Cons d css*)

**have** *hyp*:  $\bigwedge cs. \Gamma \vdash cs, css \Downarrow Fault\ f$  **by** *fact*

**obtain** *nrms abrs* **where** *d*:  $d = (nrms, abrs)$  **by** (*cases d*) *auto*

**have**  $\Gamma \vdash cs, (nrms, abrs) \# css \Downarrow Fault\ f$

**proof** (*induct cs*)

**case** *Nil*

**show**  $\Gamma \vdash [], (nrms, abrs) \# css \Downarrow Fault\ f$

**by** (*rule terminatess.ExitBlockFault*) (*rule hyp*)

**next**

**case** (*Cons c cs*)

**have** *hyp1*:  $\Gamma \vdash cs, (nrms, abrs) \# css \Downarrow Fault\ f$  **by** *fact*

**show**  $\Gamma \vdash c \# cs, (nrms, abrs) \# css \Downarrow Fault\ f$

**by** (*auto intro: hyp1 terminatess.Cons terminatess.intros dest: Fault-end*)

**qed**

```

  with d show ?case by simp
qed

lemma terminatess-Stuck:  $\bigwedge cs. \Gamma \vdash cs, css \Downarrow Stuck$ 
proof (induct css)
  case Nil
  show  $\Gamma \vdash cs, [] \Downarrow Stuck$ 
  proof (induct cs)
    case Nil show  $\Gamma \vdash [], [] \Downarrow Stuck$  by (rule terminatess.Nil)
  next
    case (Cons c cs)
    thus ?case
    by (auto intro: terminatess.intros terminates.intros dest: Stuck-end)
  qed
next
  case (Cons d css)
  have hyp:  $\bigwedge cs. \Gamma \vdash cs, css \Downarrow Stuck$  by fact
  obtain nrms abrs where d:  $d = (nrms, abrs)$  by (cases d) auto
  have  $\Gamma \vdash cs, (nrms, abrs) \# css \Downarrow Stuck$ 
  proof (induct cs)
    case Nil
    show  $\Gamma \vdash [], (nrms, abrs) \# css \Downarrow Stuck$ 
    by (rule terminatess.ExitBlockStuck) (rule hyp)
  next
    case (Cons c cs)
    have hyp1:  $\Gamma \vdash cs, (nrms, abrs) \# css \Downarrow Stuck$  by fact
    show  $\Gamma \vdash c \# cs, (nrms, abrs) \# css \Downarrow Stuck$ 
    by (auto intro: hyp1 terminatess.Cons terminates.intros dest: Stuck-end)
  qed
  with d show ?case by simp
qed

```

```

lemma Basic-terminates:  $\Gamma \vdash Basic\ f \Downarrow t$ 
  by (cases t) (auto intro: terminates.intros)

```

```

lemma step-preserves-terminations:
  assumes step:  $\Gamma \vdash (cs, css, s) \rightarrow (cs', css', t)$ 
  shows  $\Gamma \vdash cs, css \Downarrow s \implies \Gamma \vdash cs', css' \Downarrow t$ 
using step
proof (induct)
  case Skip thus ?case
  by (auto elim: terminates-Normal-elim-cases terminatess-elim-cases
    intro: exec.intros)
next
  case Guard thus ?case
  by (blast elim: terminatess-elim-cases terminates-Normal-elim-cases
    intro: terminatess.intros terminates.intros exec.intros)
next

```

```

case GuardFault thus ?case
  by (blast elim: terminatess-elim-cases terminates-Normal-elim-cases
        intro: terminatess.intros terminates.intros exec.intros)
next
case FaultProp show ?case by (rule terminatess-Fault)
next
case FaultPropBlock show ?case by (rule terminatess-Fault)
next
case AbruptProp thus ?case
  by (blast elim: terminatess-elim-cases
        intro: terminatess.intros)
next
case ExitBlockNormal thus ?case
  by (blast elim: terminatess-elim-cases
        intro: terminatess.intros )
next
case ExitBlockAbrupt thus ?case
  by (blast elim: terminatess-elim-cases
        intro: terminatess.intros )
next
case Basic thus ?case
  by (blast elim: terminatess-elim-cases terminates-Normal-elim-cases
        intro: terminatess.intros terminates.intros exec.intros)
next
case Spec thus ?case
  by (blast elim: terminatess-elim-cases terminates-Normal-elim-cases
        intro: terminatess.intros terminates.intros exec.intros)
next
case SpecStuck thus ?case
  by (blast elim: terminatess-elim-cases terminates-Normal-elim-cases
        intro: terminatess.intros terminates.intros exec.intros)
next
case Seq thus ?case
  by (blast elim: terminatess-elim-cases terminates-Normal-elim-cases
        intro: terminatess.intros terminates.intros exec.intros)
next
case CondTrue thus ?case
  by (blast elim: terminatess-elim-cases terminates-Normal-elim-cases
        intro: terminatess.intros terminates.intros exec.intros)
next
case CondFalse thus ?case
  by (blast elim: terminatess-elim-cases terminates-Normal-elim-cases
        intro: terminatess.intros terminates.intros exec.intros)
next
case WhileTrue thus ?case
  by (blast elim: terminatess-elim-cases terminates-Normal-elim-cases
        intro: terminatess.intros terminates.intros exec.intros)
next
case WhileFalse thus ?case

```

```

    by (blast elim: terminatess-elim-cases terminates-Normal-elim-cases
        intro: terminatess.intros terminates.intros exec.intros)
next
case (Call p bdy cs css s)
have bdy:  $\Gamma \vdash p = \text{Some } bdy$  by fact
from Call obtain
  term-body:  $\Gamma \vdash bdy \downarrow \text{Normal } s$  and
  term-rest:  $\forall t. \Gamma \vdash \langle \text{Call } p, \text{Normal } s \rangle \Rightarrow t \longrightarrow \Gamma \vdash cs, css \downarrow t$ 
  by (fastforce elim!: terminatess-elim-cases terminates-Normal-elim-cases)
show  $\Gamma \vdash [bdy], (cs, \text{Throw } \# \ cs) \# \ css \downarrow \text{Normal } s$ 
proof (rule terminatess.Cons [OF term-body], clarsimp)
  fix t
  assume exec-body:  $\Gamma \vdash \langle bdy, \text{Normal } s \rangle \Rightarrow t$ 
  show  $\Gamma \vdash [], (cs, \text{Throw } \# \ cs) \# \ css \downarrow t$ 
  proof (cases t)
    case (Normal t')
    with exec-body bdy
    have  $\Gamma \vdash \langle \text{Call } p, \text{Normal } s \rangle \Rightarrow \text{Normal } t'$ 
      by (auto intro: exec.intros)
    with term-rest have  $\Gamma \vdash cs, css \downarrow \text{Normal } t'$ 
      by iprover
    with Normal show ?thesis
      by (auto intro: terminatess.intros terminates.intros
          elim: exec-Normal-elim-cases)
  next
  case (Abrupt t')
  with exec-body bdy
  have  $\Gamma \vdash \langle \text{Call } p, \text{Normal } s \rangle \Rightarrow \text{Abrupt } t'$ 
    by (auto intro: exec.intros)
  with term-rest have  $\Gamma \vdash cs, css \downarrow \text{Abrupt } t'$ 
    by iprover
  with Abrupt show ?thesis
    by (fastforce intro: terminatess.intros terminates.intros
        elim: exec-Normal-elim-cases)
  next
  case Fault
  thus ?thesis
    by (iprover intro: terminatess-Fault)
  next
  case Stuck
  thus ?thesis
    by (iprover intro: terminatess-Stuck)
qed
qed
next
case CallUndefined thus ?case
  by (iprover intro: terminatess-Stuck)
next
case StuckProp show ?case by (rule terminatess-Stuck)

```



```

next
  case StuckPropBlock show ?case by (rule terminatess-Stuck)
next
  case DynCom thus ?case
    by (blast elim: terminatess-elim-cases terminates-Normal-elim-cases
      intro: terminatess.intros terminates.intros exec.intros)
next
  case Throw thus ?case
    by (blast elim: terminatess-elim-cases terminates-Normal-elim-cases
      intro: terminatess.intros terminates.intros exec.intros)
next
  case (Catch c1 c2 cs css s)
  then obtain
    term-c1:  $\Gamma \vdash c1 \downarrow \text{Normal } s$  and
    term-c2:  $\forall s'. \Gamma \vdash \langle c1, \text{Normal } s \rangle \Rightarrow \text{Abrupt } s' \longrightarrow \Gamma \vdash c2 \downarrow \text{Normal } s'$  and
    term-rest:  $\forall t. \Gamma \vdash \langle \text{Catch } c1 c2, \text{Normal } s \rangle \Rightarrow t \longrightarrow \Gamma \vdash cs, css \Downarrow t$ 
    by (clarsimp elim!: terminatess-elim-cases terminates-Normal-elim-cases)
  show  $\Gamma \vdash [c1], (cs, c2 \# cs) \# css \Downarrow \text{Normal } s$ 
  proof (rule terminatess.Cons [OF term-c1], clarsimp)
    fix t
    assume exec-c1:  $\Gamma \vdash \langle c1, \text{Normal } s \rangle \Rightarrow t$ 
    show  $\Gamma \vdash [], (cs, c2 \# cs) \# css \Downarrow t$ 
    proof (cases t)
      case (Normal t')
        with exec-c1 have  $\Gamma \vdash \langle \text{Catch } c1 c2, \text{Normal } s \rangle \Rightarrow t$ 
          by (auto intro: exec.intros)
        with term-rest have  $\Gamma \vdash cs, css \Downarrow t$ 
          by iprover
        with Normal show ?thesis
          by (iprover intro: terminatess.intros)
    next
      case (Abrupt t')
        with exec-c1 term-c2 have  $\Gamma \vdash c2 \downarrow \text{Normal } t'$ 
          by auto
        moreover
        {
          fix w
          assume exec-c2:  $\Gamma \vdash \langle c2, \text{Normal } t' \rangle \Rightarrow w$ 
          have  $\Gamma \vdash cs, css \Downarrow w$ 
          proof –
            from exec-c1 Abrupt exec-c2
            have  $\Gamma \vdash \langle \text{Catch } c1 c2, \text{Normal } s \rangle \Rightarrow w$ 
              by (auto intro: exec.intros)
            with term-rest show ?thesis by simp
          qed
        }
    ultimately
    show ?thesis using Abrupt
      by (auto intro: terminatess.intros)

```

```

next
  case Fault thus ?thesis
    by (iprover intro: terminatess-Fault)
next
  case Stuck thus ?thesis
    by (iprover intro: terminatess-Stuck)
qed
qed
qed

```

```

ML <
  ML-Thms.bind-thm (rtrancl-induct3, Split-Rule.split-rule @{context}
    (Rule-Insts.read-instantiate @{context}
      [(((a, 0), Position.none), (ax, ay, az)),
        (((b, 0), Position.none), (bx, by, bz))] []
      @{thm rtranclp-induct}));
  >

```

```

lemma steps-preserves-terminations:
  assumes steps:  $\Gamma \vdash (cs, css, s) \rightarrow^* (cs', css', t)$ 
  shows  $\Gamma \vdash cs, css \downarrow s \implies \Gamma \vdash cs', css' \downarrow t$ 
using steps
proof (induct rule: rtrancl-induct3 [consumes 1])
  assume  $\Gamma \vdash cs, css \downarrow s$  then show  $\Gamma \vdash cs, css \downarrow s$ .
next
  fix cs'' css'' w cs' css' t
  assume  $\Gamma \vdash (cs'', css'', w) \rightarrow (cs', css', t)$   $\Gamma \vdash cs, css \downarrow s \implies \Gamma \vdash cs'', css'' \downarrow w$ 
     $\Gamma \vdash cs, css \downarrow s$ 
  then show  $\Gamma \vdash cs', css' \downarrow t$ 
  by (blast dest: step-preserves-terminations)
qed

```

```

theorem steps-preserves-termination:
  assumes steps:  $\Gamma \vdash ([c], [], s) \rightarrow^* (c' \# cs', css', t)$ 
  assumes term-c:  $\Gamma \vdash c \downarrow s$ 
  shows  $\Gamma \vdash c' \downarrow t$ 
proof –
  from term-c have  $\Gamma \vdash [c], [] \downarrow s$ 
  by (auto intro: terminatess.intros)
  from steps this
  have  $\Gamma \vdash c' \# cs', css' \downarrow t$ 
  by (rule steps-preserves-terminations)
  thus  $\Gamma \vdash c' \downarrow t$ 
  by (auto elim: terminatess-elim-cases)
qed

```

**lemma** *renumber'*:  
**assumes**  $f: \forall i. (a, f\ i) \in r^* \wedge (f\ i, f(Suc\ i)) \in r$   
**assumes**  $a\text{-}b: (a, b) \in r^*$   
**shows**  $b = f\ 0 \implies (\exists f. f\ 0 = a \wedge (\forall i. (f\ i, f(Suc\ i)) \in r))$   
**using**  $a\text{-}b$   
**proof** (*induct rule: converse-rtrancl-induct [consumes 1]*)  
**assume**  $b = f\ 0$   
**with**  $f$  **show**  $\exists f. f\ 0 = b \wedge (\forall i. (f\ i, f(Suc\ i)) \in r)$   
**by** *blast*  
**next**  
**fix**  $a\ z$   
**assume**  $a\text{-}z: (a, z) \in r$  **and**  $(z, b) \in r^*$   
**assume**  $b = f\ 0 \implies \exists f. f\ 0 = z \wedge (\forall i. (f\ i, f(Suc\ i)) \in r)$   
 $b = f\ 0$   
**then obtain**  $f$  **where**  $f\ 0 = z$  **and**  $seq: \forall i. (f\ i, f(Suc\ i)) \in r$   
**by** *iprover*  
{  
**fix**  $i$  **have**  $((\lambda i. case\ i\ of\ 0 \Rightarrow a \mid Suc\ i \Rightarrow f\ i)\ i, f\ i) \in r$   
**using**  $seq\ a\text{-}z\ f\ 0$   
**by** (*cases i*) *auto*  
}  
**then**  
**show**  $\exists f. f\ 0 = a \wedge (\forall i. (f\ i, f(Suc\ i)) \in r)$   
**by** - (*rule exI [where x= $\lambda i. case\ i\ of\ 0 \Rightarrow a \mid Suc\ i \Rightarrow f\ i$ ], simp*)  
**qed**

**lemma** *renumber*:  
 $\forall i. (a, f\ i) \in r^* \wedge (f\ i, f(Suc\ i)) \in r$   
 $\implies \exists f. f\ 0 = a \wedge (\forall i. (f\ i, f(Suc\ i)) \in r)$   
**by** (*blast dest:renumber'*)

**lemma** *not-inf-Fault'*:  
**assumes** *enum-step*:  $\forall i. \Gamma \vdash f\ i \rightarrow f(Suc\ i)$   
**shows**  $\bigwedge k\ cs. f\ k = (cs, css, Fault\ m) \implies False$   
**proof** (*induct css*)  
**case** *Nil*  
**have**  $f\ k: f\ k = (cs, [], Fault\ m)$  **by** *fact*  
**have**  $\bigwedge k. f\ k = (cs, [], Fault\ m) \implies False$   
**proof** (*induct cs*)  
**case** *Nil*  
**have**  $f\ k = ([], [], Fault\ m)$  **by** *fact*  
**moreover**  
**from** *enum-step* **have**  $\Gamma \vdash f\ k \rightarrow f(Suc\ k)$ .  
**ultimately show** *False*  
**by** (*fastforce elim: step-elim-cases*)  
**next**  
**case** (*Cons c cs*)

```

have  $fk: f k = (c \# cs, [], Fault\ m)$  by fact
from enum-step have  $\Gamma \vdash f k \rightarrow f (Suc\ k)$ ..
with  $fk$  have  $f (Suc\ k) = (cs, [], Fault\ m)$ 
  by (fastforce elim: step-elim-cases)
with enum-step Cons.hyps
show False
  by blast
qed
from this f-k show False by blast
next
case (Cons ds css)
then obtain  $nrms\ ahrs$  where  $ds = (nrms, ahrs)$  by (cases ds) auto
have  $hyp: \bigwedge k\ cs. f k = (cs, css, Fault\ m) \implies False$  by fact
have  $\bigwedge k. f k = (cs, (nrms, ahrs) \# css, Fault\ m) \implies False$ 
proof (induct cs)
  case Nil
  have  $fk: f k = ([], (nrms, ahrs) \# css, Fault\ m)$  by fact
  from enum-step have  $\Gamma \vdash f k \rightarrow f (Suc\ k)$ ..
  with  $fk$  have  $f (Suc\ k) = (nrms, css, Fault\ m)$ 
    by (fastforce elim: step-elim-cases)
  thus ?case
    by (rule hyp)
  next
  case (Cons c cs)
  have  $fk: f k = (c \# cs, (nrms, ahrs) \# css, Fault\ m)$  by fact
  have  $hyp1: \bigwedge k. f k = (cs, (nrms, ahrs) \# css, Fault\ m) \implies False$  by fact
  from enum-step have  $\Gamma \vdash f k \rightarrow f (Suc\ k)$ ..
  with  $fk$  have  $f (Suc\ k) = (cs, (nrms, ahrs) \# css, Fault\ m)$ 
    by (fastforce elim: step-elim-cases)
  thus ?case
    by (rule hyp1)
  qed
with  $ds$  Cons.prems show False by auto
qed

lemma not-inf-Fault:
   $\neg inf\ \Gamma\ cs\ css\ (Fault\ m)$ 
apply (rule not-infI)
apply (rule-tac f=f in not-inf-Fault')
by auto

lemma not-inf-Stuck':
  assumes enum-step:  $\forall i. \Gamma \vdash f i \rightarrow f (Suc\ i)$ 
  shows  $\bigwedge k\ cs. f k = (cs, css, Stuck) \implies False$ 
proof (induct css)
  case Nil
  have  $f-k: f k = (cs, [], Stuck)$  by fact
  have  $\bigwedge k. f k = (cs, [], Stuck) \implies False$ 
proof (induct cs)

```

```

    case Nil
    have f k = ([], [], Stuck) by fact
    moreover
    from enum-step have  $\Gamma \vdash f k \rightarrow f (Suc k)$ ..
    ultimately show False
      by (fastforce elim: step-elim-cases)
  next
  case (Cons c cs)
  have fk: f k = (c # cs, [], Stuck) by fact
  from enum-step have  $\Gamma \vdash f k \rightarrow f (Suc k)$ ..
  with fk have f (Suc k) = (cs, [], Stuck)
    by (fastforce elim: step-elim-cases)
  with enum-step Cons.hyps
  show False
    by blast
  qed
  from this f-k show False .
next
case (Cons ds css)
then obtain nrms abrs where ds: ds=(nrms,abrs) by (cases ds) auto
have hyp:  $\bigwedge k cs. f k = (cs, css, Stuck) \implies False$  by fact
have  $\bigwedge k. f k = (cs, (nrms, abrs) \# css, Stuck) \implies False$ 
proof (induct cs)
  case Nil
  have fk: f k = ([], (nrms, abrs) # css, Stuck) by fact
  from enum-step have  $\Gamma \vdash f k \rightarrow f (Suc k)$ ..
  with fk have f (Suc k) = (nrms, css, Stuck)
    by (fastforce elim: step-elim-cases)
  thus ?case
    by (rule hyp)
next
case (Cons c cs)
have fk: f k = (c # cs, (nrms, abrs) # css, Stuck) by fact
have hyp1:  $\bigwedge k. f k = (cs, (nrms, abrs) \# css, Stuck) \implies False$  by fact
from enum-step have  $\Gamma \vdash f k \rightarrow f (Suc k)$ ..
with fk have f (Suc k) = (cs, (nrms, abrs) # css, Stuck)
  by (fastforce elim: step-elim-cases)
thus ?case
  by (rule hyp1)
qed
with ds Cons.prem show False by auto
qed

lemma not-inf-Stuck:
   $\neg inf \Gamma cs css Stuck$ 
apply (rule not-infI)
apply (rule-tac f=f in not-inf-Stuck')
by auto

```

**lemma** *last-butlast-app*:  
**assumes** *butlast*:  $butlast\ as = xs\ @\ butlast\ bs$   
**assumes** *not-Nil*:  $bs \neq []\ as \neq []$   
**assumes** *last*:  $fst\ (last\ as) = fst\ (last\ bs)\ snd\ (last\ as) = snd\ (last\ bs)$   
**shows**  $as = xs\ @\ bs$   
**proof** –  
**from** *last* **have**  $last\ as = last\ bs$   
**by** (*cases last as, cases last bs*) *simp*  
**moreover**  
**from** *not-Nil* **have**  $as = butlast\ as\ @\ [last\ as]\ bs = butlast\ bs\ @\ [last\ bs]$   
**by** *auto*  
**ultimately show** *?thesis*  
**using** *butlast*  
**by** *simp*  
**qed**

**lemma** *last-butlast-tl*:  
**assumes** *butlast*:  $butlast\ bs = x\ \#\ butlast\ as$   
**assumes** *not-Nil*:  $bs \neq []\ as \neq []$   
**assumes** *last*:  $fst\ (last\ as) = fst\ (last\ bs)\ snd\ (last\ as) = snd\ (last\ bs)$   
**shows**  $as = tl\ bs$   
**proof** –  
**from** *last* **have**  $last\ as = last\ bs$   
**by** (*cases last as, cases last bs*) *simp*  
**moreover**  
**from** *not-Nil* **have**  $as = butlast\ as\ @\ [last\ as]\ bs = butlast\ bs\ @\ [last\ bs]$   
**by** *auto*  
**ultimately show** *?thesis*  
**using** *butlast*  
**by** *simp*  
**qed**

**locale** *inf* =  
**fixes** *CS*::  $(\prime s, \prime p, \prime f)\ config \Rightarrow (\prime s, \prime p, \prime f)\ com\ list$   
**and** *CSS*::  $(\prime s, \prime p, \prime f)\ config \Rightarrow (\prime s, \prime p, \prime f)\ continuation\ list$   
**and** *S*::  $(\prime s, \prime p, \prime f)\ config \Rightarrow (\prime s, \prime f)\ xstate$   
**defines** *CS-def* :  $CS \equiv fst$   
**defines** *CSS-def* :  $CSS \equiv \lambda c. fst\ (snd\ c)$   
**defines** *S-def*:  $S \equiv \lambda c. snd\ (snd\ c)$

**lemma** (**in** *inf*) *steps-hd-drop-suffix*:  
**assumes** *f-0*:  $f\ 0 = (c\ \#\ cs, css, s)$   
**assumes** *f-step*:  $\forall i. \Gamma \vdash f(i) \rightarrow f(Suc\ i)$   
**assumes** *not-finished*:  $\forall i < k. \neg (CS\ (f\ i) = cs \wedge CSS\ (f\ i) = css)$   
**assumes** *simul*:  $\forall i \leq k.$   
**(if** *pcss*  $i = []$  **then**  $CSS\ (f\ i) = css \wedge CS\ (f\ i) = pcs\ i\ @\ cs$   
**else**  $CS\ (f\ i) = pcs\ i \wedge$   
 $CSS\ (f\ i) = butlast\ (pcss\ i)\ @$

$$[(fst (last (pcss i))@cs, (snd (last (pcss i)))@cs)]@css$$

**defines**  $p \equiv \lambda i. (pcs\ i, pcss\ i, S\ (f\ i))$   
**shows**  $\forall i < k. \Gamma \vdash p\ i \rightarrow p\ (Suc\ i)$   
**using** *not-finished simul*  
**proof** (*induct k*)  
  **case** *0*  
  **thus** *?case by simp*  
**next**  
  **case** (*Suc k*)  
  **have** *simul*:  $\forall i \leq Suc\ k.$   
    (*if*  $pcss\ i = []$  *then*  $CSS\ (f\ i) = css \wedge CS\ (f\ i) = pcs\ i@cs$   
    *else*  $CS\ (f\ i) = pcs\ i \wedge$   
     $CSS\ (f\ i) = butlast\ (pcss\ i)@$   
     $[(fst (last (pcss i))@cs, (snd (last (pcss i)))@cs)]@$   
     $css)$  **by fact**  
  **have** *not-finished'*:  $\forall i < Suc\ k. \neg (CS\ (f\ i) = cs \wedge CSS\ (f\ i) = css)$  **by fact**  
  **with** *simul*  
  **have** *not-finished*:  $\forall i < Suc\ k. \neg (pcs\ i = [] \wedge pcss\ i = [])$   
  **by** (*auto simp add: CS-def CSS-def S-def split: if-split-asm*)  
  **show** *?case*  
  **proof** (*clarify*)  
    **fix** *i*  
    **assume** *i-le-Suc-k*:  $i < Suc\ k$   
    **show**  $\Gamma \vdash p\ i \rightarrow p\ (Suc\ i)$   
    **proof** (*cases i < k*)  
      **case** *True*  
      **with** *not-finished'* *simul Suc.hyps*  
      **show** *?thesis*  
      **by** *auto*  
    **next**  
    **case** *False*  
    **with** *i-le-Suc-k*  
    **have** *eq-i-k*:  $i = k$   
    **by** *simp*  
    **show**  $\Gamma \vdash p\ i \rightarrow p\ (Suc\ i)$   
    **proof** –  
    **obtain**  $cs'\ css'\ t'$  **where**  
       $f\ Suc\ i: f\ (Suc\ i) = (cs', css', t')$   
      **by** (*cases f (Suc i)*)  
    **obtain**  $cs'' css'' t''$  **where**  
       $f\ i: f\ i = (cs'', css'', t'')$   
      **by** (*cases f i*)  
    **from** *not-finished eq-i-k*  
    **have** *pcs-pcss-not-Nil*:  $\neg (pcs\ i = [] \wedge pcss\ i = [])$   
    **by** *auto*  
    **from** *simul [rule-format, of i] i-le-Suc-k f-i*  
    **have** *pcs-pcss-i*:  
      *if*  $pcss\ i = []$  *then*  $css'' = css \wedge cs'' = pcs\ i@cs$

*else*  $cs'' = pcs\ i \wedge$   
 $css'' = butlast\ (pcss\ i) @$   
 $[(fst\ (last\ (pcss\ i)) @ cs, (snd\ (last\ (pcss\ i)) @ cs)) @$   
 $css]$   
**by** (*simp add: CS-def CSS-def S-def cong: if-cong*)  
**from** *simul* [rule-format, of *Suc i*] *i-le-Suc-k f-Suc-i*  
**have** *pcs-pcss-Suc-i*:  
*if*  $pcss\ (Suc\ i) = []$  *then*  $css' = css \wedge cs' = pcs\ (Suc\ i) @ cs$   
*else*  $cs' = pcs\ (Suc\ i) \wedge$   
 $css' = butlast\ (pcss\ (Suc\ i)) @$   
 $[(fst\ (last\ (pcss\ (Suc\ i))) @ cs, snd\ (last\ (pcss\ (Suc\ i))) @ cs] @$   
 $css]$   
**by** (*simp add: CS-def CSS-def S-def cong: if-cong*)  
**show** *?thesis*  
**proof** (*cases pcss i = []*)  
**case** *True*  
**note** *pcss-Nil = this*  
**with** *pcs-pcss-i pcs-pcss-not-Nil* **obtain**  $p\ ps$  **where**  
 $pcs\text{-}i: pcs\ i = p \# ps$  **and**  
 $css'': css'' = css$  **and**  
 $cs'': cs'' = (p \# ps) @ cs$   
**by** (*auto simp add: neg-Nil-conv*)  
**with** *f-i* **have**  $f\ i = (p \# (ps @ cs), css, t')$   
**by** *simp*  
**with** *f-Suc-i f-step* [rule-format, of *i*]  
**have** *step-css*:  $\Gamma \vdash (p \# (ps @ cs), css, t') \rightarrow (cs', css', t')$   
**by** *simp*  
**from** *step-Cons'* [*OF this*, of  $p\ ps @ cs$ ]  
**obtain**  $css'''$  **where**  
 $css''': css' = css''' @ css$   
*if*  $css''' = []$  *then*  $\exists p. cs' = p @ ps @ cs$   
*else*  $(\exists pnorm\ pabr. css''' = [(pnorm @ ps @ cs, pabr @ ps @ cs)])$   
**by** *auto*  
**show** *?thesis*  
**proof** (*cases css''' = []*)  
**case** *True*  
**with**  $css'''$   
**obtain**  $p'$  **where**  
 $css': css' = css$  **and**  
 $cs': cs' = p' @ ps @ cs$   
**by** *auto*  
  
**from**  $css'\ cs'$  *step-css*  
**have** *step*:  $\Gamma \vdash (p \# (ps @ cs), css, t') \rightarrow (p' @ ps @ cs, css, t')$   
**by** *simp*  
**hence**  $\Gamma \vdash ((p \# ps) @ cs, css, t') \rightarrow ((p' @ ps) @ cs, css, t')$   
**by** *simp*  
**from** *drop-suffix-css-step'* [*OF drop-suffix-same-css-step* [*OF this*],  
**where**  $xs = css$  **and**  $css = []$  **and**  $css' = []$



```

have  $\Gamma \vdash (p\#ps, [], t'') \rightarrow (p'\@ps, [], t')$ 
  by simp
moreover
from css' cs' pcs-pcss-Suc-i
obtain  $pcs (Suc\ i) = p'\@ps$  and  $pcss (Suc\ i) = []$ 
  by (simp split: if-split-asm)
ultimately show ?thesis
  using pcs-i pcss-Nil f-i f-Suc-i
  by (simp add: CS-def CSS-def S-def p-def)
next
case False
with css'''
obtain pnorm pabr where
   $css' : css' = css'''@css$ 
   $css''' = [(pnorm\ @\ ps\ @\ cs, pabr\ @\ ps\ @\ cs)]$ 
  by auto
with css''' step-css
have  $\Gamma \vdash (p\#ps@cs, css, t'') \rightarrow (cs', [(pnorm@ps@cs, pabr@ps@cs)]@css, t')$ 
  by simp
then
have  $\Gamma \vdash (p\#ps, css, t'') \rightarrow (cs', [(pnorm@ps, pabr@ps)]\ @\ css, t')$ 
  by (rule drop-suffix-hd-css-step)
from drop-suffix-css-step' [OF this,
  where  $css = []$  and  $xs = css$  and  $css' = [(pnorm@ps, pabr@ps)]$ ]
have  $\Gamma \vdash (p\#ps, [], t'') \rightarrow (cs', [(pnorm@ps, pabr@ps)], t')$ 
  by simp
moreover
from css' pcs-pcss-Suc-i
obtain  $pcs (Suc\ i) = cs' pcss (Suc\ i) = [(pnorm@ps, pabr@ps)]$ 
  apply (cases pcss (Suc\ i))
  apply (auto split: if-split-asm)
done
ultimately show ?thesis
  using pcs-i pcss-Nil f-i f-Suc-i
  by (simp add: p-def CS-def CSS-def S-def)
qed
next
case False
note pcss-i-not-Nil = this
with pcs-pcss-i obtain
   $cs'' : cs'' = pcs\ i$  and
   $css'' : css'' = butlast\ (pcss\ i)\ @\$ 
     $[(fst\ (last\ (pcss\ i))\ @\ cs, (snd\ (last\ (pcss\ i))\ @\ cs)]\ @\$ 
     $css$ 
  by auto
from f-Suc-i f-i f-step [rule-format, of i]
have step-i-full:  $\Gamma \vdash (cs'', css'', t'') \rightarrow (cs', css', t')$ 
  by simp
show ?thesis

```

```

proof (cases cs'')
  case (Cons c' cs)
  with step-Cons' [OF step-i-full]
  obtain css''' where css': css' = css'''@css''
    by auto
  with step-i-full
  have  $\Gamma \vdash (cs'', css'', t'') \rightarrow (cs', css'''@css'', t')$ 
    by simp
  from Cons-change-css-step [OF this, where xss=pcss i] Cons cs''
  have  $\Gamma \vdash (pcs\ i, pcss\ i, t'') \rightarrow (cs', css'''@pcss\ i, t')$ 
    by simp
  moreover
  from cs'' css'' css' False pcs-pcss-Suc-i
  obtain pcs (Suc i) = cs' pcss (Suc i) = css'''@pcss i
    apply (auto split: if-split-asm)
    apply (drule (4) last-butlast-app)
    by simp
  ultimately show ?thesis
    using f-i f-Suc-i
    by (simp add: p-def CS-def CSS-def S-def)
next
  case Nil
  note cs''-Nil = this
  show ?thesis
  proof (cases butlast (pcss i))
    case (Cons bpcs bpcss)
    with cs''-Nil step-i-full css''
    have *:  $\Gamma \vdash ([, [hd\ css'']@tl\ css'', t'') \rightarrow (cs', css', t')$ 
      by simp
    moreover
    from step-Nil [OF *]
    have css': css' = tl css''
      by simp
    ultimately have
      step-i-full:  $\Gamma \vdash ([, [hd\ css'']@tl\ css'', t'') \rightarrow (cs', tl\ css'', t')$ 
      by simp
    from css'' Cons pcss-i-not-Nil
    have hd css'' = hd (pcss i)
      by (auto simp add: neq-Nil-conv split: if-split-asm)
    with cs'' cs''-Nil
      Nil-change-css-step [where ass=[hd css''] and
        css=tl css'' and ass'=[] and
        xss=tl (pcss i), simplified, OF step-i-full [simplified]]
    have  $\Gamma \vdash (pcs\ i, [hd\ (pcss\ i)]@tl\ (pcss\ i), t'') \rightarrow (cs', tl\ (pcss\ i), t')$ 
      by simp
    with pcss-i-not-Nil
    have  $\Gamma \vdash (pcs\ i, pcss\ i, t'') \rightarrow (cs', tl\ (pcss\ i), t')$ 
      by simp
    moreover

```

```

from css' css'' cs''-Nil Cons pcss-i-not-Nil pcs-pcss-Suc-i
obtain pcs (Suc i) = cs' pcss (Suc i) = tl (pcss i)
  apply (clarsimp split: if-split-asm)
  apply (drule (4) last-butlast-tl)
  by simp
ultimately show ?thesis
  using f-i f-Suc-i
  by (simp add: p-def CS-def CSS-def S-def)
next
case Nil
with css'' pcss-i-not-Nil
obtain pnorm pabr
  where css''': css'' = [(pnorm@cs,pabr@cs)]@css and
  pcss-i: pcss i = [(pnorm,pabr)]
  by (force simp add: neg-Nil-conv split: if-split-asm)
with cs''-Nil step-i-full
have  $\Gamma \vdash ([, [(pnorm@cs,pabr@cs)]@css, t''] \rightarrow (cs', css', t'))$ 
  by simp
from step-Nil [OF this]
obtain
  css': css' = css and
  cs': (case t'' of
    Abrupt s'  $\Rightarrow cs' = pabr @ cs \wedge t' = Normal s'$ 
    | -  $\Rightarrow cs' = pnorm @ cs \wedge t' = t''$ )
  by (simp cong: xstate.case-cong)
let ?pcs-Suc-i = (case t'' of Abrupt s'  $\Rightarrow pabr$  | -  $\Rightarrow pnorm$ )
from cs'
have  $\Gamma \vdash ([, [(pnorm,pabr)], t''] \rightarrow (?pcs-Suc-i, [], t'))$ 
  by (auto intro: step.intros split: xstate.splits)
moreover
from css'' css' cs' pcss-i pcs-pcss-Suc-i
obtain pcs (Suc i) = ?pcs-Suc-i pcss (Suc i) = []
  by (simp split: if-split-asm xstate.splits)
ultimately
show ?thesis
  using pcss-i cs'' cs''-Nil f-i f-Suc-i
  by (simp add: p-def CS-def CSS-def S-def)
qed
qed
qed
qed
qed
qed
qed

```

**lemma** *k-steps-to-rtrancl:*  
**assumes** *steps:  $\forall i < k. \Gamma \vdash p i \rightarrow p (Suc i)$*   
**shows**  $\Gamma \vdash p 0 \rightarrow^* p k$   
**using** *steps*

**proof** (*induct k*)  
**case 0 thus** *?case* **by** *auto*  
**next**  
**case** (*Suc k*)  
**have**  $\forall i < \text{Suc } k. \Gamma \vdash p \ i \rightarrow p \ (\text{Suc } i)$  **by** *fact*  
**then obtain**  
*step-le-k*:  $\forall i < k. \Gamma \vdash p \ i \rightarrow p \ (\text{Suc } i)$  **and** *step-k*:  $\Gamma \vdash p \ k \rightarrow p \ (\text{Suc } k)$   
**by** *auto*  
**from** *Suc.hyps* [*OF step-le-k*]  
**have**  $\Gamma \vdash p \ 0 \rightarrow^* p \ k$ .  
**also note** *step-k*  
**finally show** *?case* .  
**qed**

**lemma** (*in inf*) *steps-hd-drop-suffix-finite*:  
**assumes** *f-0*:  $f \ 0 = (c \# cs, css, s)$   
**assumes** *f-step*:  $\forall i. \Gamma \vdash f(i) \rightarrow f(\text{Suc } i)$   
**assumes** *not-finished*:  $\forall i < k. \neg (CS \ (f \ i) = cs \wedge CSS \ (f \ i) = css)$   
**assumes** *simul*:  $\forall i \leq k$ .  

$$\begin{aligned} & \text{(if } pcss \ i = [] \text{ then } CSS \ (f \ i) = css \wedge CS \ (f \ i) = pcs \ i @ cs \\ & \quad \text{else } CS \ (f \ i) = pcs \ i \wedge \\ & \quad \quad CSS \ (f \ i) = \text{butlast } (pcss \ i) @ \\ & \quad \quad \quad [(fst \ (last \ (pcss \ i))) @ cs, (snd \ (last \ (pcss \ i))) @ cs] @ \\ & \quad \quad \quad \quad css) \end{aligned}$$
  
**shows**  $\Gamma \vdash ([c], [], s) \rightarrow^* (pcs \ k, pcss \ k, S \ (f \ k))$   
**proof** –  
**from** *steps-hd-drop-suffix* [*OF f-0 f-step not-finished simul*]  
**have**  $\forall i < k. \Gamma \vdash (pcs \ i, pcss \ i, S \ (f \ i)) \rightarrow$   
 $(pcs \ (\text{Suc } i), pcss \ (\text{Suc } i), S \ (f \ (\text{Suc } i)))$ .  
**from** *k-steps-to-rtrancl* [*OF this*]  
**have**  $\Gamma \vdash (pcs \ 0, pcss \ 0, S \ (f \ 0)) \rightarrow^* (pcs \ k, pcss \ k, S \ (f \ k))$ .  
**moreover from** *f-0 simul* [*rule-format, of 0*]  
**have**  $(pcs \ 0, pcss \ 0, S \ (f \ 0)) = ([c], [], s)$   
**by** (*auto split: if-split-asm simp add: CS-def CSS-def S-def*)  
**ultimately show** *?thesis* **by** *simp*  
**qed**

**lemma** (*in inf*) *steps-hd-drop-suffix-infinite*:  
**assumes** *f-0*:  $f \ 0 = (c \# cs, css, s)$   
**assumes** *f-step*:  $\forall i. \Gamma \vdash f(i) \rightarrow f(\text{Suc } i)$   
**assumes** *not-finished*:  $\forall i. \neg (CS \ (f \ i) = cs \wedge CSS \ (f \ i) = css)$   
**assumes** *simul*:  $\forall i$ .  

$$\begin{aligned} & \text{(if } pcss \ i = [] \text{ then } CSS \ (f \ i) = css \wedge CS \ (f \ i) = pcs \ i @ cs \\ & \quad \text{else } CS \ (f \ i) = pcs \ i \wedge \\ & \quad \quad CSS \ (f \ i) = \text{butlast } (pcss \ i) @ \\ & \quad \quad \quad [(fst \ (last \ (pcss \ i))) @ cs, (snd \ (last \ (pcss \ i))) @ cs] @ \end{aligned}$$

*css*)

**defines**  $p \equiv \lambda i. (pcs\ i, pcss\ i, S\ (f\ i))$

**shows**  $\Gamma \vdash p\ i \rightarrow p\ (Suc\ i)$

**proof** –

**from** *steps-hd-drop-suffix* [*OF* *f-0* *f-step*, *of* *Suc i pcss pcs*] *not-finished simul*

**show** *?thesis*

**by** (*auto simp add: p-def*)

**qed**

**lemma** (*in inf*) *steps-hd-progress*:

**assumes** *f-0*:  $f\ 0 = (c\ \#\ cs, css, s)$

**assumes** *f-step*:  $\forall i. \Gamma \vdash f(i) \rightarrow f(Suc\ i)$

**assumes** *c-unfinished*:  $\forall i < k. \neg (CS\ (f\ i) = cs \wedge CSS\ (f\ i) = css)$

**shows**  $\forall i \leq k. (\exists pcs\ pcss.$

*(if*  $pcss = []$  *then*  $CSS\ (f\ i) = css \wedge CS\ (f\ i) = pcs@cs$

*else*  $CS\ (f\ i) = pcs \wedge$

$CSS\ (f\ i) = butlast\ pcss@$

$[(fst\ (last\ pcss)@cs, (snd\ (last\ pcss))@cs)]@$

$css)$ )

**using** *c-unfinished*

**proof** (*induct k*)

**case** *0*

**with** *f-0* **show** *?case*

**by** (*simp add: CSS-def CS-def*)

**next**

**case** (*Suc k*)

**have** *c-unfinished*:  $\forall i < Suc\ k. \neg (CS\ (f\ i) = cs \wedge CSS\ (f\ i) = css)$  **by** *fact*

**hence** *c-unfinished'*:  $\forall i < k. \neg (CS\ (f\ i) = cs \wedge CSS\ (f\ i) = css)$  **by** *simp*

**show** *?case*

**proof** (*clarify*)

**fix** *i*

**assume** *i-le-Suc-k*:  $i \leq Suc\ k$

**show**  $\exists pcs\ pcss.$

*(if*  $pcss = []$  *then*  $CSS\ (f\ i) = css \wedge CS\ (f\ i) = pcs@cs$

*else*  $CS\ (f\ i) = pcs \wedge$

$CSS\ (f\ i) = butlast\ pcss@$

$[(fst\ (last\ pcss)@cs, (snd\ (last\ pcss))@cs)]@$

$css)$ )

**proof** (*cases i < Suc k*)

**case** *True*

**with** *Suc.hyps* [*OF* *c-unfinished'*, *rule-format*, *of i*] *c-unfinished*

**show** *?thesis*

**by** *auto*

**next**

**case** *False*

**with** *i-le-Suc-k* **have** *eq-i-Suc-k*:  $i = Suc\ k$

**by** *auto*

**obtain**  $cs'\ css'\ t'$  **where**

*f-Suc-k*:  $f\ (Suc\ k) = (cs', css', t')$

```

  by (cases f (Suc k))
obtain cs'' css'' t'' where
  f-k: f k = (cs'',css'',t'')
  by (cases f k)
with Suc.hyps [OF c-unfinished',rule-format, of k]
obtain pcs pcss where
  pcs-pcss-k:
  if pcss = [] then css'' = css ∧ cs'' = pcs @ cs
  else cs'' = pcs ∧
    css'' = butlast pcss @
      [(fst (last pcss) @ cs, snd (last pcss) @ cs)] @
      css
  by (auto simp add: CSS-def CS-def cong: if-cong)
from c-unfinished [rule-format, of k] f-k pcs-pcss-k
have pcs-pcss-empty: ¬ (pcs = [] ∧ pcss = [])
  by (auto simp add: CS-def CSS-def S-def split: if-split-asm)
show ?thesis
proof (cases pcss = [])
  case True
  note pcss-Nil = this
  with pcs-pcss-k pcs-pcss-empty obtain p ps where
  pcs-i: pcs = p#ps and
  css'': css''=css and
  cs'': cs''=(p#ps)@cs
  by (cases pcs) auto
  with f-k have f k = (p#(ps@cs),css,t'')
  by simp
  with f-Suc-k f-step [rule-format, of k]
  have step-css: Γ ⊢ (p#(ps@cs),css,t'') → (cs',css',t')
  by simp
  from step-Cons' [OF this, of p ps@cs]
  obtain css''' where
  css''': css' = css''' @ css
  if css''' = [] then ∃ p. cs' = p @ ps @ cs
  else (∃ pnorm pabr. css'''=[(pnorm @ ps @ cs,pabr @ ps @ cs)])
  by auto
  show ?thesis
  proof (cases css''' = [])
  case True
  with css'''
  obtain p' where
  css': css' = css and
  cs': cs' = p' @ ps @ cs
  by auto
  from css' cs' f-Suc-k
  show ?thesis
  apply (rule-tac x=p'@ps in exI)
  apply (rule-tac x=[] in exI)
  apply (simp add: CSS-def CS-def eq-i-Suc-k)

```

```

done
next
case False
with css'''
obtain pnorm pabr where
  css': css' = css''' @ css
  css''':  $[(pnorm @ ps @ cs, pabr @ ps @ cs)]$ 
  by auto
with f-Suc-k eq-i-Suc-k
show ?thesis
  apply (rule-tac x = cs' in exI)
  apply (rule-tac x = [(pnorm @ ps, pabr @ ps)] in exI)
  by (simp add: CSS-def CS-def)
qed
next
case False
note pcss-k-not-Nil = this
with pcs-pcss-k obtain
  cs'': cs'' = pcs and
  css'': css'' = butlast pcss @
            $[(fst (last pcss) @ cs, (snd (last pcss)) @ cs)] @$ 
           css
  by auto
from f-Suc-k f-k f-step [rule-format, of k]
have step-i-full:  $\Gamma \vdash (cs'', css'', t') \rightarrow (cs', css', t')$ 
  by simp
show ?thesis
proof (cases cs'')
case (Cons c' cs)
with step-Cons' [OF step-i-full]
obtain css''' where css': css' = css''' @ css''
  by auto
with cs'' css'' f-Suc-k eq-i-Suc-k pcss-k-not-Nil
show ?thesis
  apply (rule-tac x = cs' in exI)
  apply (rule-tac x = css''' @ pcss in exI)
  by (clarsimp simp add: CSS-def CS-def butlast-append)
next
case Nil
note cs''-Nil = this
show ?thesis
proof (cases butlast pcss)
case (Cons bpcs bpcss)
with cs''-Nil step-i-full css''
have *:  $\Gamma \vdash ([, [hd\ css''] @ tl\ css'', t']) \rightarrow (cs', css', t')$ 
  by simp
moreover
from step-Nil [OF *]
obtain css': css' = tl css'' and

```

```

      cs': cs' = (case t'' of Abrupt s' ⇒ snd (hd css'')
                | - ⇒ fst (hd css''))
    by (auto split: xstate.splits)
  from css'' Cons pcss-k-not-Nil
  have hd css'' = hd pcss
    by (auto simp add: neq-Nil-conv split: if-split-asm)
  with css' cs' css'' cs''-Nil Cons pcss-k-not-Nil f-Suc-k eq-i-Suc-k
  show ?thesis
    apply (rule-tac x=cs' in exI)
    apply (rule-tac x=tl pcss in exI)
    apply (clarsimp split: xstate.splits
           simp add: CS-def CSS-def neq-Nil-conv split: if-split-asm)
  done
next
case Nil
with css'' pcss-k-not-Nil
obtain pnorm pabr
  where css'': css'' = [(pnorm@cs,pabr@cs)]@css and
        pcss-k: pcss = [(pnorm,pabr)]
  by (force simp add: neq-Nil-conv split: if-split-asm)
with cs''-Nil step-i-full
have Γ⊢([],[(pnorm@cs,pabr@cs)]@css,t'') → (cs',css',t')
  by simp
from step-Nil [OF this]
obtain
  css': css'=css and
  cs': (case t'' of
        Abrupt s' ⇒ cs' = pabr @ cs ∧ t' = Normal s'
        | - ⇒ cs' = pnorm @ cs ∧ t' = t'')
  by (simp cong: xstate.case-cong)
let ?pcs-Suc-k = (case t'' of Abrupt s' ⇒ pabr | - ⇒ pnorm)
from css'' cs' cs' pcss-k f-Suc-k eq-i-Suc-k
show ?thesis
  apply (rule-tac x=?pcs-Suc-k in exI)
  apply (rule-tac x=[] in exI)
  apply (simp split: xstate.splits add: CS-def CSS-def)
done
qed
qed
qed
qed
qed
qed

```

**lemma** (in *inf*) *inf-progress*:  
**assumes** *f-0*:  $f\ 0 = (c\#\ cs,css,s)$   
**assumes** *f-step*:  $\forall i. \Gamma\vdash f(i) \rightarrow f(Suc\ i)$   
**assumes** *unfinished*:  $\forall i. \neg ((CS\ (f\ i) = cs) \wedge (CSS\ (f\ i) = css))$   
**shows**  $\exists pcs\ pcss.$



(if  $pcss = []$  then  $CSS (f i) = css \wedge CS (f i) = pcs @ cs$   
 else  $CS (f i) = pcs \wedge$   
 $CSS (f i) = butlast pcss @$   
 $[(fst (last pcss) @ cs, (snd (last pcss)) @ cs)] @$   
 $css)$

**proof** –  
**from** *steps-hd-progress* [*OF f-0 f-step, of i*] *unfinished*  
**show** *?thesis*  
**by** *auto*  
**qed**

**lemma** *skolemize1*:  $\forall x. P x \longrightarrow (\exists y. Q x y) \Longrightarrow \exists f. \forall x. P x \longrightarrow Q x (f x)$   
**by** (*rule choice*) *blast*

**lemma** *skolemize2*:  $\forall x. P x \longrightarrow (\exists y z. Q x y z) \Longrightarrow \exists f g. \forall x. P x \longrightarrow Q x (f x)$   
 (*g x*)  
**apply** (*drule skolemize1*)  
**apply** (*erule exE*)  
**apply** (*drule skolemize1*)  
**apply** *fast*  
**done**

**lemma** *skolemize2'*:  $\forall x. \exists y z. P x y z \Longrightarrow \exists f g. \forall x. P x (f x) (g x)$   
**apply** (*drule choice*)  
**apply** (*erule exE*)  
**apply** (*drule choice*)  
**apply** *fast*  
**done**

**theorem** (*in inf*) *inf-cases*:  
**fixes**  $c::('s, 'p, 'f) com$   
**assumes** *inf*:  $inf \Gamma (c \# cs) css s$   
**shows**  $inf \Gamma [c] [] s \vee (\exists t. \Gamma \vdash \langle c, s \rangle \Rightarrow t \wedge inf \Gamma cs css t)$

**proof** –  
**from** *inf* **obtain** *f* **where**  
 $f-0: f 0 = (c \# cs, css, s)$  **and**  
 $f-step: (\forall i. \Gamma \vdash f i \rightarrow f (Suc i))$   
**by** (*auto simp add: inf-def*)  
**show** *?thesis*  
**proof** (*cases*  $\exists i. CS (f i) = cs \wedge CSS (f i) = css$ )  
**case** *True*  
**define** *k* **where**  $k = (LEAST i. CS (f i) = cs \wedge CSS (f i) = css)$   
**from** *True*  
**obtain** *CS-f-k*:  $CS (f k) = cs$  **and** *CSS-f-k*:  $CSS (f k) = css$   
**apply** –  
**apply** (*erule exE*)  
**apply** (*drule LeastI*)  
**apply** (*simp add: k-def*)  
**done**

```

have less-k-prop:  $\forall i < k. \neg (CS (f i) = cs \wedge CSS (f i) = css)$ 
  apply (intro allI impI)
  apply (unfold k-def)
  apply (drule not-less-Least)
  apply simp
  done
have  $\Gamma \vdash ([c], [], s) \rightarrow^* ([], [], S (f k))$ 
proof –
  have  $\forall i \leq k. \exists pcs pcss.$ 
    (if pcss = [] then CSS (f i) = css  $\wedge$  CS (f i) = pcs @ cs
     else CS (f i) = pcs  $\wedge$ 
     CSS (f i) = butlast pcss @
     [(fst (last pcss) @ cs, (snd (last pcss)) @ cs)] @
     css)
    by (rule steps-hd-progress
      [OF f-0 f-step, where k=k, OF less-k-prop])
  from skolemize2 [OF this] obtain pcs pcss where
    pcs-pcss:
       $\forall i \leq k.$ 
      (if pcss i = [] then CSS (f i) = css  $\wedge$  CS (f i) = pcs i @ cs
       else CS (f i) = pcs i  $\wedge$ 
       CSS (f i) = butlast (pcss i) @
       [(fst (last (pcss i)) @ cs, (snd (last (pcss i))) @ cs)] @
       css)
      by iprover
  from pcs-pcss [rule-format, of k] CS-f-k CSS-f-k
  have finished: pcs k = [] pcss k = []
    by (auto simp add: CS-def CSS-def S-def split: if-split-asm)
  from pcs-pcss
  have simul:  $\forall i \leq k. (if pcss i = [] then CSS (f i) = css \wedge CS (f i) = pcs i @ cs$ 
    else CS (f i) = pcs i  $\wedge$ 
    CSS (f i) = butlast (pcss i) @
    [(fst (last (pcss i)) @ cs, (snd (last (pcss i))) @ cs)] @
    css)
    by auto
  from steps-hd-drop-suffix-finite [OF f-0 f-step less-k-prop simul] finished
  show ?thesis
    by simp
qed
hence  $\Gamma \vdash \langle c, s \rangle \Rightarrow S (f k)$ 
  by (rule steps-impl-exec)
moreover
from CS-f-k CSS-f-k f-step
have inf  $\Gamma \text{ cs css } (S (f k))$ 
  apply (simp add: inf-def)
  apply (rule-tac x =  $\lambda i. f (i + k)$  in exI)
  apply simp
  apply (auto simp add: CS-def CSS-def S-def)
done

```

**ultimately**  
**have**  $(\exists t. \Gamma \vdash \langle c, s \rangle \Rightarrow t \wedge \text{inf } \Gamma \text{ cs css } t)$   
**by** *blast*  
**thus** *?thesis*  
**by** *simp*  
**next**  
**case** *False*  
**hence** *unfinished*:  $\forall i. \neg ((CS (f i) = cs) \wedge (CSS (f i) = css))$   
**by** *simp*  
**from** *inf-progress* [*OF f-0 f-step this*]  
**have**  $\forall i. \exists pcs \ pcss.$   
 $(\text{if } pcss = [] \text{ then } CSS (f i) = css \wedge CS (f i) = pcs @ cs$   
 $\text{else } CS (f i) = pcs \wedge$   
 $CSS (f i) = \text{butlast } pcss @$   
 $[(fst (last pcss) @ cs, (snd (last pcss) @ cs))] @$   
 $css)$   
**by** *auto*  
**from** *skolemize2'* [*OF this*] **obtain** *pcs pcss where*  
 $pcs\text{-}pcss: \forall i.$   
 $(\text{if } pcss \ i = [] \text{ then } CSS (f i) = css \wedge CS (f i) = pcs \ i @ cs$   
 $\text{else } CS (f i) = pcs \ i \wedge$   
 $CSS (f i) = \text{butlast } (pcss \ i) @$   
 $[(fst (last (pcss \ i)) @ cs, (snd (last (pcss \ i))) @ cs))] @$   
 $css)$   
**by** *iprover*  
**define** *g where*  $g \ i = (pcs \ i, pcss \ i, S (f i))$  **for** *i*  
**from** *pcs-pcss* [*rule-format, of 0*] *f-0*  
**have**  $g \ 0 = ([c], [], s)$   
**by** (*auto split: if-split-asm simp add: CS-def CSS-def S-def g-def*)  
**moreover**  
**from** *steps-hd-drop-suffix-infinite* [*OF f-0 f-step unfinished pcs-pcss*]  
**have**  $\forall i. \Gamma \vdash g \ i \rightarrow g \ (Suc \ i)$   
**by** (*simp add: g-def*)  
**ultimately**  
**have**  $\text{inf } \Gamma \ [c] \ [] \ s$   
**by** (*auto simp add: inf-def*)  
**thus** *?thesis*  
**by** *simp*  
**qed**  
**qed**

**lemma** *infE* [*consumes 1*]:  
**assumes** *inf*:  $\text{inf } \Gamma \ (c \# cs) \ css \ s$   
**assumes** *cases*:  $\text{inf } \Gamma \ [c] \ [] \ s \Longrightarrow P$   
 $\wedge t. [\Gamma \vdash \langle c, s \rangle \Rightarrow t; \text{inf } \Gamma \text{ cs css } t] \Longrightarrow P$   
**shows** *P*  
**using** *inf cases*  
**apply**  $-$   
**apply** (*drule inf.inf-cases*)

**apply** *auto*  
**done**

**lemma** *inf-Seq*:

$\text{inf } \Gamma (Seq\ c1\ c2\ \#cs)\ css\ (Normal\ s) = \text{inf } \Gamma (c1\ \#c2\ \#cs)\ css\ (Normal\ s)$

**proof**

**assume**  $\text{inf } \Gamma (Seq\ c1\ c2\ \#cs)\ css\ (Normal\ s)$

**then obtain** *f* **where**

*f-0*:  $f\ 0 = (Seq\ c1\ c2\ \#cs, css, Normal\ s)$  **and**

*f-step*:  $\forall i. \Gamma \vdash f\ i \rightarrow f\ (Suc\ i)$

**by** (*auto simp add: inf-def*)

**from** *f-step* [*rule-format, of 0*] *f-0*

**have**  $f\ 1 = (c1\ \#c2\ \#cs, css, Normal\ s)$

**by** (*auto elim: step-Normal-elim-cases*)

**with** *f-step* **show**  $\text{inf } \Gamma (c1\ \#c2\ \#cs)\ css\ (Normal\ s)$

**apply** (*simp add: inf-def*)

**apply** (*rule-tac x= $\lambda i. f\ (Suc\ i)$  in exI*)

**apply** *simp*

**done**

**next**

**assume**  $\text{inf } \Gamma (c1\ \#c2\ \#cs)\ css\ (Normal\ s)$

**then obtain** *f* **where**

*f-0*:  $f\ 0 = (c1\ \#c2\ \#cs, css, Normal\ s)$  **and**

*f-step*:  $\forall i. \Gamma \vdash f\ i \rightarrow f\ (Suc\ i)$

**by** (*auto simp add: inf-def*)

**define** *g* **where**  $g\ i = (\text{case } i\ \text{of } 0 \Rightarrow (Seq\ c1\ c2\ \#cs, css, Normal\ s) \mid Suc\ j \Rightarrow f\ j)$  **for** *i*

**with** *f-0* **have**

$\Gamma \vdash g\ 0 \rightarrow g\ (Suc\ 0)$

**by** (*auto intro: step.intros*)

**moreover**

**from** *f-step* **have**  $\forall i. i \neq 0 \longrightarrow \Gamma \vdash g\ i \rightarrow g\ (Suc\ i)$

**by** (*auto simp add: g-def split: nat.splits*)

**ultimately**

**show**  $\text{inf } \Gamma (Seq\ c1\ c2\ \#cs)\ css\ (Normal\ s)$

**apply** (*simp add: inf-def*)

**apply** (*rule-tac x= $g$  in exI*)

**apply** (*simp add: g-def split: nat.splits*)

**done**

**qed**

**lemma** *inf-WhileTrue*:

**assumes**  $b: s \in b$

**shows**  $\text{inf } \Gamma (While\ b\ c\ \#cs)\ css\ (Normal\ s) =$

$\text{inf } \Gamma (c\ \#While\ b\ c\ \#cs)\ css\ (Normal\ s)$

**proof**

**assume**  $\text{inf } \Gamma (While\ b\ c\ \#cs)\ css\ (Normal\ s)$

**then obtain** *f* **where**

*f-0*:  $f\ 0 = (While\ b\ c\ \#cs, css, Normal\ s)$  **and**

*f-step*:  $\forall i. \Gamma \vdash f i \rightarrow f (Suc i)$   
**by** (*auto simp add: inf-def*)  
**from** *b f-step* [*rule-format, of 0*] *f-0*  
**have** *f 1* = (*c#While b c#cs,css,Normal s*)  
**by** (*auto elim: step-Normal-elim-cases*)  
**with** *f-step* **show** *inf*  $\Gamma (c \# While b c \# cs) \text{ css } (Normal s)$   
**apply** (*simp add: inf-def*)  
**apply** (*rule-tac x= $\lambda i. f (Suc i)$  in exI*)  
**apply** *simp*  
**done**  
**next**  
**assume** *inf*  $\Gamma (c \# While b c \# cs) \text{ css } (Normal s)$   
**then obtain** *f* **where**  
*f-0*: *f 0* = (*c#While b c#cs,css,Normal s*) **and**  
*f-step*:  $\forall i. \Gamma \vdash f i \rightarrow f (Suc i)$   
**by** (*auto simp add: inf-def*)  
**define** *h* **where** *h i* = (*case i of 0  $\Rightarrow$  (While b c#cs,css,Normal s) | Suc j  $\Rightarrow$  f j*) **for** *i*  
**with** *b f-0* **have**  
 $\Gamma \vdash h 0 \rightarrow h (Suc 0)$   
**by** (*auto intro: step.intros*)  
**moreover**  
**from** *f-step* **have**  $\forall i. i \neq 0 \longrightarrow \Gamma \vdash h i \rightarrow h (Suc i)$   
**by** (*auto simp add: h-def split: nat.splits*)  
**ultimately**  
**show** *inf*  $\Gamma (While b c \# cs) \text{ css } (Normal s)$   
**apply** (*simp add: inf-def*)  
**apply** (*rule-tac x=h in exI*)  
**apply** (*simp add: h-def split: nat.splits*)  
**done**  
**qed**

**lemma** *inf-Catch*:

*inf*  $\Gamma (Catch c1 c2 \# cs) \text{ css } (Normal s) = \text{inf } \Gamma [c1] ((cs, c2 \# cs) \# css) (Normal s)$

**proof**

**assume** *inf*  $\Gamma (Catch c1 c2 \# cs) \text{ css } (Normal s)$   
**then obtain** *f* **where**  
*f-0*: *f 0* = (*Catch c1 c2#cs,css,Normal s*) **and**  
*f-step*:  $\forall i. \Gamma \vdash f i \rightarrow f (Suc i)$   
**by** (*auto simp add: inf-def*)  
**from** *f-step* [*rule-format, of 0*] *f-0*  
**have** *f 1* = (*[c1],(cs,c2#cs)#css,Normal s*)  
**by** (*auto elim: step-Normal-elim-cases*)  
**with** *f-step* **show** *inf*  $\Gamma [c1] ((cs, c2 \# cs) \# css) (Normal s)$   
**apply** (*simp add: inf-def*)  
**apply** (*rule-tac x= $\lambda i. f (Suc i)$  in exI*)  
**apply** *simp*  
**done**  
**next**

```

assume inf  $\Gamma [c1] ((cs,c2\#cs)\#css) (Normal\ s)$ 
then obtain f where
  f-0:  $f\ 0 = ([c1],(cs,c2\#cs)\#css,Normal\ s)$  and
  f-step:  $\forall i. \Gamma \vdash f\ i \rightarrow f\ (Suc\ i)$ 
  by (auto simp add: inf-def)
define h where  $h\ i = (case\ i\ of\ 0 \Rightarrow (Catch\ c1\ c2\ \#cs,css,Normal\ s) \mid Suc\ j \Rightarrow$ 
f j) for i
with f-0 have
   $\Gamma \vdash h\ 0 \rightarrow h\ (Suc\ 0)$ 
  by (auto intro: step.intros)
moreover
from f-step have  $\forall i. i \neq 0 \longrightarrow \Gamma \vdash h\ i \rightarrow h\ (Suc\ i)$ 
  by (auto simp add: h-def split: nat.splits)
ultimately
show inf  $\Gamma (Catch\ c1\ c2\ \#cs) css (Normal\ s)$ 
  apply (simp add: inf-def)
  apply (rule-tac x=h in exI)
  apply (simp add: h-def split: nat.splits)
done
qed

```

```

theorem terminates-impl-not-inf:
  assumes termi:  $\Gamma \vdash c \downarrow s$ 
  shows  $\neg inf\ \Gamma [c] []\ s$ 
using termi
proof induct
  case (Skip s) thus ?case
  proof (rule not-infI)
    fix f
    assume f-step:  $\bigwedge i. \Gamma \vdash f\ i \rightarrow f\ (Suc\ i)$ 
    assume f-0:  $f\ 0 = ([Skip], [], Normal\ s)$ 
    from f-step [of 0] f-0
    have  $f\ (Suc\ 0) = ([], [], Normal\ s)$ 
    by (auto elim: step-Normal-elim-cases)
    with f-step [of 1]
    show False
    by (auto elim: step-elim-cases)
  qed

```

```

next
case (Basic g s)
thus ?case
proof (rule not-infI)
  fix f
  assume f-step:  $\bigwedge i. \Gamma \vdash f\ i \rightarrow f\ (Suc\ i)$ 
  assume f-0:  $f\ 0 = ([Basic\ g], [], Normal\ s)$ 
  from f-step [of 0] f-0
  have  $f\ (Suc\ 0) = ([], [], Normal\ (g\ s))$ 
  by (auto elim: step-Normal-elim-cases)
  with f-step [of 1]

```

```

    show False
      by (auto elim: step-elim-cases)
  qed
next
case (Spec r s)
thus ?case
proof (rule not-infI)
  fix f
  assume f-step:  $\bigwedge i. \Gamma \vdash f\ i \rightarrow f\ (Suc\ i)$ 
  assume f-0:  $f\ 0 = ([Spec\ r], [], Normal\ s)$ 
  with f-step [of 0]
  have  $\Gamma \vdash ([Spec\ r], [], Normal\ s) \rightarrow f\ (Suc\ 0)$ 
    by simp
  then show False
  proof (cases)
    fix t
    assume  $(s, t) \in r\ f\ (Suc\ 0) = ([], [], Normal\ t)$ 
    with f-step [of 1]
    show False
      by (auto elim: step-elim-cases)
  next
    assume  $\forall t. (s, t) \notin r\ f\ (Suc\ 0) = ([], [], Stuck)$ 
    with f-step [of 1]
    show False
      by (auto elim: step-elim-cases)
  qed
  qed
next
case (Guard s g c m)
have  $g: s \in g$  by fact
have hyp:  $\neg\ inf\ \Gamma\ [c]\ []\ (Normal\ s)$  by fact
show ?case
proof (rule not-infI)
  fix f
  assume f-step:  $\bigwedge i. \Gamma \vdash f\ i \rightarrow f\ (Suc\ i)$ 
  assume f-0:  $f\ 0 = ([Guard\ m\ g\ c], [], Normal\ s)$ 
  from g f-step [of 0] f-0
  have  $f\ (Suc\ 0) = ([c], [], Normal\ s)$ 
    by (auto elim: step-Normal-elim-cases)
  with f-step
  have  $inf\ \Gamma\ [c]\ []\ (Normal\ s)$ 
    apply (simp add: inf-def)
    apply (rule-tac  $x = \lambda i. f\ (Suc\ i)$  in exI)
    by simp
  with hyp show False ..
  qed
next
case (GuardFault s g m c)
have  $g: s \notin g$  by fact

```

```

show ?case
proof (rule not-infI)
  fix f
  assume f-step:  $\bigwedge i. \Gamma \vdash f i \rightarrow f (Suc i)$ 
  assume f-0:  $f 0 = ([Guard\ m\ g\ c], [], Normal\ s)$ 
  from g f-step [of 0] f-0
  have  $f (Suc\ 0) = ([], [], Fault\ m)$ 
    by (auto elim: step-Normal-elim-cases)
  with f-step [of 1]
  show False
    by (auto elim: step-elim-cases)
qed
next
case (Fault c m)
thus ?case
proof (rule not-infI)
  fix f
  assume f-step:  $\bigwedge i. \Gamma \vdash f i \rightarrow f (Suc i)$ 
  assume f-0:  $f 0 = ([c], [], Fault\ m)$ 
  from f-step [of 0] f-0
  have  $f (Suc\ 0) = ([], [], Fault\ m)$ 
    by (auto elim: step-Normal-elim-cases)
  with f-step [of 1]
  show False
    by (auto elim: step-elim-cases)
qed
next
case (Seq c1 s c2)
have hyp-c1:  $\neg inf\ \Gamma\ [c1]\ []\ (Normal\ s)$  by fact
have hyp-c2:  $\forall s'. \Gamma \vdash \langle c1, Normal\ s \rangle \Rightarrow s' \longrightarrow \Gamma \vdash c2 \downarrow s' \wedge \neg inf\ \Gamma\ [c2]\ []\ s'$  by
fact
have  $\neg inf\ \Gamma\ ([c1, c2])\ []\ (Normal\ s)$ 
proof
  assume  $inf\ \Gamma\ [c1, c2]\ []\ (Normal\ s)$ 
  then show False
  proof (cases rule: infE)
    assume  $inf\ \Gamma\ [c1]\ []\ (Normal\ s)$ 
    with hyp-c1 show ?thesis by simp
  next
  fix t
  assume  $\Gamma \vdash \langle c1, Normal\ s \rangle \Rightarrow t\ inf\ \Gamma\ [c2]\ []\ t$ 
  with hyp-c2 show ?thesis by simp
  qed
qed
thus ?case
  by (simp add: inf-Seq)
next
case (CondTrue s b c1 c2)
have  $b: s \in b$  by fact

```



```

have hyp-c1:  $\neg \text{inf } \Gamma [c1] []$  (Normal s) by fact
show ?case
proof (rule not-infI)
  fix f
  assume f-step:  $\bigwedge i. \Gamma \vdash f i \rightarrow f (\text{Suc } i)$ 
  assume f-0:  $f 0 = ([\text{Cond } b \ c1 \ c2], [], \text{Normal } s)$ 
  from b f-step [of 0] f-0
  have f 1 =  $([c1], [], \text{Normal } s)$ 
    by (auto elim: step-Normal-elim-cases)
  with f-step
  have inf  $\Gamma [c1] []$  (Normal s)
    apply (simp add: inf-def)
    apply (rule-tac  $x = \lambda i. f (\text{Suc } i)$  in exI)
    by simp
  with hyp-c1 show False by simp
qed
next
case (CondFalse s b c2 c1)
have b:  $s \notin b$  by fact
have hyp-c2:  $\neg \text{inf } \Gamma [c2] []$  (Normal s) by fact
show ?case
proof (rule not-infI)
  fix f
  assume f-step:  $\bigwedge i. \Gamma \vdash f i \rightarrow f (\text{Suc } i)$ 
  assume f-0:  $f 0 = ([\text{Cond } b \ c1 \ c2], [], \text{Normal } s)$ 
  from b f-step [of 0] f-0
  have f 1 =  $([c2], [], \text{Normal } s)$ 
    by (auto elim: step-Normal-elim-cases)
  with f-step
  have inf  $\Gamma [c2] []$  (Normal s)
    apply (simp add: inf-def)
    apply (rule-tac  $x = \lambda i. f (\text{Suc } i)$  in exI)
    by simp
  with hyp-c2 show False by simp
qed
next
case (WhileTrue s b c)
have b:  $s \in b$  by fact
have hyp-c:  $\neg \text{inf } \Gamma [c] []$  (Normal s) by fact
have hyp-w:  $\forall s'. \Gamma \vdash \langle c, \text{Normal } s \rangle \Rightarrow s' \longrightarrow$ 
 $\Gamma \vdash \text{While } b \ c \downarrow s' \wedge \neg \text{inf } \Gamma [\text{While } b \ c] [] \ s'$  by fact
have  $\neg \text{inf } \Gamma [c, \text{While } b \ c] []$  (Normal s)
proof
  assume inf  $\Gamma [c, \text{While } b \ c] []$  (Normal s)
  from this hyp-c hyp-w show False
    by (cases rule: infE) auto
qed
with b show ?case
  by (simp add: inf-WhileTrue)

```

```

next
case (WhileFalse s b c)
have b: s  $\notin$  b by fact
show ?case
proof (rule not-infI)
  fix f
  assume f-step:  $\bigwedge i. \Gamma \vdash f i \rightarrow f (Suc i)$ 
  assume f-0:  $f 0 = ([While b c], [], Normal s)$ 
  from b f-step [of 0] f-0
  have f (Suc 0) = ([], [], Normal s)
    by (auto elim: step-Normal-elim-cases)
  with f-step [of 1]
  show False
    by (auto elim: step-elim-cases)
qed
next
case (Call p bdy s)
have bdy:  $\Gamma p = Some bdy$  by fact
have hyp:  $\neg inf \Gamma [bdy] [] (Normal s)$  by fact
have not-inf-bdy:
   $\neg inf \Gamma [bdy] [([], [Throw])] (Normal s)$ 
proof
  assume inf  $\Gamma [bdy] [([], [Throw])] (Normal s)$ 
  then show False
  proof (rule infE)
    assume inf  $\Gamma [bdy] [] (Normal s)$ 
    with hyp show False by simp
  next
  fix t
  assume  $\Gamma \vdash \langle bdy, Normal s \rangle \Rightarrow t$ 
  assume inf:  $inf \Gamma [] [([], [Throw])] t$ 
  then obtain f where
    f-0:  $f 0 = ([], [([], [Throw])], t)$  and
    f-step:  $\forall i. \Gamma \vdash f i \rightarrow f (Suc i)$ 
    by (auto simp add: inf-def)
  show False
  proof (cases t)
    case (Normal t')
    with f-0 f-step [rule-format, of 0]
    have f (Suc 0) = ([], [], (Normal t'))
      by (auto elim: step-Normal-elim-cases)
    with f-step [rule-format, of Suc 0]
    show False
      by (auto elim: step.cases)
  next
  case (Abrupt t')
  with f-0 f-step [rule-format, of 0]
  have f (Suc 0) = ([Throw], [], (Normal t'))
    by (auto elim: step-Normal-elim-cases)

```

```

with f-step [rule-format, of Suc 0]
have f (Suc (Suc 0)) = ([],[],(Abrupt t'))
  by (auto elim: step-Normal-elim-cases)
with f-step [rule-format, of Suc(Suc 0)]
show False
  by (auto elim: step.cases)
next
case (Fault m)
with f-0 f-step [rule-format, of 0]
have f (Suc 0) = ([],[],Fault m)
  by (auto elim: step-Normal-elim-cases)
with f-step [rule-format, of 1]
have f (Suc (Suc 0)) = ([],[],Fault m)
  by (auto elim: step-Normal-elim-cases)
with f-step [rule-format, of Suc (Suc 0)]
show False
  by (auto elim: step.cases)
next
case Stuck
with f-0 f-step [rule-format, of 0]
have f (Suc 0) = ([],[],Stuck)
  by (auto elim: step-Normal-elim-cases)
with f-step [rule-format, of 1]
have f (Suc (Suc 0)) = ([],[],Stuck)
  by (auto elim: step-Normal-elim-cases)
with f-step [rule-format, of Suc (Suc 0)]
show False
  by (auto elim: step.cases)
qed
qed
qed
show ?case
proof (rule not-infI)
fix f
assume f-step:  $\bigwedge i. \Gamma \vdash f i \rightarrow f (Suc i)$ 
assume f-0:  $f 0 = ([Call\ p], [], Normal\ s)$ 
from bdy f-step [of 0] f-0
have f (Suc 0) =
  ([bdy],[([],[Throw]),Normal s])
  by (auto elim: step-Normal-elim-cases)
with f-step
have inf  $\Gamma [bdy] [([],[Throw])] (Normal\ s)$ 
  apply (simp add: inf-def)
  apply (rule-tac  $x = \lambda i. f (Suc i)$  in exI)
  by simp
with not-inf-bdy
show False by simp
qed
next

```

```

case (CallUndefined p s)
have undef:  $\Gamma p = \text{None}$  by fact
show ?case
proof (rule not-infI)
  fix f
  assume f-step:  $\bigwedge i. \Gamma \vdash f i \rightarrow f (\text{Suc } i)$ 
  assume f-0:  $f 0 = ([\text{Call } p], [], \text{Normal } s)$ 
  from undef f-step [of 0] f-0
  have  $f (\text{Suc } 0) = ([], [], \text{Stuck})$ 
  by (auto elim: step-Normal-elim-cases)
  with f-step [rule-format, of Suc 0]
  show False
  by (auto elim: step-elim-cases)
qed
next
case (Stuck c)
show ?case
proof (rule not-infI)
  fix f
  assume f-step:  $\bigwedge i. \Gamma \vdash f i \rightarrow f (\text{Suc } i)$ 
  assume f-0:  $f 0 = ([c], [], \text{Stuck})$ 
  from f-step [of 0] f-0
  have  $f (\text{Suc } 0) = ([], [], \text{Stuck})$ 
  by (auto elim: step-elim-cases)
  with f-step [rule-format, of Suc 0]
  show False
  by (auto elim: step-elim-cases)
qed
next
case (DynCom c s)
have hyp:  $\neg \text{inf } \Gamma [(c \ s)] [] (\text{Normal } s)$  by fact
show ?case
proof (rule not-infI)
  fix f
  assume f-step:  $\bigwedge i. \Gamma \vdash f i \rightarrow f (\text{Suc } i)$ 
  assume f-0:  $f 0 = ([\text{DynCom } c], [], \text{Normal } s)$ 
  from f-step [of 0] f-0
  have  $f (\text{Suc } 0) = ([ (c \ s) ], [], \text{Normal } s)$ 
  by (auto elim: step-elim-cases)
  with f-step have  $\text{inf } \Gamma [(c \ s)] [] (\text{Normal } s)$ 
  apply (simp add: inf-def)
  apply (rule-tac  $x = \lambda i. f (\text{Suc } i)$  in exI)
  by simp
  with hyp
  show False by simp
qed
next
case (Throw s)
thus ?case

```

```

proof (rule not-infI)
  fix f
  assume f-step:  $\bigwedge i. \Gamma \vdash f i \rightarrow f (Suc i)$ 
  assume f-0:  $f 0 = ([Throw], [], Normal s)$ 
  from f-step [of 0] f-0
  have  $f (Suc 0) = ([], [], Abrupt s)$ 
    by (auto elim: step-Normal-elim-cases)
  with f-step [of 1]
  show False
    by (auto elim: step-elim-cases)
qed
next
case (Abrupt c s)
show ?case
proof (rule not-infI)
  fix f
  assume f-step:  $\bigwedge i. \Gamma \vdash f i \rightarrow f (Suc i)$ 
  assume f-0:  $f 0 = ([c], [], Abrupt s)$ 
  from f-step [of 0] f-0
  have  $f (Suc 0) = ([], [], Abrupt s)$ 
    by (auto elim: step-elim-cases)
  with f-step [rule-format, of Suc 0]
  show False
    by (auto elim: step-elim-cases)
qed
next
case (Catch c1 s c2)
have hyp-c1:  $\neg inf \Gamma [c1] [] (Normal s)$  by fact
have hyp-c2:  $\forall s'. \Gamma \vdash \langle c1, Normal s \rangle \Rightarrow Abrupt s' \longrightarrow$ 
   $\Gamma \vdash c2 \downarrow Normal s' \wedge \neg inf \Gamma [c2] [] (Normal s')$  by fact
have  $\neg inf \Gamma [c1] [([], [c2])] (Normal s)$ 
proof
  assume  $inf \Gamma [c1] [([], [c2])] (Normal s)$ 
  then show False
  proof (rule infE)
    assume  $inf \Gamma [c1] [] (Normal s)$ 
    with hyp-c1 show False by simp
  next
  fix t
  assume eval:  $\Gamma \vdash \langle c1, Normal s \rangle \Rightarrow t$ 
  assume inf:  $inf \Gamma [] [([], [c2])] t$ 
  then obtain f where
    f-0:  $f 0 = ([], [([], [c2])], t)$  and
    f-step:  $\forall i. \Gamma \vdash f i \rightarrow f (Suc i)$ 
    by (auto simp add: inf-def)
  show False
proof (cases t)
  case (Normal t')
  with f-0 f-step [rule-format, of 0]

```

```

have  $f (Suc\ 0) = ([], [], Normal\ t')$ 
  by (auto elim: step-Normal-elim-cases)
with f-step [rule-format, of 1]
show False
  by (auto elim: step-elim-cases)
next
case (Abrupt t')
with f-0 f-step [rule-format, of 0]
have  $f (Suc\ 0) = ([c2], [], Normal\ t')$ 
  by (auto elim: step-Normal-elim-cases)
with f-step eval Abrupt
have  $inf\ \Gamma\ [c2]\ []\ (Normal\ t')$ 
  apply (simp add: inf-def)
  apply (rule-tac x= $\lambda i.$  f (Suc i) in exI)
  by simp
with eval hyp-c2 Abrupt show False by simp
next
case (Fault m)
with f-0 f-step [rule-format, of 0]
have  $f (Suc\ 0) = ([], [], Fault\ m)$ 
  by (auto elim: step-Normal-elim-cases)
with f-step [rule-format, of 1]
show False
  by (auto elim: step-elim-cases)
next
case Stuck
with f-0 f-step [rule-format, of 0]
have  $f (Suc\ 0) = ([], [], Stuck)$ 
  by (auto elim: step-Normal-elim-cases)
with f-step [rule-format, of 1]
show False
  by (auto elim: step-elim-cases)
qed
qed
qed
thus ?case
  by (simp add: inf-Catch)
qed

```

```

lemma terminatess-impl-not-inf:
assumes termi:  $\Gamma \vdash cs, css \Downarrow s$ 
shows  $\neg inf\ \Gamma\ cs\ css\ s$ 
using termi
proof (induct)
case (Nil s)
show ?case
proof (rule not-infI)
  fix f
  assume  $\bigwedge i. \Gamma \vdash f\ i \rightarrow f\ (Suc\ i)$ 

```

```

    hence  $\Gamma \vdash f\ 0 \rightarrow f\ (Suc\ 0)$ 
      by simp
    moreover
    assume  $f\ 0 = ([], [], s)$ 
    ultimately show False
      by (fastforce elim: step.cases)
  qed
next
case (ExitBlockNormal nrms css s abrs)
have hyp:  $\neg\ inf\ \Gamma\ nrms\ css\ (Normal\ s)$  by fact
show ?case
proof (rule not-infI)
  fix f
  assume f-step:  $\bigwedge i. \Gamma \vdash f\ i \rightarrow f\ (Suc\ i)$ 
  assume f0:  $f\ 0 = ([], (nrms, abrs) \# css, Normal\ s)$ 
  with f-step [of 0] have  $f\ (Suc\ 0) = (nrms, css, Normal\ s)$ 
    by (auto elim: step-Normal-elim-cases)
  with f-step have  $inf\ \Gamma\ nrms\ css\ (Normal\ s)$ 
    apply (simp add: inf-def)
    apply (rule-tac x= $\lambda i. f\ (Suc\ i)$  in exI)
    by simp
  with hyp show False ..
  qed
next
case (ExitBlockAbrupt abrs css s nrms)
have hyp:  $\neg\ inf\ \Gamma\ abrs\ css\ (Normal\ s)$  by fact
show ?case
proof (rule not-infI)
  fix f
  assume f-step:  $\bigwedge i. \Gamma \vdash f\ i \rightarrow f\ (Suc\ i)$ 
  assume f0:  $f\ 0 = ([], (nrms, abrs) \# css, Abrupt\ s)$ 
  with f-step [of 0] have  $f\ (Suc\ 0) = (abrs, css, Normal\ s)$ 
    by (auto elim: step-Normal-elim-cases)
  with f-step have  $inf\ \Gamma\ abrs\ css\ (Normal\ s)$ 
    apply (simp add: inf-def)
    apply (rule-tac x= $\lambda i. f\ (Suc\ i)$  in exI)
    by simp
  with hyp show False ..
  qed
next
case (ExitBlockFault nrms css f abrs)
show ?case
  by (rule not-inf-Fault)
next
case (ExitBlockStuck nrms css abrs)
show ?case
  by (rule not-inf-Stuck)
next
case (Cons c s cs css)

```

```

have termi-c:  $\Gamma \vdash c \downarrow s$  by fact
have hyp:  $\forall t. \Gamma \vdash \langle c, s \rangle \Rightarrow t \longrightarrow \Gamma \vdash cs, css \downarrow t \wedge \neg \text{inf } \Gamma \text{ cs css } t$  by fact
show  $\neg \text{inf } \Gamma (c \# cs) \text{ css } s$ 
proof
  assume  $\text{inf } \Gamma (c \# cs) \text{ css } s$ 
  thus False
  proof (rule infE)
    assume  $\text{inf } \Gamma [c] [] s$ 
    with terminates-impl-not-inf [OF termi-c]
    show False ..
  next
  fix t
  assume  $\Gamma \vdash \langle c, s \rangle \Rightarrow t \text{ inf } \Gamma \text{ cs css } t$ 
  with hyp show False by simp
  qed
qed
qed

```

```

lemma lem:
   $\forall y. r^{++} a y \longrightarrow P a \longrightarrow P y$ 
   $\implies ((b, a) \in \{(y, x). P x \wedge r x y\}^+) = ((b, a) \in \{(y, x). P x \wedge r^{++} x y\})$ 
apply(rule iffI)
apply clarify
apply(erule trancl-induct)
apply blast
apply(blast intro:tranclp-trans)
apply clarify
apply(erule tranclp-induct)
apply blast
apply(blast intro:trancl-trans)
done

```

```

corollary terminatess-impl-no-inf-chain:
assumes terminatess:  $\Gamma \vdash cs, css \downarrow s$ 
shows  $\neg(\exists f. f 0 = (cs, css, s) \wedge (\forall i::\text{nat}. \Gamma \vdash f i \rightarrow^+ f(\text{Suc } i)))$ 
proof -
  have  $\text{wf}(\{(y, x). \Gamma \vdash (cs, css, s) \rightarrow^* x \wedge \Gamma \vdash x \rightarrow y\}^+)$ 
  proof (rule wf-trancl)
    show  $\text{wf } \{(y, x). \Gamma \vdash (cs, css, s) \rightarrow^* x \wedge \Gamma \vdash x \rightarrow y\}$ 
  proof (simp only: wf-iff-no-infinite-down-chain, clarify, simp)
    fix f
    assume  $\forall i. \Gamma \vdash (cs, css, s) \rightarrow^* f i \wedge \Gamma \vdash f i \rightarrow f(\text{Suc } i)$ 
    hence  $\exists f. f 0 = (cs, css, s) \wedge (\forall i. \Gamma \vdash f i \rightarrow f(\text{Suc } i))$ 
    by (rule rename [to-pred])
  moreover from terminatess
  have  $\neg(\exists f. f 0 = (cs, css, s) \wedge (\forall i. \Gamma \vdash f i \rightarrow f(\text{Suc } i)))$ 
  by (rule terminatess-impl-not-inf [unfolded inf-def])
  ultimately show False
  by simp

```



**qed**  
**qed**  
**hence**  $\neg (\exists f. \forall i. (f (Suc i), f i) \in \{(y, x). \Gamma \vdash (cs, css, s) \rightarrow^* x \wedge \Gamma \vdash x \rightarrow y\}^+)$   
**by** (*simp add: wf-iff-no-infinite-down-chain*)  
**thus** *?thesis*  
**proof** (*rule contrapos-nn*)  
**assume**  $\exists f. f 0 = (cs, css, s) \wedge (\forall i. \Gamma \vdash f i \rightarrow^+ f (Suc i))$   
**then obtain  $f$  where**  
 $f0: f 0 = (cs, css, s)$  **and**  
 $seq: \forall i. \Gamma \vdash f i \rightarrow^+ f (Suc i)$   
**by** *iprover*  
**show**  
 $\exists f. \forall i. (f (Suc i), f i) \in \{(y, x). \Gamma \vdash (cs, css, s) \rightarrow^* x \wedge \Gamma \vdash x \rightarrow y\}^+$   
**proof** (*rule exI [where x=f], rule allI*)  
**fix**  $i$   
**show**  $(f (Suc i), f i) \in \{(y, x). \Gamma \vdash (cs, css, s) \rightarrow^* x \wedge \Gamma \vdash x \rightarrow y\}^+$   
**proof** –  
{  
**fix**  $i$  **have**  $\Gamma \vdash (cs, css, s) \rightarrow^* f i$   
**proof** (*induct i*)  
**case**  $0$  **show**  $\Gamma \vdash (cs, css, s) \rightarrow^* f 0$   
**by** (*simp add: f0*)  
**next**  
**case**  $(Suc n)$   
**have**  $\Gamma \vdash (cs, css, s) \rightarrow^* f n$  **by** *fact*  
**with**  $seq$  **show**  $\Gamma \vdash (cs, css, s) \rightarrow^* f (Suc n)$   
**by** (*blast intro: tranclp-into-rtranclp rtranclp-trans*)  
**qed**  
}  
**hence**  $\Gamma \vdash (cs, css, s) \rightarrow^* f i$   
**by** *iprover*  
**with**  $seq$  **have**  
 $(f (Suc i), f i) \in \{(y, x). \Gamma \vdash (cs, css, s) \rightarrow^* x \wedge \Gamma \vdash x \rightarrow^+ y\}$   
**by** *clarsimp*  
**moreover**  
**have**  $\forall y. \Gamma \vdash f i \rightarrow^+ y \longrightarrow \Gamma \vdash (cs, css, s) \rightarrow^* f i \longrightarrow \Gamma \vdash (cs, css, s) \rightarrow^* y$   
**by** (*blast intro: tranclp-into-rtranclp rtranclp-trans*)  
**ultimately**  
**show** *?thesis*  
**by** (*subst lem*)  
**qed**  
**qed**  
**qed**  
**qed**

**corollary** *terminates-impl-no-inf-chain:*

$\Gamma \vdash c \downarrow s \implies \neg (\exists f. f 0 = ([c], [], s) \wedge (\forall i :: nat. \Gamma \vdash f i \rightarrow^+ f (Suc i)))$   
**by** (*rule terminatess-impl-no-inf-chain*) (*iprover intro: terminatess.intros*)

**definition**

$$\text{termi-call-steps} :: ('s, 'p, 'f) \text{ body} \Rightarrow (('s \times 'p) \times ('s \times 'p)) \text{set}$$
**where**

$$\text{termi-call-steps } \Gamma =$$

$$\{((t, q), (s, p)). \Gamma \vdash \text{the } (\Gamma \ p) \downarrow \text{Normal } s \wedge \\ (\exists \text{css}. \Gamma \vdash ([\text{the } (\Gamma \ p)], [], \text{Normal } s) \rightarrow^+ ([\text{the } (\Gamma \ q)], \text{css}, \text{Normal } t))\}$$

Sequencing computations, or more exactly continuation stacks

$$\text{primrec seq} :: (\text{nat} \Rightarrow 'a \text{ list}) \Rightarrow \text{nat} \Rightarrow 'a \text{ list}$$
**where**

$$\text{seq css } 0 = [] \mid$$

$$\text{seq css } (\text{Suc } i) = \text{css } i @ \text{seq css } i$$

**theorem** *wf-termi-call-steps*: *wf* (*termi-call-steps*  $\Gamma$ )

**proof** (*simp only*: *termi-call-steps-def* *wf-iff-no-infinite-down-chain*,  
*clarify*, *simp*)

**fix** *S*

**assume** *inf*:

$$\forall i. (\lambda(t, q) (s, p). \Gamma \vdash (\text{the } (\Gamma \ p)) \downarrow \text{Normal } s \wedge \\ (\exists \text{css}. \Gamma \vdash ([\text{the } (\Gamma \ p)], [], \text{Normal } s) \rightarrow^+ ([\text{the } (\Gamma \ q)], \text{css}, \text{Normal } t))) \\ (S (\text{Suc } i)) (S i)$$

**obtain** *s p* **where** *s* = (*λi. fst* (*S i*)) **and** *p* = (*λi. snd* (*S i*))

**by** *auto*

**with** *inf*

**have** *inf'*:

$$\forall i. \Gamma \vdash (\text{the } (\Gamma \ (p \ i))) \downarrow \text{Normal } (s \ i) \wedge \\ (\exists \text{css}. \Gamma \vdash ([\text{the } (\Gamma \ (p \ i))], [], \text{Normal } (s \ i)) \rightarrow^+ \\ ([\text{the } (\Gamma \ (p \ (\text{Suc } i))], \text{css}, \text{Normal } (s \ (\text{Suc } i))))$$

**apply** –

**apply** (*rule allI*)

**apply** (*erule-tac x=i in allE*)

**apply** *auto*

**done**

**show** *False*

**proof** –

**from** *inf'* — Skolemization of *css* with axiom of choice

$$\text{have } \exists \text{css}. \forall i. \Gamma \vdash (\text{the } (\Gamma \ (p \ i))) \downarrow \text{Normal } (s \ i) \wedge \\ \Gamma \vdash ([\text{the } (\Gamma \ (p \ i))], [], \text{Normal } (s \ i)) \rightarrow^+ \\ ([\text{the } (\Gamma \ (p \ (\text{Suc } i))], \text{css } i, \text{Normal } (s \ (\text{Suc } i)))$$

**apply** –

**apply** (*rule choice*)

**by** *blast*

**then obtain** *css* **where**

*termi-css*:  $\forall i. \Gamma \vdash (\text{the } (\Gamma \ (p \ i))) \downarrow \text{Normal } (s \ i)$  **and**

*step-css*:  $\forall i. \Gamma \vdash ([\text{the } (\Gamma \ (p \ i))], [], \text{Normal } (s \ i)) \rightarrow^+ \\ ([\text{the } (\Gamma \ (p \ (\text{Suc } i))], \text{css } i, \text{Normal } (s \ (\text{Suc } i)))$

by *blast*  
**define**  $f$  **where**  $f\ i = ([the\ (\Gamma\ (p\ i))],\ seq\ css\ i, Normal\ (s\ i)::('a,'c)\ xstate)$  **for**  
*i*  
**have**  $f\ 0 = ([the\ (\Gamma\ (p\ 0))], [], Normal\ (s\ 0))$   
 by (*simp add: f-def*)  
**moreover**  
**have**  $\forall i. \Gamma \vdash (f\ i) \rightarrow^+ (f\ (i+1))$   
**proof**  
**fix**  $i$   
**from** *step-css* [*rule-format, of i*]  
**have**  $\Gamma \vdash ([the\ (\Gamma\ (p\ i))], [], Normal\ (s\ i)) \rightarrow^+$   
 $([the\ (\Gamma\ (p\ (Suc\ i))], css\ i, Normal\ (s\ (Suc\ i)))$ .  
**from** *app-css-steps* [*OF this, simplified*]  
**have**  $\Gamma \vdash ([the\ (\Gamma\ (p\ i))], seq\ css\ i, Normal\ (s\ i)) \rightarrow^+$   
 $([the\ (\Gamma\ (p\ (Suc\ i))], css\ i@seq\ css\ i, Normal\ (s\ (Suc\ i)))$ .  
**thus**  $\Gamma \vdash (f\ i) \rightarrow^+ (f\ (i+1))$   
 by (*simp add: f-def*)  
**qed**  
**moreover from** *termi-css* [*rule-format, of 0*]  
**have**  $\neg (\exists f. (f\ 0 = ([the\ (\Gamma\ (p\ 0))], [], Normal\ (s\ 0)) \wedge$   
 $(\forall i. \Gamma \vdash (f\ i) \rightarrow^+ f\ (Suc\ i)))$   
 by (*rule terminates-impl-no-inf-chain*)  
**ultimately show** *False*  
 by *auto*  
**qed**  
**qed**

An alternative proof using Hilbert-choice instead of axiom of choice.

**theorem** *wf* (*termi-call-steps*  $\Gamma$ )  
**proof** (*simp only: termi-call-steps-def wf-iff-no-infinite-down-chain,*  
*clarify, simp*)  
**fix**  $S$   
**assume** *inf*:  
 $\forall i. (\lambda(t,q)\ (s,p). \Gamma \vdash (the\ (\Gamma\ p)) \downarrow Normal\ s \wedge$   
 $(\exists css. \Gamma \vdash ([the\ (\Gamma\ p)], [], Normal\ s) \rightarrow^+ ([the\ (\Gamma\ q)], css, Normal\ t)))$   
 $(S\ (Suc\ i))\ (S\ i)$   
**obtain**  $s\ p$  **where**  $s = (\lambda i. fst\ (S\ i))$  **and**  $p = (\lambda i. snd\ (S\ i))$   
 by *auto*  
**with** *inf*  
**have** *inf'*:  
 $\forall i. \Gamma \vdash (the\ (\Gamma\ (p\ i))) \downarrow Normal\ (s\ i) \wedge$   
 $(\exists css. \Gamma \vdash ([the\ (\Gamma\ (p\ i))], [], Normal\ (s\ i)) \rightarrow^+$   
 $([the\ (\Gamma\ (p\ (Suc\ i))], css, Normal\ (s\ (Suc\ i)))$   
**apply** –  
**apply** (*rule allI*)  
**apply** (*erule-tac x=i in allE*)  
**apply** *auto*  
**done**  
**show** *False*

**proof** –  
**define** *CSS* **where** *CSS* *i* = (*SOME* *css*.  
 $\Gamma \vdash ([the (\Gamma (p\ i))], [], Normal (s\ i)) \rightarrow^+$   
 $([the (\Gamma (p\ (i+1)))]], css, Normal (s\ (i+1))))$  **for** *i*  
**define** *f* **where** *f* *i* = ( $[the (\Gamma (p\ i))], seq\ CSS\ i, Normal (s\ i)::('a, 'c)\ xstate$ )  
**for** *i*  
**have** *f* 0 = ( $[the (\Gamma (p\ 0))], [], Normal (s\ 0)$ )  
**by** (*simp* *add*: *f-def*)  
**moreover**  
**have**  $\forall i. \Gamma \vdash (f\ i) \rightarrow^+ (f\ (i+1))$   
**proof**  
**fix** *i*  
**from** *inf'* [*rule-format*, *of* *i*] **obtain** *css* **where**  
 $css: \Gamma \vdash ([the (\Gamma (p\ i))], [], Normal (s\ i)) \rightarrow^+$   
 $([the (\Gamma (p\ (i+1)))]], css, Normal (s\ (i+1)))$   
**by** *fastforce*  
**hence**  $\Gamma \vdash ([the (\Gamma (p\ i))], seq\ CSS\ i, Normal (s\ i)) \rightarrow^+$   
 $([the (\Gamma (p\ (i+1)))]], CSS\ i\ @\ seq\ CSS\ i, Normal (s\ (i+1)))$   
**apply** –  
**apply** (*unfold* *CSS-def*)  
**apply** (*rule* *someI2*)  
**[where** *P* =  $\lambda css.$   
 $\Gamma \vdash ([the (\Gamma (p\ i))], [], Normal (s\ i)) \rightarrow^+$   
 $([the (\Gamma (p\ (i+1)))]], css, Normal (s\ (i+1)))$   
**apply** (*rule* *css*)  
**apply** (*fastforce* *dest*: *app-css-steps*)  
**done**  
**thus**  $\Gamma \vdash (f\ i) \rightarrow^+ (f\ (i+1))$   
**by** (*simp* *add*: *f-def*)  
**qed**  
**moreover from** *inf'* [*rule-format*, *of* 0]  
**have**  $\Gamma \vdash the\ (\Gamma\ (p\ 0))\ \downarrow\ Normal\ (s\ 0)$   
**by** *iprover*  
**then have**  $\neg (\exists f. (f\ 0 = ([the (\Gamma (p\ 0))], [], Normal (s\ 0)) \wedge$   
 $(\forall i. \Gamma \vdash (f\ i) \rightarrow^+ f\ (Suc\ i))))$   
**by** (*rule* *terminates-impl-no-inf-chain*)  
**ultimately show** *False*  
**by** *auto*  
**qed**  
**qed**

**lemma** *not-inf-implies-wf*: **assumes** *not-inf*:  $\neg\ inf\ \Gamma\ cs\ css\ s$   
**shows** *wf*  $\{(c2, c1). \Gamma \vdash (cs, css, s) \rightarrow^* c1 \wedge \Gamma \vdash c1 \rightarrow c2\}$   
**proof** (*simp* *only*: *wf-iff-no-infinite-down-chain*, *clarify*, *simp*)  
**fix** *f*  
**assume**  $\forall i. \Gamma \vdash (cs, css, s) \rightarrow^* f\ i \wedge \Gamma \vdash f\ i \rightarrow f\ (Suc\ i)$   
**hence**  $\exists f. f\ 0 = (cs, css, s) \wedge (\forall i. \Gamma \vdash f\ i \rightarrow f\ (Suc\ i))$   
**by** (*rule* *renumber* [*to-pred*])  
**moreover from** *not-inf*

```

have  $\neg (\exists f. f\ 0 = (cs, css, s) \wedge (\forall i. \Gamma \vdash f\ i \rightarrow f\ (Suc\ i)))$ 
  by (unfold inf-def)
ultimately show False
  by simp
qed

lemma wf-implies-termi-reach:
assumes wf:  $wf\ \{(c2, c1). \Gamma \vdash (cs, css, s) \rightarrow^* c1 \wedge \Gamma \vdash c1 \rightarrow c2\}$ 
shows  $\bigwedge cs1\ css1\ s1. \llbracket \Gamma \vdash (cs, css, s) \rightarrow^* c1; c1 = (cs1, css1, s1) \rrbracket \Longrightarrow \Gamma \vdash cs1, css1 \Downarrow s1$ 
using wf
proof (induct c1, simp)
  fix cs1 css1 s1
  assume reach:  $\Gamma \vdash (cs, css, s) \rightarrow^* (cs1, css1, s1)$ 
  assume hyp-raw:  $\bigwedge y\ cs2\ css2\ s2. \llbracket \Gamma \vdash (cs1, css1, s1) \rightarrow (cs2, css2, s2);$ 
     $\Gamma \vdash (cs, css, s) \rightarrow^* (cs2, css2, s2); y = (cs2, css2, s2) \rrbracket \Longrightarrow$ 
     $\Gamma \vdash cs2, css2 \Downarrow s2$ 
  have hyp:  $\bigwedge cs2\ css2\ s2. \llbracket \Gamma \vdash (cs1, css1, s1) \rightarrow (cs2, css2, s2) \rrbracket \Longrightarrow$ 
     $\Gamma \vdash cs2, css2 \Downarrow s2$ 

  apply  $-$ 
  apply (rule hyp-raw)
  apply assumption
  using reach
  apply simp
  apply (rule refl)
  done
show  $\Gamma \vdash cs1, css1 \Downarrow s1$ 
proof (cases s1)
  case (Normal s1')
  show ?thesis
  proof (cases cs1)
  case Nil
  note cs1-Nil = this
  show ?thesis
  proof (cases css1)
  case Nil
  with cs1-Nil show ?thesis
    by (auto intro: terminatess.intros)
  next
  case (Cons nrms-abrs css1')
  then obtain nrms abrs where nrms-abrs: nrms-abrs = (nrms, abrs)
    by (cases nrms-abrs)
  have  $\Gamma \vdash (\llbracket, (nrms, abrs) \# css1', Normal\ s1' \rrbracket \rightarrow (nrms, css1', Normal\ s1')$ 
    by (rule step.intros)
  from hyp [simplified cs1-Nil Cons nrms-abrs Normal, OF this]
  have  $\Gamma \vdash nrms, css1' \Downarrow Normal\ s1'$ .
  from ExitBlockNormal [OF this] cs1-Nil Cons nrms-abrs Normal
  show ?thesis
    by auto
qed

```

```

next
case (Cons c1 cs1')
have  $\Gamma \vdash c1 \# cs1', css1 \Downarrow Normal s1'$ 
proof (cases c1)
  case Skip
  have  $\Gamma \vdash (Skip \# cs1', css1, Normal s1') \rightarrow (cs1', css1, Normal s1')$ 
    by (rule step.intros)
  from hyp [simplified Cons Skip Normal, OF this]
  have  $\Gamma \vdash cs1', css1 \Downarrow Normal s1'$ .
  with Normal Skip show ?thesis
    by (auto intro: terminatess.intros terminates.intros
        elim: exec-Normal-elim-cases)
next
case (Basic f)
have  $\Gamma \vdash (Basic f \# cs1', css1, Normal s1') \rightarrow (cs1', css1, Normal (f s1'))$ 
  by (rule step.intros)
from hyp [simplified Cons Basic Normal, OF this]
have  $\Gamma \vdash cs1', css1 \Downarrow Normal (f s1')$ .
with Normal Basic show ?thesis
  by (auto intro: terminatess.intros terminates.intros
      elim: exec-Normal-elim-cases)
next
case (Spec r)
with Normal show ?thesis
  apply simp
  apply (rule terminatess.Cons)
  apply (fastforce intro: terminates.intros)
  apply (clarify)
  apply (erule exec-Normal-elim-cases)
  apply clarsimp
  apply (rule hyp)
  apply (fastforce intro: step.intros simp add: Cons Spec Normal)
  apply (fastforce intro: terminatess-Stuck)
  done
next
case (Seq c1 c2)
have  $\Gamma \vdash (Seq c1 c2 \# cs1', css1, Normal s1') \rightarrow (c1 \# c2 \# cs1', css1, Normal$ 
s1')
  by (rule step.intros)
from hyp [simplified Cons Seq Normal, OF this]
have  $\Gamma \vdash c1 \# c2 \# cs1', css1 \Downarrow Normal s1'$ .
with Normal Seq show ?thesis
  by (fastforce intro: terminatess.intros terminates.intros
      elim: terminatess-elim-cases exec-Normal-elim-cases)
next
case (Cond b c1 c2)
show ?thesis
proof (cases s1'  $\in$  b)
  case True

```

```

hence  $\Gamma \vdash (\text{Cond } b \ c_1 \ c_2 \# \text{cs1}', \text{css1}, \text{Normal } s1') \rightarrow (c_1 \# \text{cs1}', \text{css1}, \text{Normal } s1')$ 
by (rule step.intros)
from hyp [simplified Cons Cond Normal, OF this]
have  $\Gamma \vdash c_1 \# \text{cs1}', \text{css1} \Downarrow \text{Normal } s1'$ .
with Normal Cond True show ?thesis
by (fastforce intro: terminatess.intros terminates.intros
      elim: terminatess-elim-cases exec-Normal-elim-cases)
next
case False
hence  $\Gamma \vdash (\text{Cond } b \ c_1 \ c_2 \# \text{cs1}', \text{css1}, \text{Normal } s1') \rightarrow (c_2 \# \text{cs1}', \text{css1}, \text{Normal } s1')$ 
by (rule step.intros)
from hyp [simplified Cons Cond Normal, OF this]
have  $\Gamma \vdash c_2 \# \text{cs1}', \text{css1} \Downarrow \text{Normal } s1'$ .
with Normal Cond False show ?thesis
by (fastforce intro: terminatess.intros terminates.intros
      elim: terminatess-elim-cases exec-Normal-elim-cases)
qed
next
case (While b c')
show ?thesis
proof (cases s1' ∈ b)
case True
then have  $\Gamma \vdash (\text{While } b \ c' \# \text{cs1}', \text{css1}, \text{Normal } s1') \rightarrow (c' \# \text{While } b \ c' \# \text{cs1}', \text{css1}, \text{Normal } s1')$ 
by (rule step.intros)
from hyp [simplified Cons While Normal, OF this]
have  $\Gamma \vdash c' \# \text{While } b \ c' \# \text{cs1}', \text{css1} \Downarrow \text{Normal } s1'$ .
with Cons While True Normal
show ?thesis
by (fastforce intro: terminatess.intros terminates.intros exec.intros
      elim: terminatess-elim-cases exec-Normal-elim-cases)
next
case False
then
have  $\Gamma \vdash (\text{While } b \ c' \# \text{cs1}', \text{css1}, \text{Normal } s1') \rightarrow (c1', \text{css1}, \text{Normal } s1')$ 
by (rule step.intros)
from hyp [simplified Cons While Normal, OF this]
have  $\Gamma \vdash c1', \text{css1} \Downarrow \text{Normal } s1'$ .
with Cons While False Normal
show ?thesis
by (fastforce intro: terminatess.intros terminates.intros exec.intros
      elim: terminatess-elim-cases exec-Normal-elim-cases)
qed
next
case (Call p)
show ?thesis
proof (cases Γ p)

```

```

case None
with Call Normal show ?thesis
by (fastforce intro: terminatess.intros terminates.intros terminatess-Stuck
      elim: exec-Normal-elim-cases)
next
case (Some bdy)
then
have  $\Gamma \vdash (\text{Call } p \# cs1', css1, \text{Normal } s1') \rightarrow$ 
      ( $[bdy], (cs1', \text{Throw} \# cs1') \# css1, \text{Normal } s1'$ )
by (rule step.intros)
from hyp [simplified Cons Call Normal Some, OF this]
have  $\Gamma \vdash [bdy], (cs1', \text{Throw} \# cs1') \# css1 \Downarrow \text{Normal } s1'$ .
with Some Call Normal show ?thesis
apply simp
apply (rule terminatess.intros)
apply (blast elim: terminatess-elim-cases intro: terminates.intros)
apply clarify
apply (erule terminatess-elim-cases)
apply (erule exec-Normal-elim-cases)
prefer 2
apply simp
apply (erule-tac x=t in allE)
apply (case-tac t)
apply (auto intro: terminatess-Stuck terminatess-Fault exec.intros
      elim: terminatess-elim-cases exec-Normal-elim-cases)
done
qed
next
case (DynCom c')
have  $\Gamma \vdash (\text{DynCom } c' \# cs1', css1, \text{Normal } s1') \rightarrow (c' \ s1' \# cs1', css1, \text{Normal}$ 
s1')
by (rule step.intros)
from hyp [simplified Cons DynCom Normal, OF this]
have  $\Gamma \vdash c' \ s1' \# cs1', css1 \Downarrow \text{Normal } s1'$ .
with Normal DynCom show ?thesis
by (fastforce intro: terminatess.intros terminates.intros exec.intros
      elim: terminatess-elim-cases exec-Normal-elim-cases)
next
case (Guard f g c')
show ?thesis
proof (cases s1' ∈ g)
case True
then have  $\Gamma \vdash (\text{Guard } f g \ c' \# cs1', css1, \text{Normal } s1') \rightarrow (c' \# cs1', css1, \text{Normal}$ 
s1')
by (rule step.intros)
from hyp [simplified Cons Guard Normal, OF this]
have  $\Gamma \vdash c' \# cs1', css1 \Downarrow \text{Normal } s1'$ .
with Normal Guard True show ?thesis
by (fastforce intro: terminatess.intros terminates.intros exec.intros)

```



```

      elim: terminatess-elim-cases exec-Normal-elim-cases)
next
  case False
  with Guard Normal show ?thesis
  by (fastforce intro: terminatess.intros terminatess-Fault
      terminatess.intros
      elim: exec-Normal-elim-cases)
qed
next
  case Throw
  have  $\Gamma \vdash (Throw \# cs1', css1, Normal s1') \rightarrow (cs1', css1, Abrupt s1')$ 
  by (rule step.intros)
  from hyp [simplified Cons Throw Normal, OF this]
  have  $\Gamma \vdash cs1', css1 \Downarrow Abrupt s1'$ .
  with Normal Throw show ?thesis
  by (auto intro: terminatess.intros terminatess.intros
      elim: exec-Normal-elim-cases)
next
  case (Catch c1 c2)
  have  $\Gamma \vdash (Catch c1 c2 \# cs1', css1, Normal s1') \rightarrow$ 
     $([c1], (cs1', c2 \# cs1') \# css1, Normal s1')$ 
  by (rule step.intros)
  from hyp [simplified Cons Catch Normal, OF this]
  have  $\Gamma \vdash [c1], (cs1', c2 \# cs1') \# css1 \Downarrow Normal s1'$ .
  with Normal Catch show ?thesis
  by (fastforce intro: terminatess.intros terminatess.intros exec.intros
      elim: terminatess-elim-cases exec-Normal-elim-cases)
qed
with Cons Normal show ?thesis
  by simp
qed
next
  case (Abrupt s1')
  show ?thesis
  proof (cases cs1)
  case Nil
  note  $cs1-Nil = this$ 
  show ?thesis
  proof (cases css1)
  case Nil
  with  $cs1-Nil$  show ?thesis by (auto intro: terminatess.intros)
  next
  case (Cons nrms-abrs css1')
  then obtain nrms abrs where  $nrms-abrs: nrms-abrs = (nrms, abrs)$ 
  by (cases nrms-abrs)
  have  $\Gamma \vdash ([], (nrms, abrs) \# css1', Abrupt s1') \rightarrow (abrs, css1', Normal s1')$ 
  by (rule step.intros)
  from hyp [simplified  $cs1-Nil$  Cons nrms-abrs Abrupt, OF this]
  have  $\Gamma \vdash abrs, css1' \Downarrow Normal s1'$ .

```

```

    from ExitBlockAbrupt [OF this] cs1-Nil Cons nrms-abrs Abrupt
    show ?thesis
      by auto
  qed
next
case (Cons c1 cs1')
have  $\Gamma \vdash c1 \# cs1', css1 \Downarrow Abrupt s1'$ 
proof -
  have  $\Gamma \vdash (c1 \# cs1', css1, Abrupt s1') \rightarrow (cs1', css1, Abrupt s1')$ 
    by (rule step.intros)
  from hyp [simplified Cons Abrupt, OF this]
  have  $\Gamma \vdash cs1', css1 \Downarrow Abrupt s1'$ .
  with Cons Abrupt
  show ?thesis
    by (fastforce intro: terminatess.intros terminates.intros exec.intros
      elim: terminatess-elim-cases exec-Normal-elim-cases)
  qed
with Cons Abrupt show ?thesis by simp
qed
next
case (Fault f)
thus ?thesis by (auto intro: terminatess-Fault)
next
case Stuck
thus ?thesis by (auto intro: terminatess-Stuck)
qed
qed

lemma not-inf-impl-terminatess:
  assumes not-inf:  $\neg inf \Gamma cs css s$ 
  shows  $\Gamma \vdash cs, css \Downarrow s$ 
proof -
  from not-inf-implies-wf [OF not-inf]
  have wf:  $wf \{(c2, c1). \Gamma \vdash (cs, css, s) \rightarrow^* c1 \wedge \Gamma \vdash c1 \rightarrow c2\}$ .
  show ?thesis
    by (rule wf-implies-termi-reach [OF wf]) auto
qed

lemma not-inf-impl-terminates:
  assumes not-inf:  $\neg inf \Gamma [c] [] s$ 
  shows  $\Gamma \vdash c \Downarrow s$ 
proof -
  from not-inf-impl-terminatess [OF not-inf]
  have  $\Gamma \vdash [c], [] \Downarrow s$ .
  thus ?thesis
    by (auto elim: terminatess-elim-cases)
qed

theorem terminatess-iff-not-inf:

```

$\Gamma \vdash cs, css \Downarrow s = (\neg \text{inf } \Gamma \text{ cs css } s)$   
**apply rule**  
**apply** (*erule terminatess-impl-not-inf*)  
**apply** (*erule not-inf-impl-terminatess*)  
**done**

**corollary** *terminates-iff-not-inf*:  
 $\Gamma \vdash c \Downarrow s = (\neg \text{inf } \Gamma [c] [] s)$   
**apply** (*rule*)  
**apply** (*erule terminates-impl-not-inf*)  
**apply** (*erule not-inf-impl-terminates*)  
**done**

## 14.4 Completeness of Total Correctness Hoare Logic

**lemma** *ConseqMGT*:  
**assumes** *modif*:  $\forall Z :: 'a. \Gamma, \Theta \vdash_{t/F} (P' Z :: 'a \text{ assn}) \ c \ (Q' Z), (A' Z)$   
**assumes** *impl*:  $\bigwedge s. s \in P \implies s \in P' s \wedge (\forall t. t \in Q' s \longrightarrow t \in Q) \wedge$   
 $(\forall t. t \in A' s \longrightarrow t \in A)$   
**shows**  $\Gamma, \Theta \vdash_{t/F} P \ c \ Q, A$   
**using** *impl*  
**by** – (*rule conseq [OF modif], blast*)

**lemma** *conseq-extract-state-indep-prop*:  
**assumes** *state-indep-prop*:  $\forall s \in P. R$   
**assumes** *to-show*:  $R \implies \Gamma, \Theta \vdash_{t/F} P \ c \ Q, A$   
**shows**  $\Gamma, \Theta \vdash_{t/F} P \ c \ Q, A$   
**apply** (*rule Conseq*)  
**apply** (*clarify*)  
**apply** (*rule-tac x=P in exI*)  
**apply** (*rule-tac x=Q in exI*)  
**apply** (*rule-tac x=A in exI*)  
**using** *state-indep-prop to-show*  
**by** *blast*

**lemma** *Call-lemma'*:  
**assumes** *Call-hyp*:  
 $\forall q \in \text{dom } \Gamma. \forall Z. \Gamma, \Theta \vdash_{t/F} \{s. s=Z \wedge \Gamma \vdash \langle \text{Call } q, \text{Normal } s \rangle \Rightarrow \notin (\{\text{Stuck}\} \cup \text{Fault } '(-F)) \wedge$   
 $\Gamma \vdash \text{Call } q \Downarrow \text{Normal } s \wedge ((s, q), (\sigma, p)) \in \text{termi-call-steps } \Gamma \}$   
 $(\text{Call } q)$   
 $\{t. \Gamma \vdash \langle \text{Call } q, \text{Normal } Z \rangle \Rightarrow \text{Normal } t\},$   
 $\{t. \Gamma \vdash \langle \text{Call } q, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\}$   
**shows**  $\bigwedge Z. \Gamma, \Theta \vdash_{t/F}$   
 $\{s. s=Z \wedge \Gamma \vdash \langle c, \text{Normal } s \rangle \Rightarrow \notin (\{\text{Stuck}\} \cup \text{Fault } '(-F)) \wedge \Gamma \vdash \text{the } (\Gamma \ p) \Downarrow \text{Normal}$   
 $\sigma \wedge$   
 $(\exists cs \ \text{css}. \Gamma \vdash ([\text{the } (\Gamma \ p)], [], \text{Normal } \sigma) \rightarrow^* (c \# cs, \text{css}, \text{Normal } s))\}$   
 $c$

$\{t. \Gamma \vdash \langle c, Normal Z \rangle \Rightarrow Normal t\},$   
 $\{t. \Gamma \vdash \langle c, Normal Z \rangle \Rightarrow Abrupt t\}$

**proof** (*induct c*)

**case** *Skip*

**show**  $\Gamma, \Theta \vdash_{t/F} \{s. s = Z \wedge \Gamma \vdash \langle Skip, Normal s \rangle \Rightarrow \notin(\{Stuck\} \cup Fault \text{ ' } (-F)) \wedge$   
 $\Gamma \vdash the (\Gamma p) \downarrow Normal \sigma \wedge$   
 $(\exists cs css. \Gamma \vdash ([the (\Gamma p)], [], Normal \sigma) \rightarrow^* (Skip \# cs, css, Normal s))\}$   
 $Skip$   
 $\{t. \Gamma \vdash \langle Skip, Normal Z \rangle \Rightarrow Normal t\},$   
 $\{t. \Gamma \vdash \langle Skip, Normal Z \rangle \Rightarrow Abrupt t\}$

**by** (*rule hoaret.Skip [THEN conseqPre]*) (*blast intro: exec.Skip*)

**next**

**case** (*Basic f*)

**show**  $\Gamma, \Theta \vdash_{t/F} \{s. s = Z \wedge \Gamma \vdash \langle Basic f, Normal s \rangle \Rightarrow \notin(\{Stuck\} \cup Fault \text{ ' } (-F)) \wedge$   
 $\Gamma \vdash the (\Gamma p) \downarrow Normal \sigma \wedge$   
 $(\exists cs css. \Gamma \vdash ([the (\Gamma p)], [], Normal \sigma) \rightarrow^* (Basic f \# cs, css, Normal s))\}$   
 $Basic f$   
 $\{t. \Gamma \vdash \langle Basic f, Normal Z \rangle \Rightarrow Normal t\},$   
 $\{t. \Gamma \vdash \langle Basic f, Normal Z \rangle \Rightarrow Abrupt t\}$

**by** (*rule hoaret.Basic [THEN conseqPre]*) (*blast intro: exec.Basic*)

**next**

**case** (*Spec r*)

**show**  $\Gamma, \Theta \vdash_{t/F} \{s. s = Z \wedge \Gamma \vdash \langle Spec r, Normal s \rangle \Rightarrow \notin(\{Stuck\} \cup Fault \text{ ' } (-F)) \wedge$   
 $\Gamma \vdash the (\Gamma p) \downarrow Normal \sigma \wedge$   
 $(\exists cs css. \Gamma \vdash ([the (\Gamma p)], [], Normal \sigma) \rightarrow^* (Spec r \# cs, css, Normal s))\}$   
 $Spec r$   
 $\{t. \Gamma \vdash \langle Spec r, Normal Z \rangle \Rightarrow Normal t\},$   
 $\{t. \Gamma \vdash \langle Spec r, Normal Z \rangle \Rightarrow Abrupt t\}$

**apply** (*rule hoaret.Spec [THEN conseqPre]*)

**apply** (*clarsimp*)

**apply** (*case-tac*  $\exists t. (Z, t) \in r$ )

**apply** (*auto elim: exec-elim-cases simp add: final-notin-def intro: exec.intros*)

**done**

**next**

**case** (*Seq c1 c2*)

**have** *hyp-c1*:

$\forall Z. \Gamma, \Theta \vdash_{t/F} \{s. s = Z \wedge \Gamma \vdash \langle c1, Normal s \rangle \Rightarrow \notin(\{Stuck\} \cup Fault \text{ ' } (-F)) \wedge$   
 $\Gamma \vdash the (\Gamma p) \downarrow Normal \sigma \wedge$   
 $(\exists cs css. \Gamma \vdash ([the (\Gamma p)], [], Normal \sigma) \rightarrow^* (c1 \# cs, css, Normal s))\}$   
 $c1$   
 $\{t. \Gamma \vdash \langle c1, Normal Z \rangle \Rightarrow Normal t\},$   
 $\{t. \Gamma \vdash \langle c1, Normal Z \rangle \Rightarrow Abrupt t\}$

**using** *Seq.hyps* **by** *iprover*

**have** *hyp-c2*:

$\forall Z. \Gamma, \Theta \vdash_{t/F} \{s. s = Z \wedge \Gamma \vdash \langle c2, Normal s \rangle \Rightarrow \notin(\{Stuck\} \cup Fault \text{ ' } (-F)) \wedge$   
 $\Gamma \vdash the (\Gamma p) \downarrow Normal \sigma \wedge$   
 $(\exists cs css. \Gamma \vdash ([the (\Gamma p)], [], Normal \sigma) \rightarrow^* (c2 \# cs, css, Normal s))\}$   
 $c2$

$\{t. \Gamma \langle c2, Normal Z \rangle \Rightarrow Normal t\},$   
 $\{t. \Gamma \langle c2, Normal Z \rangle \Rightarrow Abrupt t\}$   
**using** *Seq.hyps* **by** *iprover*  
**have**  $c1: \Gamma, \Theta \vdash_{t/F} \{s. s=Z \wedge \Gamma \langle Seq\ c1\ c2, Normal\ s \rangle \Rightarrow \notin(\{Stuck\} \cup Fault\ '(-F)) \wedge$   
 $(-F) \wedge$   
 $\Gamma \vdash the\ (\Gamma\ p) \downarrow Normal\ \sigma \wedge$   
 $(\exists\ cs\ css. \Gamma \vdash ([the\ (\Gamma\ p)], [], Normal\ \sigma) \rightarrow^* (Seq\ c1\ c2 \# cs, css, Normal\ s))\}$   
 $c1$   
 $\{t. \Gamma \langle c1, Normal Z \rangle \Rightarrow Normal t \wedge$   
 $\Gamma \langle c2, Normal t \rangle \Rightarrow \notin(\{Stuck\} \cup Fault\ '(-F)) \wedge$   
 $\Gamma \vdash the\ (\Gamma\ p) \downarrow Normal\ \sigma \wedge$   
 $(\exists\ cs\ css. \Gamma \vdash ([the\ (\Gamma\ p)], [], Normal\ \sigma) \rightarrow^* (c2 \# cs, css, Normal\ t))\},$   
 $\{t. \Gamma \langle Seq\ c1\ c2, Normal Z \rangle \Rightarrow Abrupt t\}$   
**proof** (*rule ConseqMGT [OF hyp-c1], clarify, safe*)  
**assume**  $\Gamma \langle Seq\ c1\ c2, Normal Z \rangle \Rightarrow \notin(\{Stuck\} \cup Fault\ '(-F))$   
**thus**  $\Gamma \langle c1, Normal Z \rangle \Rightarrow \notin(\{Stuck\} \cup Fault\ '(-F))$   
**by** (*blast dest: Seq-NoFaultStuckD1*)  
**next**  
**fix**  $cs\ css$   
**assume**  $\Gamma \vdash ([the\ (\Gamma\ p)], [], Normal\ \sigma) \rightarrow^* (Seq\ c1\ c2 \# cs, css, Normal\ Z)$   
**thus**  $\exists\ cs\ css. \Gamma \vdash ([the\ (\Gamma\ p)], [], Normal\ \sigma) \rightarrow^* (c1 \# cs, css, Normal\ Z)$   
**by** (*blast intro: rtranclp-into-tranclp1 [THEN tranclp-into-rtranclp] step.Seq*)  
**next**  
**fix**  $t$   
**assume**  $\Gamma \langle Seq\ c1\ c2, Normal Z \rangle \Rightarrow \notin(\{Stuck\} \cup Fault\ '(-F))$   
 $\Gamma \langle c1, Normal Z \rangle \Rightarrow Normal\ t$   
**thus**  $\Gamma \langle c2, Normal t \rangle \Rightarrow \notin(\{Stuck\} \cup Fault\ '(-F))$   
**by** (*blast dest: Seq-NoFaultStuckD2*)  
**next**  
**fix**  $cs\ css\ t$   
**assume**  $\Gamma \vdash ([the\ (\Gamma\ p)], [], Normal\ \sigma) \rightarrow^* (Seq\ c1\ c2 \# cs, css, Normal\ Z)$   
**also have**  $\Gamma \vdash (Seq\ c1\ c2 \# cs, css, Normal\ Z) \rightarrow (c1 \# c2 \# cs, css, Normal\ Z)$   
**by** (*rule step.Seq*)  
**also assume**  $\Gamma \langle c1, Normal Z \rangle \Rightarrow Normal\ t$   
**hence**  $\Gamma \vdash (c1 \# c2 \# cs, css, Normal\ Z) \rightarrow^* (c2 \# cs, css, Normal\ t)$   
**by** (*rule exec-impl-steps*)  
**finally**  
**show**  $\exists\ cs\ css. \Gamma \vdash ([the\ (\Gamma\ p)], [], Normal\ \sigma) \rightarrow^* (c2 \# cs, css, Normal\ t)$   
**by** *iprover*  
**next**  
**fix**  $t$   
**assume**  $\Gamma \langle c1, Normal Z \rangle \Rightarrow Abrupt\ t$   
**thus**  $\Gamma \langle Seq\ c1\ c2, Normal Z \rangle \Rightarrow Abrupt\ t$   
**by** (*blast intro: exec.intros*)  
**qed**  
**show**  $\Gamma, \Theta \vdash_{t/F} \{s. s=Z \wedge \Gamma \langle Seq\ c1\ c2, Normal\ s \rangle \Rightarrow \notin(\{Stuck\} \cup Fault\ '(-F))$   
 $\wedge$

$$\Gamma \vdash \text{the } (\Gamma \ p) \downarrow \text{Normal } \sigma \wedge$$

$$(\exists \text{ cs css. } \Gamma \vdash ([\text{the } (\Gamma \ p)], [], \text{Normal } \sigma) \rightarrow^* (\text{Seq } c1 \ c2 \# \text{cs, css, Normal } s))\}$$

$$\text{Seq } c1 \ c2$$

$$\{t. \Gamma \vdash \langle \text{Seq } c1 \ c2, \text{Normal } Z \rangle \Rightarrow \text{Normal } t\},$$

$$\{t. \Gamma \vdash \langle \text{Seq } c1 \ c2, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\}$$

**by** (*rule hoaret.Seq [OF c1 ConseqMGT [OF hyp-c2]]*)  
*(blast intro: exec.intros)*

**next**  
**case** (*Cond b c1 c2*)  
**have** *hyp-c1*:  

$$\forall Z. \Gamma, \Theta \vdash_{t/F} \{s. s=Z \wedge \Gamma \vdash \langle c1, \text{Normal } s \rangle \Rightarrow \notin (\{\text{Stuck}\} \cup \text{Fault } '(-F)) \wedge$$

$$\Gamma \vdash \text{the } (\Gamma \ p) \downarrow \text{Normal } \sigma \wedge$$

$$(\exists \text{ cs css. } \Gamma \vdash ([\text{the } (\Gamma \ p)], [], \text{Normal } \sigma) \rightarrow^* (c1 \# \text{cs, css, Normal } s))\}$$

$$c1$$

$$\{t. \Gamma \vdash \langle c1, \text{Normal } Z \rangle \Rightarrow \text{Normal } t\},$$

$$\{t. \Gamma \vdash \langle c1, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\}$$

**using** *Cond.hyps* **by** *iprover*  
**have**  

$$\Gamma, \Theta \vdash_{t/F} (\{s. s=Z \wedge \Gamma \vdash \langle \text{Cond } b \ c1 \ c2, \text{Normal } s \rangle \Rightarrow \notin (\{\text{Stuck}\} \cup \text{Fault } '(-F))$$

$$\wedge$$

$$\Gamma \vdash \text{the } (\Gamma \ p) \downarrow \text{Normal } \sigma \wedge$$

$$(\exists \text{ cs css. } \Gamma \vdash ([\text{the } (\Gamma \ p)], [], \text{Normal } \sigma) \rightarrow^* (\text{Cond } b \ c1 \ c2 \# \text{cs, css, Normal } s))\}$$

$$\cap b)$$

$$c1$$

$$\{t. \Gamma \vdash \langle \text{Cond } b \ c1 \ c2, \text{Normal } Z \rangle \Rightarrow \text{Normal } t\},$$

$$\{t. \Gamma \vdash \langle \text{Cond } b \ c1 \ c2, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\}$$

**proof** (*rule ConseqMGT [OF hyp-c1], safe*)  
**assume**  $Z \in b \ \Gamma \vdash \langle \text{Cond } b \ c1 \ c2, \text{Normal } Z \rangle \Rightarrow \notin (\{\text{Stuck}\} \cup \text{Fault } '(-F))$   
**thus**  $\Gamma \vdash \langle c1, \text{Normal } Z \rangle \Rightarrow \notin (\{\text{Stuck}\} \cup \text{Fault } '(-F))$   
**by** (*auto simp add: final-notin-def intro: exec.CondTrue*)

**next**  
**fix** *cs css*  
**assume**  $Z \in b$   

$$\Gamma \vdash ([\text{the } (\Gamma \ p)], [], \text{Normal } \sigma) \rightarrow^* (\text{Cond } b \ c1 \ c2 \ # \ \text{cs, css, Normal } Z)$$
**thus**  $\exists \text{ cs css. } \Gamma \vdash ([\text{the } (\Gamma \ p)], [], \text{Normal } \sigma) \rightarrow^* (c1 \ # \ \text{cs, css, Normal } Z)$   
**by** (*blast intro: rtranclp-into-tranclp1 [THEN tranclp-into-rtranclp]*)  
*step.CondTrue*

**next**  
**fix** *t* **assume**  $Z \in b \ \Gamma \vdash \langle c1, \text{Normal } Z \rangle \Rightarrow \text{Normal } t$   
**thus**  $\Gamma \vdash \langle \text{Cond } b \ c1 \ c2, \text{Normal } Z \rangle \Rightarrow \text{Normal } t$   
**by** (*blast intro: exec.CondTrue*)

**next**  
**fix** *t* **assume**  $Z \in b \ \Gamma \vdash \langle c1, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t$   
**thus**  $\Gamma \vdash \langle \text{Cond } b \ c1 \ c2, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t$   
**by** (*blast intro: exec.CondTrue*)

**qed**  
**moreover**

**have** *hyp-c2*:  
 $\forall Z. \Gamma, \Theta \vdash_{t/F} \{s. s=Z \wedge \Gamma \vdash \langle c2, Normal \ s \rangle \Rightarrow \notin (\{Stuck\} \cup Fault \ '(-F)) \wedge$   
 $\Gamma \vdash the \ (\Gamma \ p) \downarrow Normal \ \sigma \wedge$   
 $(\exists cs \ css. \Gamma \vdash ([the \ (\Gamma \ p)], [], Normal \ \sigma) \rightarrow^* (c2 \# cs, css, Normal \ s))\}$   
 $c2$   
 $\{t. \Gamma \vdash \langle c2, Normal \ Z \rangle \Rightarrow Normal \ t\},$   
 $\{t. \Gamma \vdash \langle c2, Normal \ Z \rangle \Rightarrow Abrupt \ t\}$   
**using** *Cond.hyps* **by** *iprover*  
**have**  
 $\Gamma, \Theta \vdash_{t/F} \{s. s=Z \wedge \Gamma \vdash \langle Cond \ b \ c1 \ c2, Normal \ s \rangle \Rightarrow \notin (\{Stuck\} \cup Fault \ '(-F))$   
 $\wedge$   
 $\Gamma \vdash the \ (\Gamma \ p) \downarrow Normal \ \sigma \wedge$   
 $(\exists cs \ css. \Gamma \vdash ([the \ (\Gamma \ p)], [], Normal \ \sigma) \rightarrow^* (Cond \ b \ c1 \ c2 \# cs, css, Normal$   
 $s))\}$   
 $\cap -b)$   
 $c2$   
 $\{t. \Gamma \vdash \langle Cond \ b \ c1 \ c2, Normal \ Z \rangle \Rightarrow Normal \ t\},$   
 $\{t. \Gamma \vdash \langle Cond \ b \ c1 \ c2, Normal \ Z \rangle \Rightarrow Abrupt \ t\}$   
**proof** (*rule ConseqMGT [OF hyp-c2], safe*)  
**assume**  $Z \notin b \ \Gamma \vdash \langle Cond \ b \ c1 \ c2, Normal \ Z \rangle \Rightarrow \notin (\{Stuck\} \cup Fault \ '(-F))$   
**thus**  $\Gamma \vdash \langle c2, Normal \ Z \rangle \Rightarrow \notin (\{Stuck\} \cup Fault \ '(-F))$   
**by** (*auto simp add: final-notin-def intro: exec.CondFalse*)  
**next**  
**fix** *cs css*  
**assume**  $Z \notin b$   
 $\Gamma \vdash ([the \ (\Gamma \ p)], [], Normal \ \sigma) \rightarrow^* (Cond \ b \ c1 \ c2 \# cs, css, Normal \ Z)$   
**thus**  $\exists cs \ css. \Gamma \vdash ([the \ (\Gamma \ p)], [], Normal \ \sigma) \rightarrow^* (c2 \# cs, css, Normal \ Z)$   
**by** (*blast intro: rtranclp-into-tranclp1 [THEN tranclp-into-rtranclp]*  
*step.CondFalse*)  
**next**  
**fix** *t* **assume**  $Z \notin b \ \Gamma \vdash \langle c2, Normal \ Z \rangle \Rightarrow Normal \ t$   
**thus**  $\Gamma \vdash \langle Cond \ b \ c1 \ c2, Normal \ Z \rangle \Rightarrow Normal \ t$   
**by** (*blast intro: exec.CondFalse*)  
**next**  
**fix** *t* **assume**  $Z \notin b \ \Gamma \vdash \langle c2, Normal \ Z \rangle \Rightarrow Abrupt \ t$   
**thus**  $\Gamma \vdash \langle Cond \ b \ c1 \ c2, Normal \ Z \rangle \Rightarrow Abrupt \ t$   
**by** (*blast intro: exec.CondFalse*)  
**qed**  
**ultimately**  
**show**  
 $\Gamma, \Theta \vdash_{t/F} \{s. s=Z \wedge \Gamma \vdash \langle Cond \ b \ c1 \ c2, Normal \ s \rangle \Rightarrow \notin (\{Stuck\} \cup Fault \ '(-F)) \wedge$   
 $\Gamma \vdash the \ (\Gamma \ p) \downarrow Normal \ \sigma \wedge$   
 $(\exists cs \ css. \Gamma \vdash ([the \ (\Gamma \ p)], [], Normal \ \sigma) \rightarrow^* (Cond \ b \ c1 \ c2 \# cs, css, Normal$   
 $s))\}$   
 $(Cond \ b \ c1 \ c2)$   
 $\{t. \Gamma \vdash \langle Cond \ b \ c1 \ c2, Normal \ Z \rangle \Rightarrow Normal \ t\},$   
 $\{t. \Gamma \vdash \langle Cond \ b \ c1 \ c2, Normal \ Z \rangle \Rightarrow Abrupt \ t\}$   
**by** (*rule hoaret.Cond*)  
**next**

**case** (*While*  $b\ c$ )  
**let**  $?unroll = (\{(s,t). s \in b \wedge \Gamma \vdash \langle c, Normal\ s \rangle \Rightarrow Normal\ t\})^*$   
**let**  $?P' = \lambda Z. \{t. (Z,t) \in ?unroll \wedge$   
 $(\forall e. (Z,e) \in ?unroll \longrightarrow e \in b$   
 $\longrightarrow \Gamma \vdash \langle c, Normal\ e \rangle \Rightarrow \notin(\{Stuck\} \cup Fault\ '(-F)) \wedge$   
 $(\forall u. \Gamma \vdash \langle c, Normal\ e \rangle \Rightarrow Abrupt\ u \longrightarrow$   
 $\Gamma \vdash \langle While\ b\ c, Normal\ Z \rangle \Rightarrow Abrupt\ u)) \wedge$   
 $\Gamma \vdash the\ (\Gamma\ p) \downarrow Normal\ \sigma \wedge$   
 $(\exists cs\ css. \Gamma \vdash ([the\ (\Gamma\ p)], [], Normal\ \sigma) \rightarrow^*$   
 $(While\ b\ c \# cs, css, Normal\ t))\}$   
**let**  $?A = \lambda Z. \{t. \Gamma \vdash \langle While\ b\ c, Normal\ Z \rangle \Rightarrow Abrupt\ t\}$   
**let**  $?r = \{(t,s). \Gamma \vdash (While\ b\ c) \downarrow Normal\ s \wedge s \in b \wedge$   
 $\Gamma \vdash \langle c, Normal\ s \rangle \Rightarrow Normal\ t\}$   
**show**  $\Gamma, \Theta \vdash_{t/F}$   
 $\{s. s = Z \wedge \Gamma \vdash \langle While\ b\ c, Normal\ s \rangle \Rightarrow \notin(\{Stuck\} \cup Fault\ '(-F)) \wedge$   
 $\Gamma \vdash the\ (\Gamma\ p) \downarrow Normal\ \sigma \wedge$   
 $(\exists cs\ css. \Gamma \vdash ([the\ (\Gamma\ p)], [], Normal\ \sigma) \rightarrow^*(While\ b\ c \# cs, css, Normal\ s))\}$   
 $(While\ b\ c)$   
 $\{t. \Gamma \vdash \langle While\ b\ c, Normal\ Z \rangle \Rightarrow Normal\ t\},$   
 $\{t. \Gamma \vdash \langle While\ b\ c, Normal\ Z \rangle \Rightarrow Abrupt\ t\}$   
**proof** (*rule* *ConseqMGT* [**where**  $?P' = \lambda Z. ?P'\ Z$   
**and**  $?Q' = \lambda Z. ?P'\ Z \cap -\ b$ ])  
**have** *wf-r*: *wf*  $?r$  **by** (*rule* *wf-terminates-while*)  
**show**  $\forall Z. \Gamma, \Theta \vdash_{t/F} (?P'\ Z) (While\ b\ c) (?P'\ Z \cap -\ b), (?A\ Z)$   
**proof** (*rule* *allI*, *rule* *hoaret.While* [*OF* *wf-r*])  
**fix**  $Z$   
**from** *While*  
**have** *hyp-c*:  $\forall Z. \Gamma, \Theta \vdash_{t/F}$   
 $\{s. s = Z \wedge \Gamma \vdash \langle c, Normal\ s \rangle \Rightarrow \notin(\{Stuck\} \cup Fault\ '(-F)) \wedge$   
 $\Gamma \vdash the\ (\Gamma\ p) \downarrow Normal\ \sigma \wedge$   
 $(\exists cs\ css. \Gamma \vdash ([the\ (\Gamma\ p)], [], Normal\ \sigma) \rightarrow^*(c \# cs, css, Normal\ s))\}$   
 $c$   
 $\{t. \Gamma \vdash \langle c, Normal\ Z \rangle \Rightarrow Normal\ t\},$   
 $\{t. \Gamma \vdash \langle c, Normal\ Z \rangle \Rightarrow Abrupt\ t\}$  **by** *iprover*  
**show**  $\forall \sigma. \Gamma, \Theta \vdash_{t/F} (\{\sigma\} \cap ?P'\ Z \cap b)\ c$   
 $(\{t. (t, \sigma) \in ?r\} \cap ?P'\ Z), (?A\ Z)$   
**proof** (*rule* *allI*, *rule* *ConseqMGT* [*OF* *hyp-c*])  
**fix**  $\tau\ s$   
**assume** *asm*:  $s \in \{\tau\} \cap$   
 $\{t. (Z, t) \in ?unroll \wedge$   
 $(\forall e. (Z,e) \in ?unroll \longrightarrow e \in b$   
 $\longrightarrow \Gamma \vdash \langle c, Normal\ e \rangle \Rightarrow \notin(\{Stuck\} \cup Fault\ '(-F)) \wedge$   
 $(\forall u. \Gamma \vdash \langle c, Normal\ e \rangle \Rightarrow Abrupt\ u \longrightarrow$   
 $\Gamma \vdash \langle While\ b\ c, Normal\ Z \rangle \Rightarrow Abrupt\ u)) \wedge$   
 $\Gamma \vdash the\ (\Gamma\ p) \downarrow Normal\ \sigma \wedge$   
 $(\exists cs\ css. \Gamma \vdash ([the\ (\Gamma\ p)], [], Normal\ \sigma) \rightarrow^*$   
 $(While\ b\ c \# cs, css, Normal\ t))\}$   
 $\cap b$   
**then obtain** *cs css* **where**



$s\text{-eq-}\tau$ :  $s = \tau$  **and**  
 $Z\text{-s-unroll}$ :  $(Z, s) \in ?\text{unroll}$  **and**  
 $\text{noabort}$ :  $\forall e. (Z, e) \in ?\text{unroll} \longrightarrow e \in b$   
 $\longrightarrow \Gamma \vdash \langle c, \text{Normal } e \rangle \Rightarrow \notin (\{\text{Stuck}\} \cup \text{Fault } '(-F)) \wedge$   
 $(\forall u. \Gamma \vdash \langle c, \text{Normal } e \rangle \Rightarrow \text{Abrupt } u \longrightarrow$   
 $\Gamma \vdash \langle \text{While } b \ c, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } u)$  **and**  
 $\text{termi}$ :  $\Gamma \vdash \text{the } (\Gamma \ p) \downarrow \text{Normal } \sigma$  **and**  
 $\text{reach}$ :  $\Gamma \vdash ([\text{the } (\Gamma \ p)], [], \text{Normal } \sigma) \rightarrow^*$   
 $(\text{While } b \ c \# \text{cs, css, Normal } s)$  **and**  
 $s\text{-in-}b$ :  $s \in b$   
**by blast**  
**have reach-c**:  
 $\Gamma \vdash ([\text{the } (\Gamma \ p)], [], \text{Normal } \sigma) \rightarrow^* (c \# \text{While } b \ c \# \text{cs, css, Normal } s)$   
**proof** –  
**note reach**  
**also from s-in-b**  
**have**  $\Gamma \vdash (\text{While } b \ c \# \text{cs, css, Normal } s) \rightarrow$   
 $(c \# \text{While } b \ c \# \text{cs, css, Normal } s)$   
**by (rule step.WhileTrue)**  
**finally**  
**show ?thesis .**  
**qed**  
**from reach termi have**  
 $\text{termi-while}$ :  $\Gamma \vdash \text{While } b \ c \downarrow \text{Normal } s$   
**by (rule steps-preserves-termination)**  
**show**  $s \in \{t. t = s \wedge \Gamma \vdash \langle c, \text{Normal } t \rangle \Rightarrow \notin (\{\text{Stuck}\} \cup \text{Fault } '(-F)) \wedge$   
 $\Gamma \vdash \text{the } (\Gamma \ p) \downarrow \text{Normal } \sigma \wedge$   
 $(\exists \text{cs css. } \Gamma \vdash ([\text{the } (\Gamma \ p)], [], \text{Normal } \sigma) \rightarrow^* (c \# \text{cs, css, Normal } t))\}$   $\wedge$   
 $(\forall t. t \in \{t. \Gamma \vdash \langle c, \text{Normal } s \rangle \Rightarrow \text{Normal } t\} \longrightarrow$   
 $t \in \{t. (t, \tau) \in ?r\} \cap$   
 $\{t. (Z, t) \in ?\text{unroll} \wedge$   
 $(\forall e. (Z, e) \in ?\text{unroll} \longrightarrow e \in b$   
 $\longrightarrow \Gamma \vdash \langle c, \text{Normal } e \rangle \Rightarrow \notin (\{\text{Stuck}\} \cup \text{Fault } '(-F)) \wedge$   
 $(\forall u. \Gamma \vdash \langle c, \text{Normal } e \rangle \Rightarrow \text{Abrupt } u \longrightarrow$   
 $\Gamma \vdash \langle \text{While } b \ c, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } u)) \wedge$   
 $\Gamma \vdash \text{the } (\Gamma \ p) \downarrow \text{Normal } \sigma \wedge$   
 $(\exists \text{cs css. } \Gamma \vdash ([\text{the } (\Gamma \ p)], [], \text{Normal } \sigma) \rightarrow^*$   
 $(\text{While } b \ c \# \text{cs, css, Normal } t))\}$   $\wedge$   
 $(\forall t. t \in \{t. \Gamma \vdash \langle c, \text{Normal } s \rangle \Rightarrow \text{Abrupt } t\} \longrightarrow$   
 $t \in \{t. \Gamma \vdash \langle \text{While } b \ c, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\})$   
**(is ?C1  $\wedge$  ?C2  $\wedge$  ?C3)**  
**proof (intro conjI)**  
**from Z-s-unroll noabort s-in-b termi reach-c show ?C1**  
**by blast**  
**next**  
**{**  
**fix t**  
**assume s-t**:  $\Gamma \vdash \langle c, \text{Normal } s \rangle \Rightarrow \text{Normal } t$   
**with s-eq- $\tau$  termi-while s-in-b have (t,  $\tau$ )  $\in$  ?r**

by *blast*  
**moreover**  
**from** *Z-s-unroll s-t s-in-b*  
**have**  $(Z, t) \in ?unroll$   
 by (*blast intro: rtrancl-into-rtrancl*)  
**moreover**  
**have**  $\Gamma \vdash ([the (\Gamma p)], [], Normal \sigma) \rightarrow^* (While b c \# cs, css, Normal t)$   
**proof** –  
**note** *reach-c*  
**also from** *s-t*  
**have**  $\Gamma \vdash (c \# While b c \# cs, css, Normal s) \rightarrow^*$   
 $(While b c \# cs, css, Normal t)$   
 by (*rule exec-impl-steps*)  
**finally show** *?thesis* .  
**qed**  
**moreover note** *noabort termi*  
**ultimately**  
**have**  $(t, \tau) \in ?r \wedge (Z, t) \in ?unroll \wedge$   
 $(\forall e. (Z, e) \in ?unroll \rightarrow e \in b$   
 $\rightarrow \Gamma \vdash \langle c, Normal e \rangle \Rightarrow \notin (\{Stuck\} \cup Fault ' (-F)) \wedge$   
 $(\forall u. \Gamma \vdash \langle c, Normal e \rangle \Rightarrow Abrupt u \rightarrow$   
 $\Gamma \vdash \langle While b c, Normal Z \rangle \Rightarrow Abrupt u)) \wedge$   
 $\Gamma \vdash the (\Gamma p) \downarrow Normal \sigma \wedge$   
 $(\exists cs css. \Gamma \vdash ([the (\Gamma p)], [], Normal \sigma) \rightarrow^*$   
 $(While b c \# cs, css, Normal t))$   
 by *iprover*  
**}**  
**then show** *?C2* by *blast*  
**next**  
**{**  
**fix** *t*  
**assume** *s-t*:  $\Gamma \vdash \langle c, Normal s \rangle \Rightarrow Abrupt t$   
**from** *Z-s-unroll noabort s-t s-in-b*  
**have**  $\Gamma \vdash \langle While b c, Normal Z \rangle \Rightarrow Abrupt t$   
 by *blast*  
**}** **thus** *?C3* by *simp*  
**qed**  
**qed**  
**qed**  
**next**  
**fix** *s*  
**assume** *P*:  $s \in \{s. s = Z \wedge \Gamma \vdash \langle While b c, Normal s \rangle \Rightarrow \notin (\{Stuck\} \cup Fault ' (-F))$   
 $\wedge$   
 $\Gamma \vdash the (\Gamma p) \downarrow Normal \sigma \wedge$   
 $(\exists cs css. \Gamma \vdash ([the (\Gamma p)], [], Normal \sigma) \rightarrow^*$   
 $(While b c \# cs, css, Normal s))\}$   
**hence** *WhileNoFault*:  $\Gamma \vdash \langle While b c, Normal Z \rangle \Rightarrow \notin (\{Stuck\} \cup Fault ' (-F))$   
 by *auto*  
**show**  $s \in ?P' s \wedge$

$(\forall t. t \in (?P' s \cap - b) \longrightarrow$   
 $t \in \{t. \Gamma \vdash \langle \text{While } b \ c, \text{Normal } Z \rangle \Rightarrow \text{Normal } t\}) \wedge$   
 $(\forall t. t \in ?A \ s \longrightarrow t \in ?A \ Z)$   
**proof** (*intro conjI*)  
 $\{$   
**fix**  $e$   
**assume**  $(Z, e) \in ?\text{unroll } e \in b$   
**from** *this WhileNoFault*  
**have**  $\Gamma \vdash \langle c, \text{Normal } e \rangle \Rightarrow \notin (\{\text{Stuck}\} \cup \text{Fault } '(-F)) \wedge$   
 $(\forall u. \Gamma \vdash \langle c, \text{Normal } e \rangle \Rightarrow \text{Abrupt } u \longrightarrow$   
 $\Gamma \vdash \langle \text{While } b \ c, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } u)$  (**is**  $?Prop \ Z \ e$ )  
**proof** (*induct rule: converse-rtrancl-induct [consumes 1]*)  
**assume**  $e\text{-in-}b: e \in b$   
**assume** *WhileNoFault*:  $\Gamma \vdash \langle \text{While } b \ c, \text{Normal } e \rangle \Rightarrow \notin (\{\text{Stuck}\} \cup \text{Fault } '(-F))$   
 $(-F)$   
**with**  $e\text{-in-}b$  *WhileNoFault*  
**have**  $c\text{NoFault}: \Gamma \vdash \langle c, \text{Normal } e \rangle \Rightarrow \notin (\{\text{Stuck}\} \cup \text{Fault } '(-F))$   
**by** (*auto simp add: final-notin-def intro: exec.intros*)  
**moreover**  
 $\{$   
**fix**  $u$  **assume**  $\Gamma \vdash \langle c, \text{Normal } e \rangle \Rightarrow \text{Abrupt } u$   
**with**  $e\text{-in-}b$  **have**  $\Gamma \vdash \langle \text{While } b \ c, \text{Normal } e \rangle \Rightarrow \text{Abrupt } u$   
**by** (*blast intro: exec.intros*)  
 $\}$   
**ultimately**  
**show**  $?Prop \ e \ e$   
**by** *iprover*  
**next**  
**fix**  $Z \ r$   
**assume**  $e\text{-in-}b: e \in b$   
**assume** *WhileNoFault*:  $\Gamma \vdash \langle \text{While } b \ c, \text{Normal } Z \rangle \Rightarrow \notin (\{\text{Stuck}\} \cup \text{Fault } '(-F))$   
 $(-F)$   
**assume** *hyp*:  $\llbracket e \in b; \Gamma \vdash \langle \text{While } b \ c, \text{Normal } r \rangle \Rightarrow \notin (\{\text{Stuck}\} \cup \text{Fault } '(-F)) \rrbracket$   
 $\implies ?Prop \ r \ e$   
**assume**  $Z\text{-}r$ :  
 $(Z, r) \in \{(Z, r). Z \in b \wedge \Gamma \vdash \langle c, \text{Normal } Z \rangle \Rightarrow \text{Normal } r\}$   
**with** *WhileNoFault*  
**have**  $\Gamma \vdash \langle \text{While } b \ c, \text{Normal } r \rangle \Rightarrow \notin (\{\text{Stuck}\} \cup \text{Fault } '(-F))$   
**by** (*auto simp add: final-notin-def intro: exec.intros*)  
**from** *hyp* [*OF e-in-b this*] **obtain**  
 $c\text{NoFault}: \Gamma \vdash \langle c, \text{Normal } e \rangle \Rightarrow \notin (\{\text{Stuck}\} \cup \text{Fault } '(-F))$  **and**  
 $\text{Abrupt-}r: \forall u. \Gamma \vdash \langle c, \text{Normal } e \rangle \Rightarrow \text{Abrupt } u \longrightarrow$   
 $\Gamma \vdash \langle \text{While } b \ c, \text{Normal } r \rangle \Rightarrow \text{Abrupt } u$   
**by** *simp*  
 $\{$   
**fix**  $u$  **assume**  $\Gamma \vdash \langle c, \text{Normal } e \rangle \Rightarrow \text{Abrupt } u$   
**with**  $\text{Abrupt-}r$  **have**  $\Gamma \vdash \langle \text{While } b \ c, \text{Normal } r \rangle \Rightarrow \text{Abrupt } u$  **by** *simp*  
**moreover from**  $Z\text{-}r$  **obtain**

```

      Z ∈ b Γ⊢⟨c, Normal Z⟩ ⇒ Normal r
      by simp
      ultimately have Γ⊢⟨While b c, Normal Z⟩ ⇒ Abrupt u
      by (blast intro: exec.intros)
    }
    with cNoFault show ?Prop Z e
    by iprover
  qed
}
with P show s ∈ ?P' s
by blast
next
{
  fix t
  assume termination: t ∉ b
  assume (Z, t) ∈ ?unroll
  hence Γ⊢⟨While b c, Normal Z⟩ ⇒ Normal t
  proof (induct rule: converse-rtrancl-induct [consumes 1])
    from termination
    show Γ⊢⟨While b c, Normal t⟩ ⇒ Normal t
    by (blast intro: exec.WhileFalse)
  next
  fix Z r
  assume first-body:
    (Z, r) ∈ {(s, t). s ∈ b ∧ Γ⊢⟨c, Normal s⟩ ⇒ Normal t}
  assume (r, t) ∈ ?unroll
  assume rest-loop: Γ⊢⟨While b c, Normal r⟩ ⇒ Normal t
  show Γ⊢⟨While b c, Normal Z⟩ ⇒ Normal t
  proof -
    from first-body obtain
      Z ∈ b Γ⊢⟨c, Normal Z⟩ ⇒ Normal r
    by fast
    moreover
    from rest-loop have
      Γ⊢⟨While b c, Normal r⟩ ⇒ Normal t
    by fast
    ultimately show Γ⊢⟨While b c, Normal Z⟩ ⇒ Normal t
    by (rule exec.WhileTrue)
  qed
  qed
}
with P
show ∀ t. t ∈ (?P' s ∩ - b)
  → t ∈ {t. Γ⊢⟨While b c, Normal Z⟩ ⇒ Normal t}
by blast
next
from P show ∀ t. t ∈ ?A s → t ∈ ?A Z
by simp
qed

```

**qed**  
**next**  
**case** (*Call q*)  
**let**  $?P = \{s. s=Z \wedge \Gamma \vdash \langle \text{Call } q, \text{Normal } s \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F)) \wedge$   
 $\Gamma \vdash \text{the } (\Gamma \text{ } p) \downarrow \text{Normal } \sigma \wedge$   
 $(\exists cs \text{ } css. \Gamma \vdash ([\text{the } (\Gamma \text{ } p)], [], \text{Normal } \sigma) \rightarrow^*$   
 $(\text{Call } q \# cs, css, \text{Normal } s))\}$   
**from** *noStuck-Call*  
**have**  $\forall s \in ?P. q \in \text{dom } \Gamma$   
**by** (*fastforce simp add: final-notin-def*)  
**then show**  $?case$   
**proof** (*rule conseq-extract-state-indep-prop*)  
**assume** *q-defined*:  $q \in \text{dom } \Gamma$   
**from** *Call-hyp* **have**  
 $\forall q \in \text{dom } \Gamma. \forall Z.$   
 $\Gamma, \Theta \vdash_{t/F} \{s. s=Z \wedge \Gamma \vdash \langle \text{the } (\Gamma \text{ } q), \text{Normal } s \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F)) \wedge$   
 $\Gamma \vdash (\text{the } (\Gamma \text{ } q)) \downarrow \text{Normal } s \wedge ((s, q), (\sigma, p)) \in \text{termi-call-steps } \Gamma\}$   
 $(\text{Call } q)$   
 $\{t. \Gamma \vdash \langle \text{the } (\Gamma \text{ } q), \text{Normal } Z \rangle \Rightarrow \text{Normal } t\},$   
 $\{t. \Gamma \vdash \langle \text{the } (\Gamma \text{ } q), \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\}$   
**by** (*simp add: exec-Call-body' noFaultStuck-Call-body' [simplified]*)  
*terminates-Normal-Call-body*)  
**from** *Call-hyp* *q-defined* **have** *Call-hyp'*:  
 $\forall Z. \Gamma, \Theta \vdash_{t/F} \{s. s=Z \wedge \Gamma \vdash \langle \text{Call } q, \text{Normal } s \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F)) \wedge$   
 $\Gamma \vdash \text{Call } q \downarrow \text{Normal } s \wedge ((s, q), (\sigma, p)) \in \text{termi-call-steps } \Gamma\}$   
 $(\text{Call } q)$   
 $\{t. \Gamma \vdash \langle \text{Call } q, \text{Normal } Z \rangle \Rightarrow \text{Normal } t\},$   
 $\{t. \Gamma \vdash \langle \text{Call } q, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\}$   
**by** *auto*  
**show**  
 $\Gamma, \Theta \vdash_{t/F} ?P$   
 $(\text{Call } q)$   
 $\{t. \Gamma \vdash \langle \text{Call } q, \text{Normal } Z \rangle \Rightarrow \text{Normal } t\},$   
 $\{t. \Gamma \vdash \langle \text{Call } q, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\}$   
**proof** (*rule ConseqMGT [OF Call-hyp', safe]*)  
**fix** *cs css*  
**assume**  
 $\Gamma \vdash ([\text{the } (\Gamma \text{ } p)], [], \text{Normal } \sigma) \rightarrow^* (\text{Call } q \# cs, css, \text{Normal } Z)$   
 $\Gamma \vdash \text{the } (\Gamma \text{ } p) \downarrow \text{Normal } \sigma$   
**hence**  $\Gamma \vdash \text{Call } q \downarrow \text{Normal } Z$   
**by** (*rule steps-preserves-termination*)  
**with** *q-defined* **show**  $\Gamma \vdash \text{Call } q \downarrow \text{Normal } Z$   
**by** (*auto elim: terminates-Normal-elim-cases*)  
**next**  
**fix** *cs css*  
**assume** *reach*:  
 $\Gamma \vdash ([\text{the } (\Gamma \text{ } p)], [], \text{Normal } \sigma) \rightarrow^* (\text{Call } q \# cs, css, \text{Normal } Z)$   
**moreover** **have**  $\Gamma \vdash (\text{Call } q \# cs, css, \text{Normal } Z) \rightarrow$   
 $([\text{the } (\Gamma \text{ } q)], (cs, \text{Throw} \# cs) \# css, \text{Normal } Z)$

by (rule step.Call) (insert q-defined,auto)  
 ultimately  
 have  $\Gamma \vdash ([the (\Gamma p)], [], Normal \sigma) \rightarrow^+ ([the (\Gamma q)], (cs, Throw \# cs) \# css, Normal Z)$   
 by (rule rtranclp-into-tranclp1)  
 moreover  
 assume termi:  $\Gamma \vdash the (\Gamma p) \downarrow Normal \sigma$   
 ultimately  
 show  $((Z, q), \sigma, p) \in termi\text{-call-steps } \Gamma$   
 by (auto simp add: termi-call-steps-def)  
 qed  
 qed  
 next  
 case (DynCom c)  
 have hyp:  
 $\bigwedge s'. \forall Z. \Gamma, \Theta \vdash_{t/F} \{s. s = Z \wedge \Gamma \vdash \langle c s', Normal s \rangle \Rightarrow \notin (\{Stuck\} \cup Fault \text{ ' } (-F)) \wedge$   
 $\Gamma \vdash the (\Gamma p) \downarrow Normal \sigma \wedge$   
 $(\exists cs css. \Gamma \vdash ([the (\Gamma p)], [], Normal \sigma) \rightarrow^* (c s' \# cs, css, Normal s))\}$   
 $(c s')$   
 $\{t. \Gamma \vdash \langle c s', Normal Z \rangle \Rightarrow Normal t\}, \{t. \Gamma \vdash \langle c s', Normal Z \rangle \Rightarrow Abrupt t\}$   
 using DynCom by simp  
 have hyp':  
 $\Gamma, \Theta \vdash_{t/F} \{s. s = Z \wedge \Gamma \vdash \langle DynCom c, Normal s \rangle \Rightarrow \notin (\{Stuck\} \cup Fault \text{ ' } (-F)) \wedge$   
 $\Gamma \vdash the (\Gamma p) \downarrow Normal \sigma \wedge$   
 $(\exists cs css. \Gamma \vdash ([the (\Gamma p)], [], Normal \sigma) \rightarrow^* (DynCom c \# cs, css, Normal s))\}$   
 $(c Z)$   
 $\{t. \Gamma \vdash \langle DynCom c, Normal Z \rangle \Rightarrow Normal t\}, \{t. \Gamma \vdash \langle DynCom c, Normal Z \rangle \Rightarrow$   
 Abrupt t}  
 proof (rule ConseqMGT [OF hyp], safe)  
 assume  $\Gamma \vdash \langle DynCom c, Normal Z \rangle \Rightarrow \notin (\{Stuck\} \cup Fault \text{ ' } (-F))$   
 then show  $\Gamma \vdash \langle c Z, Normal Z \rangle \Rightarrow \notin (\{Stuck\} \cup Fault \text{ ' } (-F))$   
 by (fastforce simp add: final-notin-def intro: exec.intros)  
 next  
 fix cs css  
 assume  $\Gamma \vdash ([the (\Gamma p)], [], Normal \sigma) \rightarrow^* (DynCom c \# cs, css, Normal Z)$   
 also have  $\Gamma \vdash (DynCom c \# cs, css, Normal Z) \rightarrow (c Z \# cs, css, Normal Z)$   
 by (rule step.DynCom)  
 finally  
 show  $\exists cs css. \Gamma \vdash ([the (\Gamma p)], [], Normal \sigma) \rightarrow^* (c Z \# cs, css, Normal Z)$   
 by blast  
 next  
 fix t  
 assume  $\Gamma \vdash \langle c Z, Normal Z \rangle \Rightarrow Normal t$   
 thus  $\Gamma \vdash \langle DynCom c, Normal Z \rangle \Rightarrow Normal t$   
 by (auto intro: exec.intros)  
 next  
 fix t

```

assume  $\Gamma \vdash \langle c \ Z, Normal \ Z \rangle \Rightarrow Abrupt \ t$ 
thus  $\Gamma \vdash \langle DynCom \ c, Normal \ Z \rangle \Rightarrow Abrupt \ t$ 
  by (auto intro: exec.intros)
qed
show ?case
  apply (rule hoaret.DynCom)
  apply safe
  apply (rule hyp')
  done
next
case (Guard f g c)
have hyp-c:  $\forall Z. \Gamma, \Theta \vdash_{t/F}$ 
   $\{s. s=Z \wedge \Gamma \vdash \langle c, Normal \ s \rangle \Rightarrow \notin(\{Stuck\} \cup Fault \ '(-F)) \wedge$ 
     $\Gamma \vdash the \ (\Gamma \ p) \downarrow Normal \ \sigma \wedge$ 
     $(\exists cs \ css. \Gamma \vdash ([the \ (\Gamma \ p)], [], Normal \ \sigma) \rightarrow^* (c\#cs, css, Normal \ s))\}$ 
   $c$ 
   $\{t. \Gamma \vdash \langle c, Normal \ Z \rangle \Rightarrow Normal \ t\},$ 
   $\{t. \Gamma \vdash \langle c, Normal \ Z \rangle \Rightarrow Abrupt \ t\}$ 
  using Guard.hyps by iprover
show  $\Gamma, \Theta \vdash_{t/F} \{s. s=Z \wedge \Gamma \vdash \langle Guard \ f \ g \ c \ , Normal \ s \rangle \Rightarrow \notin(\{Stuck\} \cup Fault \ '(-F)) \wedge$ 
   $(-F)) \wedge$ 
   $\Gamma \vdash the \ (\Gamma \ p) \downarrow Normal \ \sigma \wedge$ 
   $(\exists cs \ css. \Gamma \vdash ([the \ (\Gamma \ p)], [], Normal \ \sigma) \rightarrow^* (Guard \ f \ g \ c\#cs, css, Normal$ 
   $s))\}$ 
   $Guard \ f \ g \ c$ 
   $\{t. \Gamma \vdash \langle Guard \ f \ g \ c \ , Normal \ Z \rangle \Rightarrow Normal \ t\},$ 
   $\{t. \Gamma \vdash \langle Guard \ f \ g \ c \ , Normal \ Z \rangle \Rightarrow Abrupt \ t\}$ 
proof (cases f ∈ F)
  case True
  have  $\Gamma, \Theta \vdash_{t/F} (g \cap \{s. s=Z \wedge$ 
   $\Gamma \vdash \langle Guard \ f \ g \ c \ , Normal \ s \rangle \Rightarrow \notin(\{Stuck\} \cup Fault \ '(-F)) \wedge$ 
   $\Gamma \vdash the \ (\Gamma \ p) \downarrow Normal \ \sigma \wedge$ 
   $(\exists cs \ css. \Gamma \vdash ([the \ (\Gamma \ p)], [], Normal \ \sigma) \rightarrow^* (Guard \ f \ g \ c\#cs, css, Normal$ 
   $s))\}$ 
   $c$ 
   $\{t. \Gamma \vdash \langle Guard \ f \ g \ c \ , Normal \ Z \rangle \Rightarrow Normal \ t\},$ 
   $\{t. \Gamma \vdash \langle Guard \ f \ g \ c \ , Normal \ Z \rangle \Rightarrow Abrupt \ t\}$ 
proof (rule ConseqMGT [OF hyp-c], safe)
  assume  $\Gamma \vdash \langle Guard \ f \ g \ c \ , Normal \ Z \rangle \Rightarrow \notin(\{Stuck\} \cup Fault \ '(-F)) \ Z \in g$ 
  thus  $\Gamma \vdash \langle c, Normal \ Z \rangle \Rightarrow \notin(\{Stuck\} \cup Fault \ '(-F))$ 
  by (auto simp add: final-notin-def intro: exec.intros)
next
  fix cs css
  assume  $\Gamma \vdash ([the \ (\Gamma \ p)], [], Normal \ \sigma) \rightarrow^* (Guard \ f \ g \ c\#cs, css, Normal \ Z)$ 
  also
  assume  $Z \in g$ 
  hence  $\Gamma \vdash (Guard \ f \ g \ c\#cs, css, Normal \ Z) \rightarrow (c\#cs, css, Normal \ Z)$ 
  by (rule step.Guard)
  finally show  $\exists cs \ css. \Gamma \vdash ([the \ (\Gamma \ p)], [], Normal \ \sigma) \rightarrow^* (c\#cs, css, Normal \ Z)$ 

```

```

    by iprover
next
fix t
assume  $\Gamma \vdash \langle \text{Guard } f \ g \ c \ , \text{Normal } Z \rangle \Rightarrow \notin (\{\text{Stuck}\} \cup \text{Fault } '(-F))$ 
       $\Gamma \vdash \langle c, \text{Normal } Z \rangle \Rightarrow \text{Normal } t \ Z \in g$ 
thus  $\Gamma \vdash \langle \text{Guard } f \ g \ c \ , \text{Normal } Z \rangle \Rightarrow \text{Normal } t$ 
  by (auto simp add: final-notin-def intro: exec.intros )
next
fix t
assume  $\Gamma \vdash \langle \text{Guard } f \ g \ c \ , \text{Normal } Z \rangle \Rightarrow \notin (\{\text{Stuck}\} \cup \text{Fault } '(-F))$ 
       $\Gamma \vdash \langle c, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t \ Z \in g$ 
thus  $\Gamma \vdash \langle \text{Guard } f \ g \ c \ , \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t$ 
  by (auto simp add: final-notin-def intro: exec.intros )
qed
from True this show ?thesis
  by (rule conseqPre [OF Guarantee]) auto
next
case False
have  $\Gamma, \Theta \vdash_{t/F} (g \cap \{s. s=Z \wedge$ 
       $\Gamma \vdash \langle \text{Guard } f \ g \ c \ , \text{Normal } s \rangle \Rightarrow \notin (\{\text{Stuck}\} \cup \text{Fault } '(-F)) \wedge$ 
       $\Gamma \vdash \text{the } (\Gamma \ p) \downarrow \text{Normal } \sigma \wedge$ 
       $(\exists cs \ css. \Gamma \vdash ([\text{the } (\Gamma \ p)], [], \text{Normal } \sigma) \rightarrow^* (\text{Guard } f \ g \ c \# cs, \text{css}, \text{Normal } s)))$ 
)
  c
  {t.  $\Gamma \vdash \langle \text{Guard } f \ g \ c \ , \text{Normal } Z \rangle \Rightarrow \text{Normal } t$ },
  {t.  $\Gamma \vdash \langle \text{Guard } f \ g \ c \ , \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t$ }
proof (rule ConseqMGT [OF hyp-c], safe)
  assume  $\Gamma \vdash \langle \text{Guard } f \ g \ c \ , \text{Normal } Z \rangle \Rightarrow \notin (\{\text{Stuck}\} \cup \text{Fault } '(-F))$ 
  thus  $\Gamma \vdash \langle c, \text{Normal } Z \rangle \Rightarrow \notin (\{\text{Stuck}\} \cup \text{Fault } '(-F))$ 
    using False
  by (cases  $Z \in g$ ) (auto simp add: final-notin-def intro: exec.intros)
next
fix cs css
assume  $\Gamma \vdash ([\text{the } (\Gamma \ p)], [], \text{Normal } \sigma) \rightarrow^* (\text{Guard } f \ g \ c \# cs, \text{css}, \text{Normal } Z)$ 
also assume  $\Gamma \vdash \langle \text{Guard } f \ g \ c \ , \text{Normal } Z \rangle \Rightarrow \notin (\{\text{Stuck}\} \cup \text{Fault } '(-F))$ 
hence  $Z \in g$ 
  using False by (auto simp add: final-notin-def intro: exec.GuardFault)
hence  $\Gamma \vdash (\text{Guard } f \ g \ c \# cs, \text{css}, \text{Normal } Z) \rightarrow (c \# cs, \text{css}, \text{Normal } Z)$ 
  by (rule step.Guard)
finally show  $\exists cs \ css. \Gamma \vdash ([\text{the } (\Gamma \ p)], [], \text{Normal } \sigma) \rightarrow^* (c \# cs, \text{css}, \text{Normal } Z)$ 
  by iprover
next
fix t
assume  $\Gamma \vdash \langle \text{Guard } f \ g \ c \ , \text{Normal } Z \rangle \Rightarrow \notin (\{\text{Stuck}\} \cup \text{Fault } '(-F))$ 
       $\Gamma \vdash \langle c, \text{Normal } Z \rangle \Rightarrow \text{Normal } t$ 
thus  $\Gamma \vdash \langle \text{Guard } f \ g \ c \ , \text{Normal } Z \rangle \Rightarrow \text{Normal } t$ 
  using False
  by (cases  $Z \in g$ ) (auto simp add: final-notin-def intro: exec.intros )
next

```



```

fix  $t$ 
assume  $\Gamma \vdash \langle \text{Guard } f \ g \ c \ , \text{Normal } Z \rangle \Rightarrow \notin (\{ \text{Stuck} \} \cup \text{Fault } '(-F))$ 
       $\Gamma \vdash \langle c, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t$ 
thus  $\Gamma \vdash \langle \text{Guard } f \ g \ c \ , \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t$ 
      using  $\text{False}$ 
      by ( $\text{cases } Z \in g$ ) ( $\text{auto simp add: final-notin-def intro: exec.intros}$ )
qed
then show  $?thesis$ 
      apply ( $\text{rule conseqPre [OF hoaret.Guard]}$ )
      apply  $\text{clarify}$ 
      apply ( $\text{frule Guard-noFaultStuckD [OF - False]}$ )
      apply  $\text{auto}$ 
      done
qed
next
case  $\text{Throw}$ 
show  $\Gamma, \Theta \vdash_{t/F} \{ s. s = Z \wedge \Gamma \vdash \langle \text{Throw}, \text{Normal } s \rangle \Rightarrow \notin (\{ \text{Stuck} \} \cup \text{Fault } '(-F)) \wedge$ 
       $\Gamma \vdash \text{the } (\Gamma \ p) \downarrow \text{Normal } \sigma \wedge$ 
       $(\exists cs \ \text{css}. \Gamma \vdash ([\text{the } (\Gamma \ p)], [], \text{Normal } \sigma) \rightarrow^* (\text{Throw} \# cs, \text{css}, \text{Normal } s))\}$ 
       $\text{Throw}$ 
       $\{ t. \Gamma \vdash \langle \text{Throw}, \text{Normal } Z \rangle \Rightarrow \text{Normal } t \},$ 
       $\{ t. \Gamma \vdash \langle \text{Throw}, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t \}$ 
      by ( $\text{rule conseqPre [OF hoaret.Throw]}$ )
      ( $\text{blast intro: exec.intros terminates.intros}$ )
next
case ( $\text{Catch } c_1 \ c_2$ )
have  $\text{hyp-c1}$ :
   $\forall Z. \Gamma, \Theta \vdash_{t/F} \{ s. s = Z \wedge \Gamma \vdash \langle c_1, \text{Normal } s \rangle \Rightarrow \notin (\{ \text{Stuck} \} \cup \text{Fault } '(-F)) \wedge$ 
     $\Gamma \vdash \text{the } (\Gamma \ p) \downarrow \text{Normal } \sigma \wedge$ 
     $(\exists cs \ \text{css}. \Gamma \vdash ([\text{the } (\Gamma \ p)], [], \text{Normal } \sigma) \rightarrow^* (c_1 \# cs, \text{css}, \text{Normal } s))\}$ 
     $c_1$ 
     $\{ t. \Gamma \vdash \langle c_1, \text{Normal } Z \rangle \Rightarrow \text{Normal } t \}, \{ t. \Gamma \vdash \langle c_1, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t \}$ 
    using  $\text{Catch.hyps}$  by  $\text{iprover}$ 
have  $\text{hyp-c2}$ :
   $\forall Z. \Gamma, \Theta \vdash_{t/F} \{ s. s = Z \wedge \Gamma \vdash \langle c_2, \text{Normal } s \rangle \Rightarrow \notin (\{ \text{Stuck} \} \cup \text{Fault } '(-F)) \wedge$ 
     $\Gamma \vdash \text{the } (\Gamma \ p) \downarrow \text{Normal } \sigma \wedge$ 
     $(\exists cs \ \text{css}. \Gamma \vdash ([\text{the } (\Gamma \ p)], [], \text{Normal } \sigma) \rightarrow^* (c_2 \# cs, \text{css}, \text{Normal } s))\}$ 
     $c_2$ 
     $\{ t. \Gamma \vdash \langle c_2, \text{Normal } Z \rangle \Rightarrow \text{Normal } t \}, \{ t. \Gamma \vdash \langle c_2, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t \}$ 
    using  $\text{Catch.hyps}$  by  $\text{iprover}$ 
have
   $\Gamma, \Theta \vdash_{t/F} \{ s. s = Z \wedge \Gamma \vdash \langle \text{Catch } c_1 \ c_2, \text{Normal } s \rangle \Rightarrow \notin (\{ \text{Stuck} \} \cup \text{Fault } '(-F))$ 
   $\wedge$ 
     $\Gamma \vdash \text{the } (\Gamma \ p) \downarrow \text{Normal } \sigma \wedge$ 
     $(\exists cs \ \text{css}. \Gamma \vdash ([\text{the } (\Gamma \ p)], [], \text{Normal } \sigma) \rightarrow^* (\text{Catch } c_1 \ c_2 \ # cs, \text{css}, \text{Normal } s))\}$ 
     $c_1$ 
     $\{ t. \Gamma \vdash \langle \text{Catch } c_1 \ c_2, \text{Normal } Z \rangle \Rightarrow \text{Normal } t \},$ 
     $\{ t. \Gamma \vdash \langle c_1, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t \wedge$ 
     $\Gamma \vdash \langle c_2, \text{Normal } t \rangle \Rightarrow \notin (\{ \text{Stuck} \} \cup \text{Fault } '(-F)) \wedge \Gamma \vdash \text{the } (\Gamma \ p) \downarrow \text{Normal}$ 

```

$\sigma \wedge$   
 $(\exists cs\ css. \Gamma \vdash ([the\ (\Gamma\ p)], [], Normal\ \sigma) \rightarrow^* (c_2 \# cs, css, Normal\ t))\}$   
**proof** (rule *ConseqMGT* [*OF hyp-c1*], *clarify, safe*)  
**assume**  $\Gamma \vdash \langle Catch\ c_1\ c_2, Normal\ Z \rangle \Rightarrow \notin(\{Stuck\} \cup Fault\ '(-F))$   
**thus**  $\Gamma \vdash \langle c_1, Normal\ Z \rangle \Rightarrow \notin(\{Stuck\} \cup Fault\ '(-F))$   
**by** (*fastforce simp add: final-notin-def intro: exec.intros*)  
**next**  
**fix**  $cs\ css$   
**assume**  $\Gamma \vdash ([the\ (\Gamma\ p)], [], Normal\ \sigma) \rightarrow^* (Catch\ c_1\ c_2\ \# cs, css, Normal\ Z)$   
**also have**  
 $\Gamma \vdash (Catch\ c_1\ c_2\ \# cs, css, Normal\ Z) \rightarrow ([c_1], (cs, c_2 \# cs) \# css, Normal\ Z)$   
**by** (*rule step.Catch*)  
**finally**  
**show**  $\exists cs\ css. \Gamma \vdash ([the\ (\Gamma\ p)], [], Normal\ \sigma) \rightarrow^* (c_1 \# cs, css, Normal\ Z)$   
**by** *iprover*  
**next**  
**fix**  $t$   
**assume**  $\Gamma \vdash \langle c_1, Normal\ Z \rangle \Rightarrow Normal\ t$   
**thus**  $\Gamma \vdash \langle Catch\ c_1\ c_2, Normal\ Z \rangle \Rightarrow Normal\ t$   
**by** (*auto intro: exec.intros*)  
**next**  
**fix**  $t$   
**assume**  $\Gamma \vdash \langle Catch\ c_1\ c_2, Normal\ Z \rangle \Rightarrow \notin(\{Stuck\} \cup Fault\ '(-F))$   
 $\Gamma \vdash \langle c_1, Normal\ Z \rangle \Rightarrow Abrupt\ t$   
**thus**  $\Gamma \vdash \langle c_2, Normal\ t \rangle \Rightarrow \notin(\{Stuck\} \cup Fault\ '(-F))$   
**by** (*auto simp add: final-notin-def intro: exec.intros*)  
**next**  
**fix**  $cs\ css\ t$   
**assume**  $\Gamma \vdash ([the\ (\Gamma\ p)], [], Normal\ \sigma) \rightarrow^* (Catch\ c_1\ c_2\ \# cs, css, Normal\ Z)$   
**also have**  
 $\Gamma \vdash (Catch\ c_1\ c_2\ \# cs, css, Normal\ Z) \rightarrow ([c_1], (cs, c_2 \# cs) \# css, Normal\ Z)$   
**by** (*rule step.Catch*)  
**also**  
**assume**  $\Gamma \vdash \langle c_1, Normal\ Z \rangle \Rightarrow Abrupt\ t$   
**hence**  $\Gamma \vdash ([c_1], (cs, c_2 \# cs) \# css, Normal\ Z) \rightarrow^* ([], (cs, c_2 \# cs) \# css, Abrupt\ t)$   
**by** (*rule exec-impl-steps*)  
**also**  
**have**  $\Gamma \vdash ([], (cs, c_2 \# cs) \# css, Abrupt\ t) \rightarrow (c_2 \# cs, css, Normal\ t)$   
**by** (*rule step.intros*)  
**finally**  
**show**  $\exists cs\ css. \Gamma \vdash ([the\ (\Gamma\ p)], [], Normal\ \sigma) \rightarrow^* (c_2 \# cs, css, Normal\ t)$   
**by** *iprover*  
**qed**  
**moreover**  
**have**  $\Gamma, \Theta \vdash_{t/F} \{t. \Gamma \vdash \langle c_1, Normal\ Z \rangle \Rightarrow Abrupt\ t \wedge$   
 $\Gamma \vdash \langle c_2, Normal\ t \rangle \Rightarrow \notin(\{Stuck\} \cup Fault\ '(-F)) \wedge$   
 $\Gamma \vdash the\ (\Gamma\ p) \downarrow Normal\ \sigma \wedge$   
 $(\exists cs\ css. \Gamma \vdash ([the\ (\Gamma\ p)], [], Normal\ \sigma) \rightarrow^* (c_2 \# cs, css, Normal\ t))\}$   
 $c_2$

$\{t. \Gamma \vdash \langle \text{Catch } c_1 \ c_2, \text{Normal } Z \rangle \Rightarrow \text{Normal } t\},$   
 $\{t. \Gamma \vdash \langle \text{Catch } c_1 \ c_2, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\}$   
**by** (rule *ConseqMGT* [*OF hyp-c2*]) (*fastforce intro: exec.intros*)  
**ultimately show** *?case*  
**by** (rule *hoaret.Catch*)  
**qed**

To prove a procedure implementation correct it suffices to assume only the procedure specifications of procedures that actually occur during evaluation of the body.

**lemma** *Call-lemma*:

**assumes**

*Call*:  $\forall q \in \text{dom } \Gamma. \forall Z. \Gamma, \Theta \vdash_t / F$

$\{s. s=Z \wedge \Gamma \vdash \langle \text{Call } q, \text{Normal } s \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F)) \wedge$   
 $\Gamma \vdash \text{Call } q \downarrow \text{Normal } s \wedge ((s, q), (\sigma, p)) \in \text{termi-call-steps } \Gamma\}$   
*(Call q)*

$\{t. \Gamma \vdash \langle \text{Call } q, \text{Normal } Z \rangle \Rightarrow \text{Normal } t\},$   
 $\{t. \Gamma \vdash \langle \text{Call } q, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\}$

**shows**  $\bigwedge Z. \Gamma, \Theta \vdash_t / F$

$(\{\sigma\} \cap \{s. s=Z \wedge \Gamma \vdash \langle \text{the } (\Gamma \ p), \text{Normal } s \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F))$

$\wedge$

$\Gamma \vdash \text{the } (\Gamma \ p) \downarrow \text{Normal } s\})$

*the* ( $\Gamma \ p$ )

$\{t. \Gamma \vdash \langle \text{the } (\Gamma \ p), \text{Normal } Z \rangle \Rightarrow \text{Normal } t\},$   
 $\{t. \Gamma \vdash \langle \text{the } (\Gamma \ p), \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\}$

**apply** (rule *conseqPre*)

**apply** (rule *Call-lemma'*)

**apply** (rule *Call*)

**apply** *blast*

**done**

**lemma** *Call-lemma-switch-Call-body*:

**assumes**

*call*:  $\forall q \in \text{dom } \Gamma. \forall Z. \Gamma, \Theta \vdash_t / F$

$\{s. s=Z \wedge \Gamma \vdash \langle \text{Call } q, \text{Normal } s \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F)) \wedge$   
 $\Gamma \vdash \text{Call } q \downarrow \text{Normal } s \wedge ((s, q), (\sigma, p)) \in \text{termi-call-steps } \Gamma\}$   
*(Call q)*

$\{t. \Gamma \vdash \langle \text{Call } q, \text{Normal } Z \rangle \Rightarrow \text{Normal } t\},$   
 $\{t. \Gamma \vdash \langle \text{Call } q, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\}$

**assumes** *p-defined*:  $p \in \text{dom } \Gamma$

**shows**  $\bigwedge Z. \Gamma, \Theta \vdash_t / F$

$(\{\sigma\} \cap \{s. s=Z \wedge \Gamma \vdash \langle \text{Call } p, \text{Normal } s \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F)) \wedge$   
 $\Gamma \vdash \text{Call } p \downarrow \text{Normal } s\})$

*the* ( $\Gamma \ p$ )

$\{t. \Gamma \vdash \langle \text{Call } p, \text{Normal } Z \rangle \Rightarrow \text{Normal } t\},$   
 $\{t. \Gamma \vdash \langle \text{Call } p, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\}$

**apply** (*simp only: exec-Call-body' [OF p-defined] noFaultStuck-Call-body' [OF p-defined]*)  
*terminates-Normal-Call-body [OF p-defined]*)

**apply** (*rule conseqPre*)  
**apply** (*rule Call-lemma'*)  
**apply** (*rule call*)  
**apply** *blast*  
**done**

**lemma** *MGT-Call*:

$\forall p \in \text{dom } \Gamma. \forall Z.$

$\Gamma, \Theta \vdash_{t/F} \{s. s=Z \wedge \Gamma \vdash \langle \text{Call } p, \text{Normal } s \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F)) \wedge$   
 $\Gamma \vdash (\text{Call } p) \downarrow \text{Normal } s\}$   
 $(\text{Call } p)$   
 $\{t. \Gamma \vdash \langle \text{Call } p, \text{Normal } Z \rangle \Rightarrow \text{Normal } t\},$   
 $\{t. \Gamma \vdash \langle \text{Call } p, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\}$

**apply** (*intro ballI allI*)

**apply** (*rule CallRec'* [**where**  $\text{Procs} = \text{dom } \Gamma$  **and**

$P = \lambda p Z. \{s. s=Z \wedge \Gamma \vdash \langle \text{Call } p, \text{Normal } s \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F)) \wedge$   
 $\Gamma \vdash \text{Call } p \downarrow \text{Normal } s\}$  **and**

$Q = \lambda p Z. \{t. \Gamma \vdash \langle \text{Call } p, \text{Normal } Z \rangle \Rightarrow \text{Normal } t\}$  **and**

$A = \lambda p Z. \{t. \Gamma \vdash \langle \text{Call } p, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\}$  **and**

$r = \text{termi-call-steps } \Gamma$

)

**apply** *simp*

**apply** *simp*

**apply** (*rule wf-termi-call-steps*)

**apply** (*intro ballI allI*)

**apply** *simp*

**apply** (*rule Call-lemma-switch-Call-body* [*rule-format, simplified*])

**apply** (*rule hoaret.Asm*)

**apply** *fastforce*

**apply** *assumption*

**done**

**lemma** *CollInt-iff*:  $\{s. P s\} \cap \{s. Q s\} = \{s. P s \wedge Q s\}$

**by** *auto*

**lemma** *image-Un-conv*:  $f ' (\bigcup_{p \in \text{dom } \Gamma} \bigcup Z. \{x p Z\}) = (\bigcup_{p \in \text{dom } \Gamma} \bigcup Z. \{f (x p Z)\})$

**by** (*auto iff: not-None-eq*)

Another proof of *MGT-Call*, maybe a little more readable

**lemma**

$\forall p \in \text{dom } \Gamma. \forall Z.$

$\Gamma, \{\} \vdash_{t/F} \{s. s=Z \wedge \Gamma \vdash \langle \text{Call } p, \text{Normal } s \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F)) \wedge$   
 $\Gamma \vdash (\text{Call } p) \downarrow \text{Normal } s\}$   
 $(\text{Call } p)$   
 $\{t. \Gamma \vdash \langle \text{Call } p, \text{Normal } Z \rangle \Rightarrow \text{Normal } t\},$   
 $\{t. \Gamma \vdash \langle \text{Call } p, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\}$

```

proof –
{
  fix  $p Z \sigma$ 
  assume defined:  $p \in \text{dom } \Gamma$ 
  define Specs where  $\text{Specs} = (\bigcup_{p \in \text{dom } \Gamma} \bigcup Z.$ 
     $\{\{s. s=Z \wedge$ 
       $\Gamma \vdash \langle \text{Call } p, \text{Normal } s \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F)) \wedge$ 
       $\Gamma \vdash \text{Call } p \downarrow \text{Normal } s\},$ 
       $p,$ 
       $\{t. \Gamma \vdash \langle \text{Call } p, \text{Normal } Z \rangle \Rightarrow \text{Normal } t\},$ 
       $\{t. \Gamma \vdash \langle \text{Call } p, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\}\})$ 
  define Specs-wf where  $\text{Specs-wf } p \sigma = (\lambda(P, q, Q, A).$ 
     $(P \cap \{s. ((s, q), \sigma, p) \in \text{termi-call-steps } \Gamma\}, q, Q, A)) ' \text{Specs for}$ 
 $p \sigma$ 
  have  $\Gamma, \text{Specs-wf } p \sigma$ 
     $\vdash_{t/F}(\{\sigma\} \cap$ 
       $\{s. s = Z \wedge \Gamma \vdash \langle \text{the } (\Gamma p), \text{Normal } s \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F)) \wedge$ 
       $\Gamma \vdash \text{the } (\Gamma p) \downarrow \text{Normal } s\})$ 
       $(\text{the } (\Gamma p))$ 
       $\{t. \Gamma \vdash \langle \text{the } (\Gamma p), \text{Normal } Z \rangle \Rightarrow \text{Normal } t\},$ 
       $\{t. \Gamma \vdash \langle \text{the } (\Gamma p), \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\}$ 
    apply (rule Call-lemma [rule-format])
    apply (rule hoaret.Asm)
    apply (clarsimp simp add: Specs-wf-def Specs-def image-Un-conv)
    apply (rule-tac x=q in bezI)
    apply (rule-tac x=Z in exI)
    apply (clarsimp simp add: CollInt-iff)
    apply auto
    done
  hence  $\Gamma, \text{Specs-wf } p \sigma$ 
     $\vdash_{t/F}(\{\sigma\} \cap$ 
       $\{s. s = Z \wedge \Gamma \vdash \langle \text{Call } p, \text{Normal } s \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F)) \wedge$ 
       $\Gamma \vdash \text{Call } p \downarrow \text{Normal } s\})$ 
       $(\text{the } (\Gamma p))$ 
       $\{t. \Gamma \vdash \langle \text{Call } p, \text{Normal } Z \rangle \Rightarrow \text{Normal } t\},$ 
       $\{t. \Gamma \vdash \langle \text{Call } p, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\}$ 
    by (simp only: exec-Call-body' [OF defined])
      noFaultStuck-Call-body' [OF defined]
      terminates-Normal-Call-body [OF defined])
} note bdy=this
show ?thesis
apply (intro ballI allI)
apply (rule hoaret.CallRec [where  $\text{Specs} = (\bigcup_{p \in \text{dom } \Gamma} \bigcup Z.$ 
   $\{\{s. s=Z \wedge$ 
     $\Gamma \vdash \langle \text{Call } p, \text{Normal } s \rangle \Rightarrow \notin(\{\text{Stuck}\} \cup \text{Fault } '(-F)) \wedge$ 
     $\Gamma \vdash \text{Call } p \downarrow \text{Normal } s\},$ 
     $p,$ 
     $\{t. \Gamma \vdash \langle \text{Call } p, \text{Normal } Z \rangle \Rightarrow \text{Normal } t\},$ 
     $\{t. \Gamma \vdash \langle \text{Call } p, \text{Normal } Z \rangle \Rightarrow \text{Abrupt } t\}\})$ 

```

```

      OF - wf-termi-call-steps [of  $\Gamma$ ] refl])
    apply fastforce
    apply clarify
    apply (rule conjI)
    apply fastforce
    apply (rule allI)
    apply (simp (no-asm-use) only : Un-empty-left)
    apply (rule bdy)
    apply auto
  done
qed

end

```

```

theory Simpl-Heap
imports Main
begin

```

## 14.5 References

```

definition ref = (UNIV::nat set)

```

```

typedef ref = ref by (simp add: ref-def)

```

```

code-datatype Abs-ref

```

```

lemma finite-nat-ex-max:
  assumes fin: finite (N::nat set)
  shows  $\exists m. \forall n \in N. n < m$ 
using fin
proof (induct)
  case empty
  show ?case by auto
next
  case (insert k N)
  have  $\exists m. \forall n \in N. n < m$  by fact
  then obtain m where m-max:  $\forall n \in N. n < m..$ 
  show  $\exists m. \forall n \in \text{insert } k \ N. n < m$ 
  proof (rule exI [where x=Suc (max k m)])
    qed (insert m-max, auto simp add: max-def)
  qed

```

```

lemma infinite-nat:  $\neg \text{finite (UNIV::nat set)}$ 

```

```

proof
  assume fin: finite (UNIV::nat set)
  then obtain m::nat where  $\forall n \in \text{UNIV}. n < m$ 
    by (rule finite-nat-ex-max [elim-format]) auto
  moreover have  $m \in \text{UNIV}..$ 

```

ultimately show *False* by *blast*  
 qed

**lemma** *infinite-ref* [*simp,intro*]:  $\neg \text{finite } (\text{UNIV}::\text{ref set})$

**proof**

assume *finite* (*UNIV::ref set*)

hence *finite* (*range Rep-ref*)

by *simp*

moreover

have *range Rep-ref = ref*

**proof**

show *range Rep-ref*  $\subseteq$  *ref*

by (*simp add: ref-def*)

next

show *ref*  $\subseteq$  *range Rep-ref*

**proof**

fix *x*

assume *x*: *x*  $\in$  *ref*

show *x*  $\in$  *range Rep-ref*

by (*rule Rep-ref-induct*) (*auto simp add: ref-def*)

qed

qed

ultimately have *finite ref*

by *simp*

thus *False*

by (*simp add: ref-def infinite-nat*)

qed

**consts** *Null* :: *ref*

**definition** *new* :: *ref set*  $\Rightarrow$  *ref* **where**

*new A = (SOME a. a  $\notin$  {Null}  $\cup$  A)*

Constant *Null* can be defined later on. Conceptually *Null* and *new* are *fixes* of a locale with *finite A*  $\Longrightarrow$  *new A*  $\notin$  *A*  $\cup$  {*Null*}. But since definitions relative to a locale do not yet work in Isabelle2005 we use this workaround to avoid lots of parameters in definitions.

**lemma** *new-notin* [*simp,intro*]:

*finite A*  $\Longrightarrow$  *new (A)*  $\notin$  *A*

**apply** (*unfold new-def*)

**apply** (*rule someI2-ex*)

**apply** (*fastforce intro: ex-new-if-finite*)

**apply** *simp*

**done**

**lemma** *new-not-Null* [*simp,intro*]:

*finite A*  $\Longrightarrow$  *new (A)*  $\neq$  *Null*

**apply** (*unfold new-def*)

**apply** (*rule someI2-ex*)

```

  apply (fastforce intro: ex-new-if-finite)
  apply simp
done

end

```

## 15 Paths and Lists in the Heap

```

theory HeapList
imports Simpl-Heap
begin

```

Adapted from 'HOL/Hoare/Heap.thy'.

### 15.1 Paths in The Heap

```

primrec
  Path :: ref  $\Rightarrow$  (ref  $\Rightarrow$  ref)  $\Rightarrow$  ref  $\Rightarrow$  ref list  $\Rightarrow$  bool
where
  Path x h y [] = (x = y) |
  Path x h y (p#ps) = (x = p  $\wedge$  x  $\neq$  Null  $\wedge$  Path (h x) h y ps)

```

```

lemma Path-Null-iff [iff]: Path Null h y xs = (xs = []  $\wedge$  y = Null)
apply(case-tac xs)
apply fastforce
apply fastforce
done

```

```

lemma Path-not-Null-iff [simp]: p $\neq$ Null  $\implies$ 
  Path p h q as = (as = []  $\wedge$  q = p  $\vee$  ( $\exists$  ps. as = p#ps  $\wedge$  Path (h p) h q ps ))
apply(case-tac as)
apply fastforce
apply fastforce
done

```

```

lemma Path-append [simp]:
   $\bigwedge$ p. Path p f q (as@bs) = ( $\exists$  y. Path p f y as  $\wedge$  Path y f q bs)
by(induct as, simp+)

```

```

lemma notin-Path-update[simp]:
   $\bigwedge$ p. u  $\notin$  set ps  $\implies$  Path p (f(u := v)) q ps = Path p f q ps
by(induct ps, simp, simp add:eq-sym-conv)

```

```

lemma Path-upd-same [simp]:
  Path p (f(p:=p)) q qs =
    ((p=Null  $\wedge$  q=Null  $\wedge$  qs = [])  $\vee$  (p $\neq$ Null  $\wedge$  q=p  $\wedge$  ( $\forall$  x $\in$ set qs. x=p)))
by (induct qs) auto

```

*Path-upd-same* prevents  $p \neq \text{Null} \implies \text{Path } p (f(p := p)) q qs = X$  from



looping, because of *Path-not-Null-iff* and *fun-upd-apply*.

**lemma** *notin-Path-updateI* [*intro*]:  
 $\llbracket \text{Path } p \ h \ q \ ps ; r \notin \text{set } ps \rrbracket \implies \text{Path } p \ (h(r := y)) \ q \ ps$   
**by** *simp*

**lemma** *Path-update-new* [*simp*]:  $\llbracket \text{set } ps \subseteq \text{set } \text{alloc} \rrbracket$   
 $\implies \text{Path } p \ (f(\text{new } (\text{set } \text{alloc}) := x)) \ q \ ps = \text{Path } p \ f \ q \ ps$   
**by** (*rule notin-Path-update*) *fastforce*

**lemma** *Null-notin-Path* [*simp,intro*]:  
 $\bigwedge p. \text{Path } p \ f \ q \ ps \implies \text{Null} \notin \text{set } ps$   
**by**(*induct ps*) *auto*

**lemma** *Path-snoc*:  
 $\llbracket \text{Path } p \ (f(a := q)) \ a \ as ; a \neq \text{Null} \rrbracket \implies \text{Path } p \ (f(a := q)) \ q \ (as @ [a])$   
**by** *simp*

## 15.2 Lists on The Heap

### 15.2.1 Relational Abstraction

**definition**  
 $\text{List} :: \text{ref} \Rightarrow (\text{ref} \Rightarrow \text{ref}) \Rightarrow \text{ref list} \Rightarrow \text{bool}$  **where**  
 $\text{List } p \ h \ ps = \text{Path } p \ h \ \text{Null } ps$

**lemma** *List-empty* [*simp*]:  $\text{List } p \ h \ [] = (p = \text{Null})$   
**by**(*simp add:List-def*)

**lemma** *List-cons* [*simp*]:  $\text{List } p \ h \ (a\#ps) = (p = a \wedge p \neq \text{Null} \wedge \text{List } (h \ p) \ h \ ps)$   
**by**(*simp add:List-def*)

**lemma** *List-Null* [*simp*]:  $\text{List } \text{Null} \ h \ ps = (ps = [])$   
**by**(*case-tac ps, simp-all*)

**lemma** *List-not-Null* [*simp*]:  $p \neq \text{Null} \implies$   
 $\text{List } p \ h \ as = (\exists ps. as = p\#ps \wedge \text{List } (h \ p) \ h \ ps)$   
**by**(*case-tac as, simp-all, fast*)

**lemma** *Null-notin-List* [*simp,intro*]:  $\bigwedge p. \text{List } p \ h \ ps \implies \text{Null} \notin \text{set } ps$   
**by** (*simp add : List-def*)

**theorem** *notin-List-update*[*simp*]:  
 $\bigwedge p. q \notin \text{set } ps \implies \text{List } p \ (h(q := y)) \ ps = \text{List } p \ h \ ps$   
**apply**(*induct ps*)  
**apply** *simp*  
**apply** *clarsimp*  
**done**

**lemma** *List-upd-same-lemma*:  $\bigwedge p. p \neq \text{Null} \implies \neg \text{List } p (h(p := p)) ps$   
**apply** (*induct ps*)  
**apply** *simp*  
**apply** (*simp (no-asm-simp) del: fun-upd-apply*)  
**apply** (*simp (no-asm-simp) only: fun-upd-apply refl if-True*)  
**apply** *blast*  
**done**

**lemma** *List-upd-same* [*simp*]:  $\text{List } p (h(p:=p)) ps = (p = \text{Null} \wedge ps = [])$   
**apply** (*cases p=Null*)  
**apply** *simp*  
**apply** (*fast dest: List-upd-same-lemma*)  
**done**

*List-upd-same* prevents  $p \neq \text{Null} \implies \text{List } p (h(p := p)) as = X$  from looping, because of *List-not-Null* and *fun-upd-apply*.

**lemma** *List-update-new* [*simp*]:  $\llbracket \text{set } ps \subseteq \text{set } \text{alloc} \rrbracket$   
 $\implies \text{List } p (h(\text{new } (\text{set } \text{alloc}) := x)) ps = \text{List } p h ps$   
**by** (*rule notin-List-update*) *fastforce*

**lemma** *List-updateI* [*intro*]:  
 $\llbracket \text{List } p h ps; q \notin \text{set } ps \rrbracket \implies \text{List } p (h(q := y)) ps$   
**by** *simp*

**lemma** *List-unique*:  $\bigwedge p bs. \text{List } p h as \implies \text{List } p h bs \implies as = bs$   
**by**(*induct as, simp, clarsimp*)

**lemma** *List-unique1*:  $\text{List } p h as \implies \exists! as. \text{List } p h as$   
**by**(*blast intro:List-unique*)

**lemma** *List-app*:  $\bigwedge p. \text{List } p h (as@bs) = (\exists y. \text{Path } p h y as \wedge \text{List } y h bs)$   
**by**(*induct as, simp, clarsimp*)

**lemma** *List-hd-not-in-tl*[*simp*]:  $\text{List } (h p) h ps \implies p \notin \text{set } ps$   
**apply** (*clarsimp simp add:in-set-conv-decomp*)  
**apply**(*frule List-app[THEN iffD1]*)  
**apply**(*fastforce dest: List-unique*)  
**done**

**lemma** *List-distinct*[*simp*]:  $\bigwedge p. \text{List } p h ps \implies \text{distinct } ps$   
**apply**(*induct ps, simp*)  
**apply**(*fastforce dest:List-hd-not-in-tl*)  
**done**

**lemma** *heap-eq-List-eq*:  
 $\bigwedge p. \forall x \in \text{set } ps. h x = g x \implies \text{List } p h ps = \text{List } p g ps$   
**by** (*induct ps*) *auto*

**lemma** *heap-eq-ListI*:  
**assumes** *list*: *List p h ps*  
**assumes** *hp-eq*:  $\forall x \in \text{set } ps. h x = g x$   
**shows** *List p g ps*  
**using** *list*  
**by** (*simp add: heap-eq-List-eq [OF hp-eq]*)

**lemma** *heap-eq-ListII*:  
**assumes** *list*: *List p h ps*  
**assumes** *hp-eq*:  $\forall x \in \text{set } ps. g x = h x$   
**shows** *List p g ps*  
**using** *list*  
**by** (*simp add: heap-eq-List-eq [OF hp-eq]*)

The following lemmata are useful for the simplifier to instantiate bound variables in the assumptions resp. conclusion, using the uniqueness of the List predicate

**lemma** *conj-impl-simp*:  $(P \wedge Q \longrightarrow K) = (P \longrightarrow Q \longrightarrow K)$   
**by** *auto*

**lemma** *List-unique-all-impl-simp [simp]*:  
*List p h ps  $\implies (\forall ps. List p h ps \longrightarrow P ps) = P ps$*   
**by** (*auto dest: List-unique*)

**lemma** *List-unique-ex-conj-simp [simp]*:  
*List p h ps  $\implies (\exists ps. List p h ps \wedge P ps) = P ps$*   
**by** (*auto dest: List-unique*)

### 15.3 Functional abstraction

**definition**  
*islist* :: *ref  $\Rightarrow (ref \Rightarrow ref) \Rightarrow bool$*  **where**  
*islist p h = ( $\exists ps. List p h ps$ )*

**definition**  
*list* :: *ref  $\Rightarrow (ref \Rightarrow ref) \Rightarrow ref list$*  **where**  
*list p h = (THE ps. List p h ps)*

**lemma** *List-conv-islist-list*: *List p h ps = (islist p h  $\wedge ps = list p h$ )*  
**apply**(*simp add: islist-def list-def*)  
**apply**(*rule iffI*)  
**apply**(*rule conjI*)  
**apply** *blast*  
**apply**(*subst the1-equality*)  
**apply**(*erule List-unique1*)  
**apply** *assumption*

```

apply(rule refl)
apply simp
apply (clarify)
apply (rule theI)
apply assumption
by (rule List-unique)

```

```

lemma List-islist [intro]:
  List p h ps  $\implies$  islist p h
  apply (simp add: List-conv-islist-list)
  done

```

```

lemma List-list:
  List p h ps  $\implies$  list p h = ps
  apply (simp only: List-conv-islist-list)
  done

```

```

lemma [simp]: islist Null h
by(simp add:islist-def)

```

```

lemma [simp]: p  $\neq$  Null  $\implies$  islist (h p) h = islist p h
by(simp add:islist-def)

```

```

lemma [simp]: list Null h = []
by(simp add:list-def)

```

```

lemma list-Ref-conv[simp]:
   $\llbracket$ islist (h p) h; p  $\neq$  Null  $\rrbracket \implies$  list p h = p # list (h p) h
  apply(insert List-not-Null[of - h])
  apply(fastforce simp:List-conv-islist-list)
  done

```

```

lemma [simp]: islist (h p) h  $\implies$  p  $\notin$  set(list (h p) h)
  apply(insert List-hd-not-in-tl[of h])
  apply(simp add:List-conv-islist-list)
  done

```

```

lemma list-upd-conv[simp]:
  islist p h  $\implies$  y  $\notin$  set(list p h)  $\implies$  list p (h(y := q)) = list p h
  apply(drule notin-List-update[of - - p h q])
  apply(simp add:List-conv-islist-list)
  done

```

```

lemma islist-upd[simp]:
  islist p h  $\implies$  y  $\notin$  set(list p h)  $\implies$  islist p (h(y := q))
  apply(frule notin-List-update[of - - p h q])
  apply(simp add:List-conv-islist-list)

```

**done**

```
lemma list-distinct[simp]: islist p h  $\implies$  distinct (list p h)  
apply (clarsimp simp add: list-def islist-def)  
apply (frule List-unique1)  
apply (drule (1) the1-equality)  
apply simp  
done
```

```
lemma Null-notin-list [simp,intro]: islist p h  $\implies$  Null  $\notin$  set (list p h)  
apply (clarsimp simp add: list-def islist-def)  
apply (frule List-unique1)  
apply (drule (1) the1-equality)  
apply simp  
done
```

**end**

```
theory Generalise imports HOL-Statespace.DistinctTreeProver  
begin
```

```
lemma protectRefl: PROP Pure.prop (PROP C)  $\implies$  PROP Pure.prop (PROP C)  
  by (simp add: prop-def)
```

```
lemma protectImp:  
  assumes i: PROP Pure.prop (PROP P  $\implies$  PROP Q)  
  shows PROP Pure.prop (PROP Pure.prop P  $\implies$  PROP Pure.prop Q)  
proof –  
  {  
    assume P: PROP Pure.prop P  
    from i [unfolded prop-def, OF P [unfolded prop-def]]  
    have PROP Pure.prop Q  
    by (simp add: prop-def)  
  }  
  note i' = this  
  show PROP ?thesis  
    apply (rule protectI)  
    apply (rule i')  
    apply assumption  
  done  
qed
```

```
lemma generaliseConj:  
  assumes i1: PROP Pure.prop (PROP Pure.prop (Trueprop P)  $\implies$  PROP Pure.prop  
  (Trueprop Q))  
  assumes i2: PROP Pure.prop (PROP Pure.prop (Trueprop P')  $\implies$  PROP  
  Pure.prop (Trueprop Q'))
```

**shows**  $PROP\ Pure.prop (PROP\ Pure.prop (Trueprop (P \wedge P'))) \implies (PROP\ Pure.prop (Trueprop (Q \wedge Q')))$   
**using**  $i1\ i2$   
**by** (*auto simp add: prop-def*)

**lemma** *generaliseAll*:

**assumes**  $i: PROP\ Pure.prop (\bigwedge s. PROP\ Pure.prop (Trueprop (P\ s)) \implies PROP\ Pure.prop (Trueprop (Q\ s)))$   
**shows**  $PROP\ Pure.prop (PROP\ Pure.prop (Trueprop (\forall s. P\ s)) \implies PROP\ Pure.prop (Trueprop (\forall s. Q\ s)))$   
**using**  $i$   
**by** (*auto simp add: prop-def*)

**lemma** *generalise-all*:

**assumes**  $i: PROP\ Pure.prop (\bigwedge s. PROP\ Pure.prop (PROP\ P\ s) \implies PROP\ Pure.prop (PROP\ Q\ s))$   
**shows**  $PROP\ Pure.prop ((PROP\ Pure.prop (\bigwedge s. PROP\ P\ s)) \implies (PROP\ Pure.prop (\bigwedge s. PROP\ Q\ s)))$   
**using**  $i$   
**proof** (*unfold prop-def*)  
**assume**  $i1: \bigwedge s. (PROP\ P\ s) \implies (PROP\ Q\ s)$   
**assume**  $i2: \bigwedge s. PROP\ P\ s$   
**show**  $\bigwedge s. PROP\ Q\ s$   
**by** (*rule i1*) (*rule i2*)  
**qed**

**lemma** *generaliseTrans*:

**assumes**  $i1: PROP\ Pure.prop (PROP\ P \implies PROP\ Q)$   
**assumes**  $i2: PROP\ Pure.prop (PROP\ Q \implies PROP\ R)$   
**shows**  $PROP\ Pure.prop (PROP\ P \implies PROP\ R)$   
**using**  $i1\ i2$   
**proof** (*unfold prop-def*)  
**assume**  $P-Q: PROP\ P \implies PROP\ Q$   
**assume**  $Q-R: PROP\ Q \implies PROP\ R$   
**assume**  $P: PROP\ P$   
**show**  $PROP\ R$   
**by** (*rule Q-R [OF P-Q [OF P]]*)  
**qed**

**lemma** *meta-spec*:

**assumes**  $\bigwedge x. PROP\ P\ x$   
**shows**  $PROP\ P\ x$  **by** *fact*

**lemma** *meta-spec-protect*:

**assumes**  $g: \bigwedge x. PROP\ P\ x$   
**shows**  $PROP\ Pure.prop (PROP\ P\ x)$   
**using**  $g$   
**by** (*auto simp add: prop-def*)

**lemma** *generaliseImp*:  
**assumes**  $i$ :  $PROP\ Pure.prop (PROP\ Pure.prop (Trueprop\ P) \implies PROP\ Pure.prop (Trueprop\ Q))$   
**shows**  $PROP\ Pure.prop (PROP\ Pure.prop (Trueprop (X \longrightarrow P)) \implies PROP\ Pure.prop (Trueprop (X \longrightarrow Q)))$   
**using**  $i$   
**by** (*auto simp add: prop-def*)

**lemma** *generaliseEx*:  
**assumes**  $i$ :  $PROP\ Pure.prop (\bigwedge s. PROP\ Pure.prop (Trueprop (P\ s)) \implies PROP\ Pure.prop (Trueprop (Q\ s)))$   
**shows**  $PROP\ Pure.prop (PROP\ Pure.prop (Trueprop (\exists s. P\ s)) \implies PROP\ Pure.prop (Trueprop (\exists s. Q\ s)))$   
**using**  $i$   
**by** (*auto simp add: prop-def*)

**lemma** *generaliseReft*:  $PROP\ Pure.prop (PROP\ Pure.prop (Trueprop\ P) \implies PROP\ Pure.prop (Trueprop\ P))$   
**by** (*auto simp add: prop-def*)

**lemma** *generaliseReft'*:  $PROP\ Pure.prop (PROP\ P \implies PROP\ P)$   
**by** (*auto simp add: prop-def*)

**lemma** *generaliseAllShift*:  
**assumes**  $i$ :  $PROP\ Pure.prop (\bigwedge s. P \implies Q\ s)$   
**shows**  $PROP\ Pure.prop (PROP\ Pure.prop (Trueprop\ P) \implies PROP\ Pure.prop (Trueprop (\forall s. Q\ s)))$   
**using**  $i$   
**by** (*auto simp add: prop-def*)

**lemma** *generalise-allShift*:  
**assumes**  $i$ :  $PROP\ Pure.prop (\bigwedge s. PROP\ P \implies PROP\ Q\ s)$   
**shows**  $PROP\ Pure.prop (PROP\ Pure.prop (PROP\ P) \implies PROP\ Pure.prop (\bigwedge s. PROP\ Q\ s))$   
**using**  $i$   
**proof** (*unfold prop-def*)  
**assume**  $P\text{-}Q$ :  $\bigwedge s. PROP\ P \implies PROP\ Q\ s$   
**assume**  $P$ :  $PROP\ P$   
**show**  $\bigwedge s. PROP\ Q\ s$   
**by** (*rule P-Q [OF P]*)  
**qed**

**lemma** *generaliseImpl*:  
**assumes**  $i$ :  $PROP\ Pure.prop (PROP\ Pure.prop\ P \implies PROP\ Pure.prop\ Q)$   
**shows**  $PROP\ Pure.prop ((PROP\ Pure.prop (PROP\ X \implies PROP\ P)) \implies (PROP\ Pure.prop (PROP\ X \implies PROP\ Q)))$   
**using**  $i$

```

proof (unfold prop-def)
  assume i1: PROP P  $\implies$  PROP Q
  assume i2: PROP X  $\implies$  PROP P
  assume X: PROP X
  show PROP Q
  by (rule i1 [OF i2 [OF X]])
qed

```

**ML-file**  $\langle$ generalise-state.ML $\rangle$

**end**

## 16 Facilitating the Hoare Logic

```

theory Vcg
imports StateSpace HOL-Statespace.StateSpaceLocale Generalise
keywords procedures hoarestate :: thy-defn
begin

```

```

axiomatization NoBody::('s,'p,'f) com

```

**ML-file**  $\langle$ hoare.ML $\rangle$

```

method-setup hoare = Hoare.hoare
  raw verification condition generator for Hoare Logic

```

```

method-setup hoare-raw = Hoare.hoare-raw
  even more raw verification condition generator for Hoare Logic

```

```

method-setup vcg = Hoare.vcg
  verification condition generator for Hoare Logic

```

```

method-setup vcg-step = Hoare.vcg-step
  single verification condition generation step with light simplification

```

```

method-setup hoare-rule = Hoare.hoare-rule
  apply single hoare rule and solve certain sideconditions

```

Variables of the programming language are represented as components of a record. To avoid cluttering up the namespace of Isabelle with lots of typical variable names, we append a unusual suffix at the end of each name by parsing

```

definition list-multsel:: 'a list  $\Rightarrow$  nat list  $\Rightarrow$  'a list (infixl  $\langle$ !! $\rangle$  100)
  where xs !! ns = map (nth xs) ns

```

```

definition list-multupd:: 'a list  $\Rightarrow$  nat list  $\Rightarrow$  'a list  $\Rightarrow$  'a list

```



**where** *list-multupd xs ns ys = foldl* ( $\lambda xs (n,v). xs[n:=v]$ ) *xs (zip ns ys)*

**nonterminal** *lmupbinds and lmupbind*

**syntax**

— multiple list update

*-lmupbind*:: [*'a, 'a*] => *lmupbind* ( $\langle \langle 2- [:=] / - \rangle \rangle$ )  
:: *lmupbind* => *lmupbinds* ( $\langle \rightarrow \rangle$ )  
*-lmupbinds* :: [*lmupbind, lmupbinds*] => *lmupbinds* ( $\langle \langle -, / - \rangle \rangle$ )  
*-LMUpdate* :: [*'a, lmupbinds*] => *'a* ( $\langle \langle - / (-) \rangle \rangle$ ) [*900,0*] *900*)

**syntax-consts**

*-lmupbind -lmupbinds -LMUpdate == list-multupd*

**translations**

*-LMUpdate xs (-lmupbinds b bs) == -LMUpdate (-LMUpdate xs b) bs*  
*xs[is[:=]ys] == CONST list-multupd xs is ys*

## 16.1 Some Fancy Syntax

reverse application

**definition** *rapp*:: *'a*  $\Rightarrow$  (*'a*  $\Rightarrow$  *'b*)  $\Rightarrow$  *'b* (**infixr**  $\langle \langle \rangle \rangle$  60)

**where** *rapp x f = f x*

**nonterminal**

*newinit and*  
*newinits and*  
*locinit and*  
*locinits and*  
*switchcase and*  
*switchcases and*  
*grds and*  
*grd and*  
*bdy and*  
*basics and*  
*basic and*  
*basicblock*

**notation**

*Skip* ( $\langle \langle \text{SKIP} \rangle \rangle$ ) **and**  
*Throw* ( $\langle \langle \text{THROW} \rangle \rangle$ )

**syntax**

*-raise*:: *'c*  $\Rightarrow$  *'c*  $\Rightarrow$  (*'a, 'b, 'f*) *com* ( $\langle \langle \text{RAISE} - ::= / - \rangle \rangle$ ) [*30, 30*] *23*)  
*-seq*::(*'s, 'p, 'f*) *com*  $\Rightarrow$  (*'s, 'p, 'f*) *com*  $\Rightarrow$  (*'s, 'p, 'f*) *com* ( $\langle \langle ::; / - \rangle \rangle$ ) [*20, 21*] *20*)  
*-guarantee* :: *'s set*  $\Rightarrow$  *grd* ( $\langle \langle \sqrt{} \rangle \rangle$ ) [*1000*] *1000*)  
*-guaranteeStrip*:: *'s set*  $\Rightarrow$  *grd* ( $\langle \langle \# \rangle \rangle$ ) [*1000*] *1000*)  
*-grd* :: *'s set*  $\Rightarrow$  *grd* ( $\langle \langle \rightarrow \rangle \rangle$ ) [*1000*] *1000*)

$-last-grd \quad :: \text{grd} \Rightarrow \text{grds} \quad (\langle \cdot \rangle \ 1000)$   
 $-grds \quad :: [\text{grd}, \text{grds}] \Rightarrow \text{grds} \quad (\langle \cdot, / \rightarrow [999, 1000] \ 1000)$   
 $-guards \quad :: \text{grds} \Rightarrow ('s, 'p, 'f) \text{ com} \Rightarrow ('s, 'p, 'f) \text{ com}$   
 $\quad \quad \quad (\langle (- / \mapsto -) \rangle [60, 21] \ 23)$   
 $-quote \quad :: 'b \Rightarrow ('a \Rightarrow 'b)$   
 $-antiquoteCur0 \quad :: ('a \Rightarrow 'b) \Rightarrow 'b \quad (\langle \cdot \rangle [1000] \ 1000)$   
 $-antiquoteCur \quad :: ('a \Rightarrow 'b) \Rightarrow 'b$   
 $-antiquoteOld0 \quad :: ('a \Rightarrow 'b) \Rightarrow 'a \Rightarrow 'b \quad (\langle \cdot \rangle [1000, 1000] \ 1000)$   
 $-antiquoteOld \quad :: ('a \Rightarrow 'b) \Rightarrow 'a \Rightarrow 'b$   
 $-Assert \quad :: 'a \Rightarrow 'a \text{ set} \quad (\langle (\{ \! \! - \} \! \! -) \rangle [0] \ 1000)$   
 $-AssertState \quad :: \text{idt} \Rightarrow 'a \Rightarrow 'a \text{ set} \quad (\langle (\{ \! \! - \! \! - \} \! \! -) \rangle [1000, 0] \ 1000)$   
 $-Assign \quad :: 'b \Rightarrow 'b \Rightarrow ('a, 'p, 'f) \text{ com} \quad (\langle (- ::= / -) \rangle [30, 30] \ 23)$   
 $-Init \quad :: \text{idnt} \Rightarrow 'c \Rightarrow 'b \Rightarrow ('a, 'p, 'f) \text{ com}$   
 $\quad \quad \quad (\langle (\cdot ::= / -) \rangle [30, 1000, 30] \ 23)$   
 $-GuardedAssign \quad :: 'b \Rightarrow 'b \Rightarrow ('a, 'p, 'f) \text{ com} \quad (\langle (- ::=_g / -) \rangle [30, 30] \ 23)$   
 $-newinit \quad :: [\text{idnt}, 'a] \Rightarrow \text{newinit} \quad (\langle (2' \cdot ::= / -) \rangle)$   
 $\quad \quad \quad :: \text{newinit} \Rightarrow \text{newinits} \quad (\langle \cdot \rangle)$   
 $-newinits \quad :: [\text{newinit}, \text{newinits}] \Rightarrow \text{newinits} \quad (\langle \cdot, / \rightarrow \rangle)$   
 $-New \quad :: ['a, 'b, \text{newinits}] \Rightarrow ('a, 'b, 'f) \text{ com}$   
 $\quad \quad \quad (\langle (- ::= / (2 \text{ NEW } - / [-]) \rangle [30, 65, 0] \ 23)$   
 $-GuardedNew \quad :: ['a, 'b, \text{newinits}] \Rightarrow ('a, 'b, 'f) \text{ com}$   
 $\quad \quad \quad (\langle (- ::=_g / (2 \text{ NEW } - / [-]) \rangle [30, 65, 0] \ 23)$   
 $-NNew \quad :: ['a, 'b, \text{newinits}] \Rightarrow ('a, 'b, 'f) \text{ com}$   
 $\quad \quad \quad (\langle (- ::= / (2 \text{ NNEW } - / [-]) \rangle [30, 65, 0] \ 23)$   
 $-GuardedNNew \quad :: ['a, 'b, \text{newinits}] \Rightarrow ('a, 'b, 'f) \text{ com}$   
 $\quad \quad \quad (\langle (- ::=_g / (2 \text{ NNEW } - / [-]) \rangle [30, 65, 0] \ 23)$   
  
 $-Cond \quad :: 'a \text{ bexp} \Rightarrow ('a, 'p, 'f) \text{ com} \Rightarrow ('a, 'p, 'f) \text{ com} \Rightarrow ('a, 'p, 'f) \text{ com}$   
 $\quad \quad \quad (\langle (0\text{IF } (-) / (2\text{THEN } -) / (2\text{ELSE } -) / \text{FI}) \rangle [0, 0, 0] \ 71)$   
 $-Cond-no-else \quad :: 'a \text{ bexp} \Rightarrow ('a, 'p, 'f) \text{ com} \Rightarrow ('a, 'p, 'f) \text{ com}$   
 $\quad \quad \quad (\langle (0\text{IF } (-) / (2\text{THEN } -) / \text{FI}) \rangle [0, 0] \ 71)$   
 $-GuardedCond \quad :: 'a \text{ bexp} \Rightarrow ('a, 'p, 'f) \text{ com} \Rightarrow ('a, 'p, 'f) \text{ com} \Rightarrow ('a, 'p, 'f) \text{ com}$   
 $\quad \quad \quad (\langle (0\text{IF}_g (-) / (2\text{THEN } -) / (2\text{ELSE } -) / \text{FI}) \rangle [0, 0, 0] \ 71)$   
 $-GuardedCond-no-else \quad :: 'a \text{ bexp} \Rightarrow ('a, 'p, 'f) \text{ com} \Rightarrow ('a, 'p, 'f) \text{ com}$   
 $\quad \quad \quad (\langle (0\text{IF}_g (-) / (2\text{THEN } -) / \text{FI}) \rangle [0, 0] \ 71)$   
 $-While-inv-var \quad :: 'a \text{ bexp} \Rightarrow 'a \text{ assn} \Rightarrow ('a \times 'a) \text{ set} \Rightarrow \text{bdy}$   
 $\quad \quad \quad \Rightarrow ('a, 'p, 'f) \text{ com}$   
 $\quad \quad \quad (\langle (0\text{WHILE } (-) / \text{INV } (-) / \text{VAR } (-) / -) \rangle [25, 0, 0, 81] \ 71)$   
 $-WhileFix-inv-var \quad :: 'a \text{ bexp} \Rightarrow \text{pttrn} \Rightarrow ('z \Rightarrow 'a \text{ assn}) \Rightarrow$   
 $\quad \quad \quad ('z \Rightarrow ('a \times 'a) \text{ set}) \Rightarrow \text{bdy}$   
 $\quad \quad \quad \Rightarrow ('a, 'p, 'f) \text{ com}$   
 $\quad \quad \quad (\langle (0\text{WHILE } (-) / \text{FIX } - / \text{INV } (-) / \text{VAR } (-) / -) \rangle [25, 0, 0, 81] \ 71)$   
 $-WhileFix-inv \quad :: 'a \text{ bexp} \Rightarrow \text{pttrn} \Rightarrow ('z \Rightarrow 'a \text{ assn}) \Rightarrow \text{bdy}$   
 $\quad \quad \quad \Rightarrow ('a, 'p, 'f) \text{ com}$   
 $\quad \quad \quad (\langle (0\text{WHILE } (-) / \text{FIX } - / \text{INV } (-) / -) \rangle [25, 0, 0, 81] \ 71)$   
 $-GuardedWhileFix-inv-var \quad :: 'a \text{ bexp} \Rightarrow \text{pttrn} \Rightarrow ('z \Rightarrow 'a \text{ assn}) \Rightarrow$   
 $\quad \quad \quad ('z \Rightarrow ('a \times 'a) \text{ set}) \Rightarrow \text{bdy}$   
 $\quad \quad \quad \Rightarrow ('a, 'p, 'f) \text{ com}$   
 $\quad \quad \quad (\langle (0\text{WHILE}_g (-) / \text{FIX } - / \text{INV } (-) / \text{VAR } (-) / -) \rangle [25, 0, 0, 81] \ 71)$

-GuardedWhileFix-inv-var-hook :: 'a bexp  $\Rightarrow$  ('z  $\Rightarrow$  'a assn)  $\Rightarrow$   
     ('z  $\Rightarrow$  ('a  $\times$  'a) set)  $\Rightarrow$  bdy  
      $\Rightarrow$  ('a,'p,'f) com  
 -GuardedWhileFix-inv :: 'a bexp  $\Rightarrow$  pttrn  $\Rightarrow$  ('z  $\Rightarrow$  'a assn)  $\Rightarrow$  bdy  
      $\Rightarrow$  ('a,'p,'f) com  
     ( $\langle$ (0WHILE<sub>g</sub> (-)/ FIX -./ INV (-)/-)  $\rangle$  [25, 0, 0, 81] 71)

-GuardedWhile-inv-var::  
     'a bexp  $\Rightarrow$  'a assn  $\Rightarrow$  ('a  $\times$  'a) set  $\Rightarrow$  bdy  $\Rightarrow$  ('a,'p,'f) com  
     ( $\langle$ (0WHILE<sub>g</sub> (-)/ INV (-)/ VAR (-) /-)  $\rangle$  [25, 0, 0, 81] 71)

-While-inv :: 'a bexp  $\Rightarrow$  'a assn  $\Rightarrow$  bdy  $\Rightarrow$  ('a,'p,'f) com  
     ( $\langle$ (0WHILE (-)/ INV (-) /-)  $\rangle$  [25, 0, 81] 71)

-GuardedWhile-inv :: 'a bexp  $\Rightarrow$  'a assn  $\Rightarrow$  ('a,'p,'f) com  $\Rightarrow$  ('a,'p,'f) com  
     ( $\langle$ (0WHILE<sub>g</sub> (-)/ INV (-) /-)  $\rangle$  [25, 0, 81] 71)

-While :: 'a bexp  $\Rightarrow$  bdy  $\Rightarrow$  ('a,'p,'f) com  
     ( $\langle$ (0WHILE (-) /-)  $\rangle$  [25, 81] 71)

-GuardedWhile :: 'a bexp  $\Rightarrow$  bdy  $\Rightarrow$  ('a,'p,'f) com  
     ( $\langle$ (0WHILE<sub>g</sub> (-) /-)  $\rangle$  [25, 81] 71)

-While-guard :: grds  $\Rightarrow$  'a bexp  $\Rightarrow$  bdy  $\Rightarrow$  ('a,'p,'f) com  
     ( $\langle$ (0WHILE (-/  $\mapsto$  (1-)) /-)  $\rangle$  [1000,25,81] 71)

-While-guard-inv:: grds  $\Rightarrow$  'a bexp  $\Rightarrow$  'a assn  $\Rightarrow$  bdy  $\Rightarrow$  ('a,'p,'f) com  
     ( $\langle$ (0WHILE (-/  $\mapsto$  (1-)) INV (-) /-)  $\rangle$  [1000,25,0,81] 71)

-While-guard-inv-var:: grds  $\Rightarrow$  'a bexp  $\Rightarrow$  'a assn  $\Rightarrow$  ('a  $\times$  'a) set  
      $\Rightarrow$  bdy  $\Rightarrow$  ('a,'p,'f) com  
     ( $\langle$ (0WHILE (-/  $\mapsto$  (1-)) INV (-)/ VAR (-) /-)  $\rangle$  [1000,25,0,0,81] 71)

-WhileFix-guard-inv-var:: grds  $\Rightarrow$  'a bexp  $\Rightarrow$  pttrn  $\Rightarrow$  ('z  $\Rightarrow$  'a assn)  $\Rightarrow$  ('z  $\Rightarrow$  ('a  $\times$  'a) set)  
      $\Rightarrow$  bdy  $\Rightarrow$  ('a,'p,'f) com  
     ( $\langle$ (0WHILE (-/  $\mapsto$  (1-)) FIX -./ INV (-)/ VAR (-) /-)  $\rangle$  [1000,25,0,0,0,81] 71)

-WhileFix-guard-inv:: grds  $\Rightarrow$  'a bexp  $\Rightarrow$  pttrn  $\Rightarrow$  ('z  $\Rightarrow$  'a assn)  
      $\Rightarrow$  bdy  $\Rightarrow$  ('a,'p,'f) com  
     ( $\langle$ (0WHILE (-/  $\mapsto$  (1-)) FIX -./ INV (-)/-)  $\rangle$  [1000,25,0,0,81] 71)

-Try-Catch:: ('a,'p,'f) com  $\Rightarrow$  ('a,'p,'f) com  $\Rightarrow$  ('a,'p,'f) com  
     ( $\langle$ (0TRY (-)/ (2CATCH -)/ END)  $\rangle$  [0,0] 71)

-DoPre :: ('a,'p,'f) com  $\Rightarrow$  ('a,'p,'f) com  
 -Do :: ('a,'p,'f) com  $\Rightarrow$  bdy ( $\langle$ (2DO/ (-)) /OD  $\rangle$  [0] 1000)

-Lab:: 'a bexp  $\Rightarrow$  ('a,'p,'f) com  $\Rightarrow$  bdy  
     ( $\langle$ -./-  $\rangle$  [1000,71] 81)

:: bdy  $\Rightarrow$  ('a,'p,'f) com ( $\langle$ -  $\rangle$ )

-Spec:: pttrn  $\Rightarrow$  's set  $\Rightarrow$  ('s,'p,'f) com  $\Rightarrow$  's set  $\Rightarrow$  's set  $\Rightarrow$  ('s,'p,'f) com  
     ( $\langle$ (ANNO -. -/ (-) / -,/-)  $\rangle$  [0,1000,20,1000,1000] 60)

-SpecNoAbrupt:: pttrn  $\Rightarrow$  's set  $\Rightarrow$  ('s,'p,'f) com  $\Rightarrow$  's set  $\Rightarrow$  ('s,'p,'f) com  
     ( $\langle$ (ANNO -. -/ (-) /-)  $\rangle$  [0,1000,20,1000] 60)

-LemAnno:: 'n  $\Rightarrow$  ('s,'p,'f) com  $\Rightarrow$  ('s,'p,'f) com  
     ( $\langle$ (0 LEMMA (-) / - END)  $\rangle$  [1000,0] 71)

-locnoinit :: ident  $\Rightarrow$  locinit ( $\langle$ '-  $\rangle$ )  
 -locinit :: [ident,'a]  $\Rightarrow$  locinit ( $\langle$ (2'- ::= / -)  $\rangle$ )

$\text{locinit} \Rightarrow \text{locinits} \quad (\langle \rightarrow \rangle)$   
 $\text{-locinits} \quad :: [\text{locinit}, \text{locinits}] \Rightarrow \text{locinits} \quad (\langle -, / \rightarrow \rangle)$   
 $\text{-Loc}:: [\text{locinits}, ('s, 'p, 'f) \text{ com}] \Rightarrow ('s, 'p, 'f) \text{ com}$   
 $\quad (\langle (2 \text{ LOC } -;; / (-) \text{ COL}) \rangle [0, 0] 71)$   
 $\text{-Switch}:: ('s \Rightarrow 'v) \Rightarrow \text{switchcases} \Rightarrow ('s, 'p, 'f) \text{ com}$   
 $\quad (\langle (0 \text{ SWITCH } (-) / - \text{ END}) \rangle [22, 0] 71)$   
 $\text{-switchcase}:: 'v \text{ set} \Rightarrow ('s, 'p, 'f) \text{ com} \Rightarrow \text{switchcase} \quad (\langle \Rightarrow / \rightarrow \rangle)$   
 $\text{-switchcasesSingle} \quad :: \text{switchcase} \Rightarrow \text{switchcases} \quad (\langle \rightarrow \rangle)$   
 $\text{-switchcasesCons}:: \text{switchcase} \Rightarrow \text{switchcases} \Rightarrow \text{switchcases}$   
 $\quad (\langle - / | \rightarrow \rangle)$   
 $\text{-Basic}:: \text{basicblock} \Rightarrow ('s, 'p, 'f) \text{ com} \quad (\langle (0 \text{ BASIC } / (-) / \text{ END}) \rangle [22] 71)$   
 $\text{-BasicBlock}:: \text{basics} \Rightarrow \text{basicblock} \quad (\langle \rightarrow \rangle)$   
 $\text{-BAssign} \quad :: 'b = > 'b \Rightarrow \text{basic} \quad (\langle (- ::= / -) \rangle [30, 30] 23)$   
 $\quad :: \text{basic} \Rightarrow \text{basics} \quad (\langle \rightarrow \rangle)$   
 $\text{-basics} \quad :: [\text{basic}, \text{basics}] \Rightarrow \text{basics} \quad (\langle -, / \rightarrow \rangle)$

### syntax (ASCII)

$\text{-Assert} \quad :: 'a = > 'a \text{ set} \quad (\langle (\{ | - \}) \rangle [0] 1000)$   
 $\text{-AssertState} \quad :: \text{idt} \Rightarrow 'a \Rightarrow 'a \text{ set} \quad (\langle (\{ | - . - \}) \rangle [1000, 0] 1000)$   
 $\text{-While-guard} \quad :: \text{grds} \Rightarrow 'a \text{ bexp} \Rightarrow \text{bdy} \Rightarrow ('a, 'p, 'f) \text{ com}$   
 $\quad (\langle (0 \text{ WHILE } (- | \rightarrow - / -) / -) \rangle [0, 0, 1000] 71)$   
 $\text{-While-guard-inv}:: \text{grds} \Rightarrow 'a \text{ bexp} \Rightarrow 'a \text{ assn} \Rightarrow \text{bdy} \Rightarrow ('a, 'p, 'f) \text{ com}$   
 $\quad (\langle (0 \text{ WHILE } (- | \rightarrow - / -) \text{ INV } (-) / -) \rangle [0, 0, 0, 1000] 71)$   
 $\text{-guards} \quad :: \text{grds} \Rightarrow ('s, 'p, 'f) \text{ com} \Rightarrow ('s, 'p, 'f) \text{ com} \quad (\langle (- | \rightarrow -) \rangle [60, 21] 23)$

### syntax (output)

$\text{-hidden-grds} \quad :: \text{grds} \quad (\langle \dots \rangle)$

### translations

$\text{-Do } c = > c$   
 $b \bullet c = > \text{CONST condCatch } c \text{ b SKIP}$   
 $b \bullet (-\text{DoPre } c) <= \text{CONST condCatch } c \text{ b SKIP}$   
 $l \bullet (\text{CONST whileAnnoG } gs \text{ b I V } c) <= l \bullet (-\text{DoPre } (\text{CONST whileAnnoG } gs \text{ b I V } c))$   
 $l \bullet (\text{CONST whileAnno } b \text{ I V } c) <= l \bullet (-\text{DoPre } (\text{CONST whileAnno } b \text{ I V } c))$   
 $\text{CONST condCatch } c \text{ b SKIP} <= (-\text{DoPre } (\text{CONST condCatch } c \text{ b SKIP}))$   
 $\text{-Do } c <= -\text{DoPre } c$   
 $c;; d == \text{CONST Seq } c \text{ d}$   
 $\text{-guarantee } g = > (\text{CONST True}, g)$   
 $\text{-guaranteeStrip } g == \text{CONST guaranteeStripPair } (\text{CONST True}) \text{ g}$   
 $\text{-grd } g = > (\text{CONST False}, g)$   
 $\text{-grds } g \text{ gs} = > g \# \text{gs}$   
 $\text{-last-grd } g = > [g]$   
 $\text{-guards } gs \text{ c} == \text{CONST guards } gs \text{ c}$

$\{ | s. P | \} \quad == \{ | -\text{antiquoteCur}((=) s) \wedge P | \}$   
 $\{ | b | \} \quad == > \text{CONST Collect } (-\text{quote } b)$   
 $\text{IF } b \text{ THEN } c1 \text{ ELSE } c2 \text{ FI} = > \text{CONST Cond } \{ | b | \} \text{ c1 } c2$   
 $\text{IF } b \text{ THEN } c1 \text{ FI} \quad == \text{IF } b \text{ THEN } c1 \text{ ELSE SKIP FI}$

$IF_g b THEN c1 FI \quad == \quad IF_g b THEN c1 ELSE SKIP FI$

$-While\text{-}inv\text{-}var b I V c \quad ==> \quad CONST \text{ whileAnno } \{|b|\} I V c$   
 $-While\text{-}inv\text{-}var b I V (-DoPre c) <= \quad CONST \text{ whileAnno } \{|b|\} I V c$   
 $-While\text{-}inv b I c \quad == \quad -While\text{-}inv\text{-}var b I (CONST \text{ undefined}) c$   
 $-While b c \quad == \quad -While\text{-}inv b \{|CONST \text{ undefined}|\} c$

$-While\text{-}guard\text{-}inv\text{-}var gs b I V c \quad ==> \quad CONST \text{ whileAnnoG } gs \{|b|\} I V c$

$-While\text{-}guard\text{-}inv gs b I c \quad == \quad -While\text{-}guard\text{-}inv\text{-}var gs b I (CONST \text{ undefined}) c$   
 $-While\text{-}guard gs b c \quad == \quad -While\text{-}guard\text{-}inv gs b \{|CONST \text{ undefined}|\} c$

$-GuardedWhile\text{-}inv b I c \quad == \quad -GuardedWhile\text{-}inv\text{-}var b I (CONST \text{ undefined}) c$   
 $-GuardedWhile b c \quad == \quad -GuardedWhile\text{-}inv b \{|CONST \text{ undefined}|\} c$

$TRY c1 CATCH c2 END \quad == \quad CONST \text{ Catch } c1 c2$   
 $ANNO s. P c Q, A ==> \quad CONST \text{ specAnno } (\lambda s. P) (\lambda s. c) (\lambda s. Q) (\lambda s. A)$   
 $ANNO s. P c Q == \quad ANNO s. P c Q, \{\}$

$-WhileFix\text{-}inv\text{-}var b z I V c ==> \quad CONST \text{ whileAnnoFix } \{|b|\} (\lambda z. I) (\lambda z. V) (\lambda z. c)$   
 $-WhileFix\text{-}inv\text{-}var b z I V (-DoPre c) <= \quad -WhileFix\text{-}inv\text{-}var \{|b|\} z I V c$   
 $-WhileFix\text{-}inv b z I c == \quad -WhileFix\text{-}inv\text{-}var b z I (CONST \text{ undefined}) c$

$-GuardedWhileFix\text{-}inv b z I c == \quad -GuardedWhileFix\text{-}inv\text{-}var b z I (CONST \text{ undefined}) c$

$-GuardedWhileFix\text{-}inv\text{-}var b z I V c ==>$   
 $\quad -GuardedWhileFix\text{-}inv\text{-}var\text{-}hook \{|b|\} (\lambda z. I) (\lambda z. V) (\lambda z. c)$

$-WhileFix\text{-}guard\text{-}inv\text{-}var gs b z I V c ==>$   
 $\quad CONST \text{ whileAnnoGFix } gs \{|b|\} (\lambda z. I) (\lambda z. V)$   
 $(\lambda z. c)$

$-WhileFix\text{-}guard\text{-}inv\text{-}var gs b z I V (-DoPre c) <=$   
 $\quad -WhileFix\text{-}guard\text{-}inv\text{-}var gs \{|b|\} z I V c$   
 $-WhileFix\text{-}guard\text{-}inv gs b z I c == \quad -WhileFix\text{-}guard\text{-}inv\text{-}var gs b z I (CONST \text{ undefined}) c$

$LEMMA x c END == \quad CONST \text{ lem } x c$

#### translations

$(-switchcase V c) ==> (V, c)$   
 $(-switchcasesSingle b) ==> [b]$   
 $(-switchcasesCons b bs) ==> CONST \text{ Cons } b bs$   
 $(-Switch v vs) ==> CONST \text{ switch } (-quote v) vs$

#### parse-ast-translation <

$let$   
 $\quad fun tr c asts = Ast.mk-appl (Ast.Constant c) (map Ast.strip-positions asts)$   
 $in$

```

    [(@{syntax-const -antiquoteCur0}, K (tr @ {syntax-const -antiquoteCur})),
     (@{syntax-const -antiquoteOld0}, K (tr @ {syntax-const -antiquoteOld}))]
  end
>

```

#### print-ast-translation <

```

  let
    fun tr c asts = Ast.mk-appl (Ast.Constant c) asts
  in
    [(@{syntax-const -antiquoteCur}, K (tr @ {syntax-const -antiquoteCur})),
     (@{syntax-const -antiquoteOld}, K (tr @ {syntax-const -antiquoteOld}))]
  end
>

```

#### print-ast-translation <

```

  let
    fun dest-abs (Ast.Appl [Ast.Constant @ {syntax-const -abs}, x, t]) = (x, t)
      | dest-abs - = raise Match;
    fun spec-tr' [P, c, Q, A] =
      let
        val (x', P') = dest-abs P;
        val (-, c') = dest-abs c;
        val (-, Q') = dest-abs Q;
        val (-, A') = dest-abs A;
      in
        if (A' = Ast.Constant @ {const-syntax bot})
        then Ast.mk-appl (Ast.Constant @ {syntax-const -SpecNoAbrupt}) [x', P', c',
Q']
        else Ast.mk-appl (Ast.Constant @ {syntax-const -Spec}) [x', P', c', Q', A']
        end;
      fun whileAnnoFix-tr' [b, I, V, c] =
        let
          val (x', I') = dest-abs I;
          val (-, V') = dest-abs V;
          val (-, c') = dest-abs c;
        in
          Ast.mk-appl (Ast.Constant @ {syntax-const -WhileFix-inv-var}) [b, x', I',
V', c']
        end;
      in
        [(@ {const-syntax specAnno}, K spec-tr'),
         (@ {const-syntax whileAnnoFix}, K whileAnnoFix-tr')]
      end
  end
>

```

#### syntax

```

-faccess :: 'ref ⇒ ('ref ⇒ 'v) ⇒ 'v

```

( $\langle \dashrightarrow \rangle$  [65,1000] 100)

**syntax** (ASCII)

-faccess :: 'ref  $\Rightarrow$  ('ref  $\Rightarrow$  'v)  $\Rightarrow$  'v  
( $\langle \dashrightarrow \rangle$  [65,1000] 100)

**translations**

$p \rightarrow f \quad \Rightarrow \quad f p$   
 $g \rightarrow (-\text{antiquoteCur } f) \leq \text{-antiquoteCur } f g$

**nonterminal** *par* and *pars* and *actuals*

**syntax**

-par :: 'a  $\Rightarrow$  par ( $\langle \dashrightarrow \rangle$ )  
:: par  $\Rightarrow$  pars ( $\langle \dashrightarrow \rangle$ )  
-pars :: [par,pars]  $\Rightarrow$  pars ( $\langle \dashrightarrow, / \dashrightarrow \rangle$ )  
-actuals :: pars  $\Rightarrow$  actuals ( $\langle '(-)' \rangle$ )  
-actuals-empty :: actuals ( $\langle '()' \rangle$ )

**syntax** -Call :: 'p  $\Rightarrow$  actuals  $\Rightarrow$  (('a,string,'f) com) ( $\langle \text{CALL} \dashrightarrow$  [1000,1000] 21)  
-GuardedCall :: 'p  $\Rightarrow$  actuals  $\Rightarrow$  (('a,string,'f) com) ( $\langle \text{CALL}_g \dashrightarrow$  [1000,1000] 21)  
-CallAss:: 'a  $\Rightarrow$  'p  $\Rightarrow$  actuals  $\Rightarrow$  (('a,string,'f) com)  
( $\langle \dashrightarrow ::= \text{CALL} \dashrightarrow$  [30,1000,1000] 21)  
-Call-exn :: 'p  $\Rightarrow$  actuals  $\Rightarrow$  (('a,string,'f) com) ( $\langle \text{CALL}_e \dashrightarrow$  [1000,1000] 21)  
-CallAss-exn:: 'a  $\Rightarrow$  'p  $\Rightarrow$  actuals  $\Rightarrow$  (('a,string,'f) com)  
( $\langle \dashrightarrow ::= \text{CALL}_e \dashrightarrow$  [30,1000,1000] 21)  
-Proc :: 'p  $\Rightarrow$  actuals  $\Rightarrow$  (('a,string,'f) com) ( $\langle \text{PROC} \dashrightarrow$  21)  
-ProcAss:: 'a  $\Rightarrow$  'p  $\Rightarrow$  actuals  $\Rightarrow$  (('a,string,'f) com)  
( $\langle \dashrightarrow ::= \text{PROC} \dashrightarrow$  [30,1000,1000] 21)  
-GuardedCallAss:: 'a  $\Rightarrow$  'p  $\Rightarrow$  actuals  $\Rightarrow$  (('a,string,'f) com)  
( $\langle \dashrightarrow ::= \text{CALL}_g \dashrightarrow$  [30,1000,1000] 21)  
-DynCall :: 'p  $\Rightarrow$  actuals  $\Rightarrow$  (('a,string,'f) com) ( $\langle \text{DYNCALL} \dashrightarrow$  [1000,1000] 21)  
-GuardedDynCall :: 'p  $\Rightarrow$  actuals  $\Rightarrow$  (('a,string,'f) com) ( $\langle \text{DYNCALL}_g \dashrightarrow$  [1000,1000] 21)  
-DynCallAss:: 'a  $\Rightarrow$  'p  $\Rightarrow$  actuals  $\Rightarrow$  (('a,string,'f) com)  
( $\langle \dashrightarrow ::= \text{DYNCALL} \dashrightarrow$  [30,1000,1000] 21)  
-DynCall-exn :: 'p  $\Rightarrow$  actuals  $\Rightarrow$  (('a,string,'f) com) ( $\langle \text{DYNCALL}_e \dashrightarrow$  [1000,1000] 21)  
-DynCallAss-exn:: 'a  $\Rightarrow$  'p  $\Rightarrow$  actuals  $\Rightarrow$  (('a,string,'f) com)  
( $\langle \dashrightarrow ::= \text{DYNCALL}_e \dashrightarrow$  [30,1000,1000] 21)  
-GuardedDynCallAss:: 'a  $\Rightarrow$  'p  $\Rightarrow$  actuals  $\Rightarrow$  (('a,string,'f) com)  
( $\langle \dashrightarrow ::= \text{DYNCALL}_g \dashrightarrow$  [30,1000,1000] 21)  
-Bind:: ['s  $\Rightarrow$  'v, idt, 'v  $\Rightarrow$  ('s,'p,'f) com]  $\Rightarrow$  ('s,'p,'f) com  
( $\langle \dashrightarrow \gg \dashrightarrow \dashrightarrow$  [22,1000,21] 21)

```

-bseq::('s,'p,'f) com => ('s,'p,'f) com => ('s,'p,'f) com
  (<->>/ -> [22, 21] 21)
-FCall :: ['p,actuals,idt, (('a,string,'f) com)] => (('a,string,'f) com)
  (<CALL -- >> -./ -> [1000,1000,1000,21] 21)

```

### translations

```

-Bind e i c == CONST bind (-quote e) (λi. c)
-FCall p acts i c == -FCall p acts (λi. c)
-bseq c d == CONST bseq c d

```

### nonterminal modifyargs

#### syntax

```

-may-modify :: ['a,'a,modifyargs] => bool
  (<- may'-only'-modify'-globals - in [-] > [100,100,0] 100)
-may-not-modify :: ['a,'a] => bool
  (<- may'-not'-modify'-globals -> [100,100] 100)
-may-modify-empty :: ['a,'a] => bool
  (<- may'-only'-modify'-globals - in [] > [100,100] 100)
-modifyargs :: [id,modifyargs] => modifyargs (<-./ ->)
  :: id => modifyargs (<->)

```

### translations

```

s may-only-modify-globals Z in [] => s may-not-modify-globals Z

```

**definition** *Let'* :: ['a, 'a => 'b] => 'b  
**where** *Let'* = *Let*

**ML-file** <hoare-syntax.ML>

### parse-translation <

```

let
  val argsC = @{syntax-const -modifyargs};
  val globalsN = globals;
  val ex = @{const-syntax mex};
  val eq = @{const-syntax meq};
  val varn = Hoare.varname;

  fun extract-args (Const (argsC,-)$Free (n,-)$t) = varn n::extract-args t
    | extract-args (Free (n,-)) = [varn n]
    | extract-args t = raise TERM (extract-args, [t])

  fun idx [] y = error idx: element not in list

```



```

| idx (x::xs) y = if x=y then 0 else (idx xs y)+1

fun gen-update ctxt names (name,t) =
  Hoare-Syntax.update-comp ctxt NONE [] false true name (Bound (idx names
name)) t

fun gen-updates ctxt names t = Library.foldr (gen-update ctxt names) (names,t)

fun gen-ex (name,t) = Syntax.const ex $ Abs (name,dummyT,t)

fun gen-exs names t = Library.foldr gen-ex (names,t)

fun tr ctxt s Z names =
  let val upds = gen-updates ctxt (rev names) (Syntax.free globalsN$Z);
      val eq   = Syntax.const eq $ (Syntax.free globalsN$s) $ upds;
      in gen-exs names eq end;

fun may-modify-tr ctxt [s,Z,names] = tr ctxt s Z
                                   (sort-strings (extract-args names))
fun may-not-modify-tr ctxt [s,Z] = tr ctxt s Z []
in
  [(@{syntax-const -may-modify}, may-modify-tr),
   (@{syntax-const -may-not-modify}, may-not-modify-tr)]
end
>

```

### print-translation <

```

let
  val argsC = @{syntax-const -modifyargs};
  val chop = Hoare.chopsfx Hoare.deco;

  fun get-state ( - $ - $ t) = get-state t (* for record-updates*)
    | get-state ( - $ - $ - $ - $ t) = get-state t (* for statespace-updates *)
    | get-state (globals$(s as Const (@{syntax-const -free},-) $ Free -)) = s
    | get-state (globals$(s as Const (@{syntax-const -bound},-) $ Free -)) = s
    | get-state (globals$(s as Const (@{syntax-const -var},-) $ Var -)) = s
    | get-state (globals$(s as Const -)) = s
    | get-state (globals$(s as Free -)) = s
    | get-state (globals$(s as Bound -)) = s
    | get-state t = raise Match;

  fun mk-args [n] = Syntax.free (chop n)
    | mk-args (n::ns) = Syntax.const argsC $ Syntax.free (chop n) $ mk-args ns
    | mk-args - = raise Match;

  fun tr' names (Abs (n,-,t)) = tr' (n::names) t
    | tr' names (Const (@{const-syntax mex},-) $ t) = tr' names t

```

```

| tr' names (Const (@{const-syntax meq},-) $ (globals$s) $ upd) =
  let val Z = get-state upd;

  in (case names of
      [] => Syntax.const @{{syntax-const -may-not-modify}} $ s $ Z
    | xs => Syntax.const @{{syntax-const -may-modify}} $ s $ Z $ mk-args
    (rev names))
  end;

  fun may-modify-tr' [t] = tr' [] t
  fun may-not-modify-tr' [-$s,-$Z] = Syntax.const @{{syntax-const -may-not-modify}}
    $ s $ Z
  in
    [(@{{const-syntax mex}}, K may-modify-tr'),
     (@{{const-syntax meq}}, K may-not-modify-tr')]
  end
>

```

```

parse-translation <
  [(@{{syntax-const -antiquoteCur}},
    K (Hoare-Syntax.antiquote-varname-tr @{{syntax-const -antiquoteCur}}))]
>

```

```

parse-translation <
  [(@{{syntax-const -antiquoteOld}}, Hoare-Syntax.antiquoteOld-tr),
   (@{{syntax-const -Call}}, Hoare-Syntax.call-tr false false []),
   (@{{syntax-const -Call-exn}}, Hoare-Syntax.call-tr false true []),
   (@{{syntax-const -FCall}}, Hoare-Syntax.fcall-tr),
   (@{{syntax-const -CallAss}}, Hoare-Syntax.call-ass-tr false false []),
   (@{{syntax-const -CallAss-exn}}, Hoare-Syntax.call-ass-tr false true []),
   (@{{syntax-const -GuardedCall}}, Hoare-Syntax.call-tr false true []),
   (@{{syntax-const -GuardedCallAss}}, Hoare-Syntax.call-ass-tr false true []),
   (@{{syntax-const -Proc}}, Hoare-Syntax.proc-tr),
   (@{{syntax-const -ProcAss}}, Hoare-Syntax.proc-ass-tr),
   (@{{syntax-const -DynCall}}, Hoare-Syntax.call-tr true false []),
   (@{{syntax-const -DynCall-exn}}, Hoare-Syntax.call-tr true true []),
   (@{{syntax-const -DynCallAss}}, Hoare-Syntax.call-ass-tr true false []),
   (@{{syntax-const -DynCallAss-exn}}, Hoare-Syntax.call-ass-tr true true []),
   (@{{syntax-const -GuardedDynCall}}, Hoare-Syntax.call-tr true true []),
   (@{{syntax-const -GuardedDynCallAss}}, Hoare-Syntax.call-ass-tr true true []),
   (@{{syntax-const -BasicBlock}}, Hoare-Syntax.basic-assigns-tr)]
>

```

```

parse-translation <
  let
    fun quote-tr ctxt [t] = Hoare-Syntax.quote-tr ctxt @{syntax-const -antiquoteCur}
  t
    | quote-tr ctxt ts = raise TERM (quote-tr, ts);
  in [(@{syntax-const -quote}, quote-tr)] end
>

```

```

parse-translation <
  [(@{syntax-const -Assign}, Hoare-Syntax.assign-tr),
   (@{syntax-const -raise}, Hoare-Syntax.raise-tr),
   (@{syntax-const -New}, Hoare-Syntax.new-tr),
   (@{syntax-const -NNew}, Hoare-Syntax.nnew-tr),
   (@{syntax-const -GuardedAssign}, Hoare-Syntax.guarded-Assign-tr),
   (@{syntax-const -GuardedNew}, Hoare-Syntax.guarded-New-tr),
   (@{syntax-const -GuardedNNew}, Hoare-Syntax.guarded-NNew-tr),
   (@{syntax-const -GuardedWhile-inv-var}, Hoare-Syntax.guarded-While-tr),
   (@{syntax-const -GuardedWhileFix-inv-var-hook}, Hoare-Syntax.guarded-WhileFix-tr),
   (@{syntax-const -GuardedCond}, Hoare-Syntax.guarded-Cond-tr),
   (@{syntax-const -Basic}, Hoare-Syntax.basic-tr)]
>

```

```

parse-translation <
  [(@{syntax-const -Init}, Hoare-Syntax.init-tr),
   (@{syntax-const -Loc}, Hoare-Syntax.loc-tr)]
>

```

```

print-translation <
  [(@{const-syntax Basic}, Hoare-Syntax.assign-tr'),
   (@{const-syntax raise}, Hoare-Syntax.raise-tr'),
   (@{const-syntax Basic}, Hoare-Syntax.new-tr'),
   (@{const-syntax Basic}, Hoare-Syntax.init-tr'),
   (@{const-syntax Spec}, Hoare-Syntax.nnew-tr'),
   (@{const-syntax block}, Hoare-Syntax.loc-tr'),
   (@{const-syntax Collect}, Hoare-Syntax.assert-tr'),
   (@{const-syntax Cond}, Hoare-Syntax.bexp-tr' -Cond),
   (@{const-syntax switch}, Hoare-Syntax.switch-tr'),
   (@{const-syntax Basic}, Hoare-Syntax.basic-tr'),
   (@{const-syntax guards}, Hoare-Syntax.guards-tr'),
   (@{const-syntax whileAnnoG}, Hoare-Syntax.whileAnnoG-tr'),
   (@{const-syntax whileAnnoGFix}, Hoare-Syntax.whileAnnoGFix-tr'),
   (@{const-syntax bind}, Hoare-Syntax.bind-tr')]
>

```

```

print-translation <
  let
    fun spec-tr' ctxt ((coll as Const -)$
      ((splt as Const -) $ (t as (Abs (s,T,p))))::ts) =
      let
        fun selector (Const (c, T)) = Hoare.is-state-var c
          | selector (Const (@{syntax-const -free}, -) $ (Free (c, T))) =
            Hoare.is-state-var c
          | selector - = false;
      in
        if Hoare-Syntax.antiquote-applied-only-to selector p then
          Syntax.const @{const-syntax Spec} $ coll $
            (splt $ Hoare-Syntax.quote-mult-tr' ctxt selector
              Hoare-Syntax.antiquoteCur Hoare-Syntax.antiquoteOld (Abs
                (s,T,t)))
          else raise Match
        end
      | spec-tr' - ts = raise Match
    in [(@{const-syntax Spec}, spec-tr')] end
  >

```

```

syntax
-Measure:: ('a ⇒ nat) ⇒ ('a × 'a) set
  (<MEASURE -> [22] 1)
-Mlex:: ('a ⇒ nat) ⇒ ('a × 'a) set ⇒ ('a × 'a) set
  (infixr <*>MLEX*> 30)

```

```

syntax-consts
-Measure == measure and
-Mlex == mlex-prod

```

```

translations
MEASURE f      => (CONST measure) (-quote f)
f <*>MLEX*> r  => (-quote f) <*>mlex*> r

```

```

print-translation <
  let
    fun selector (Const (c,T)) = Hoare.is-state-var c
      | selector - = false;

    fun measure-tr' ctxt ((t as (Abs (-,-,p))))::ts) =
      if Hoare-Syntax.antiquote-applied-only-to selector p
      then Hoare-Syntax.app-quote-tr' ctxt (Syntax.const @{syntax-const -Measure})
      (t::ts)
      else raise Match
  end

```

```

| measure-tr' - - = raise Match

fun mlex-tr' ctxt ((t as (Abs (-,-, p)))::r::ts) =
  if Hoare-Syntax.antiquote-applied-only-to selector p
  then Hoare-Syntax.app-quote-tr' ctxt (Syntax.const @{syntax-const -Mlex})
(t::r::ts)
  else raise Match
| mlex-tr' - - = raise Match

in
  [(@{const-syntax measure}, measure-tr'),
   (@{const-syntax mlex-prod}, mlex-tr')]
end
>

```

```

print-translation <
  [(@{const-syntax call}, Hoare-Syntax.call-tr'),
   (@{const-syntax dynCall}, Hoare-Syntax.dyn-call-tr'),
   (@{const-syntax call-exn}, Hoare-Syntax.call-exn-tr'),
   (@{const-syntax dynCall-exn}, Hoare-Syntax.dyn-call-exn-tr'),
   (@{const-syntax fcall}, Hoare-Syntax.fcall-tr'),
   (@{const-syntax Call}, Hoare-Syntax.proc-tr')]
>

```

**end**

## 17 Examples using the Verification Environment

**theory** *VcgEx* **imports** *../HeapList ../Vcg* **begin**

Some examples, especially the single-step Isar proofs are taken from `HOL/Isar_examples/HoareEx.th`

### 17.1 State Spaces

First of all we provide a store of program variables that occur in the programs considered later. Slightly unexpected things may happen when attempting to work with undeclared variables.

```

record 'g vars = 'g state +
  A' :: nat
  I' :: nat
  M' :: nat
  N' :: nat
  R' :: nat
  S' :: nat
  B' :: bool

```

*Arr-'* :: *nat list*  
*Abr-'* :: *string*

We decorate the state components in the record with the suffix *-'*, to avoid cluttering the namespace with the simple names that could no longer be used for logical variables otherwise.

We will first consider programs without procedures, later on we will regard procedures without global variables and finally we will get the full pictures: mutually recursive procedures with global variables (including heap).

## 17.2 Basic Examples

We look at few trivialities involving assignment and sequential composition, in order to get an idea of how to work with our formulation of Hoare Logic.

Using the basic rule directly is a bit cumbersome.

```
lemma  $\Gamma \vdash \{|'N = 5|\} \ 'N ::= 2 * 'N \{|'N = 10|\}$ 
  apply (rule HoarePartial.Basic) apply simp
done
```

If we refer to components (variables) of the state-space of the program we always mark these with *'*. It is the acute-symbol and is present on most keyboards. So all program variables are marked with the acute and all logical variables are not. The assertions of the Hoare tuple are ordinary Isabelle sets. As we usually want to refer to the state space in the assertions, we provide special brackets for them. They can be written as  $\{| \ | \}$  in ASCII or  $\{ | \}$  with symbols. Internally marking variables has two effects. First of all we refer to the implicit state and secondary we get rid of the suffix *-'*. So the assertion  $\{|'N = 5|\}$  internally gets expanded to  $\{s. N-'s = 5\}$  written in ordinary set comprehension notation of Isabelle. It describes the set of states where the *N-'* component is equal to 5.

Certainly we want the state modification already done, e.g. by simplification. The *vcg* method performs the basic state update for us; we may apply the Simplifier afterwards to achieve “obvious” consequences as well.

```
lemma  $\Gamma \vdash \{True\} \ 'N ::= 10 \{|'N = 10|\}$ 
  by vcg
```

```
lemma  $\Gamma \vdash \{2 * 'N = 10\} \ 'N ::= 2 * 'N \{|'N = 10|\}$ 
  by vcg
```

```
lemma  $\Gamma \vdash \{|'N = 5|\} \ 'N ::= 2 * 'N \{|'N = 10|\}$ 
  apply vcg
  apply simp
done
```

**lemma**  $\Gamma \vdash \{\! \{ 'N + 1 = a + 1 \} \! \}$   $'N ::= 'N + 1$   $\{\! \{ 'N = a + 1 \} \! \}$   
**by** *vcg*

**lemma**  $\Gamma \vdash \{\! \{ 'N = a \} \! \}$   $'N ::= 'N + 1$   $\{\! \{ 'N = a + 1 \} \! \}$   
**by** *vcg*

**lemma**  $\Gamma \vdash \{\! \{ a = a \wedge b = b \} \! \}$   $'M ::= a;; 'N ::= b$   $\{\! \{ 'M = a \wedge 'N = b \} \! \}$   
**by** *vcg*

**lemma**  $\Gamma \vdash \{\! \{ True \} \! \}$   $'M ::= a;; 'N ::= b$   $\{\! \{ 'M = a \wedge 'N = b \} \! \}$   
**by** *vcg*

**lemma**  $\Gamma \vdash \{\! \{ 'M = a \wedge 'N = b \} \! \}$   
 $'I ::= 'M;; 'M ::= 'N;; 'N ::= 'I$   
 $\{\! \{ 'M = b \wedge 'N = a \} \! \}$   
**by** *vcg*

We can also perform verification conditions generation step by step by using the *vcg-step* method.

**lemma**  $\Gamma \vdash \{\! \{ 'M = a \wedge 'N = b \} \! \}$   
 $'I ::= 'M;; 'M ::= 'N;; 'N ::= 'I$   
 $\{\! \{ 'M = b \wedge 'N = a \} \! \}$   
**apply** *vcg-step*  
**apply** *vcg-step*  
**apply** *vcg-step*  
**apply** *vcg-step*  
**done**

It is important to note that statements like the following one can only be proven for each individual program variable. Due to the extra-logical nature of record fields, we cannot formulate a theorem relating record selectors and updates schematically.

**lemma**  $\Gamma \vdash \{\! \{ 'N = a \} \! \}$   $'N ::= 'N$   $\{\! \{ 'N = a \} \! \}$   
**by** *vcg*

**lemma**  $\Gamma \vdash \{s. x-' s = a\}$  (*Basic*  $(\lambda s. x\text{'-update } (x\text{' } s) s)$ )  $\{s. x\text{' } s = a\}$   
**oops**

In the following assignments we make use of the consequence rule in order to achieve the intended precondition. Certainly, the *vcg* method is able to handle this case, too.

**lemma**  $\Gamma \vdash \{\! \{ 'M = 'N \} \! \}$   $'M ::= 'M + 1$   $\{\! \{ 'M \neq 'N \} \! \}$   
**proof** –  
**have**  $\{\! \{ 'M = 'N \} \! \} \subseteq \{\! \{ 'M + 1 \neq 'N \} \! \}$

by *auto*  
 also have  $\Gamma \vdash \dots \ 'M ::= \ 'M + 1 \ \{ \ 'M \neq \ 'N \}$   
 by *vcg*  
 finally show *?thesis* .  
 qed

**lemma**  $\Gamma \vdash \{ \ 'M = \ 'N \} \ 'M ::= \ 'M + 1 \ \{ \ 'M \neq \ 'N \}$   
**proof** –  
 have  $\bigwedge m \ n :: \text{nat. } m = n \longrightarrow m + 1 \neq n$   
 — inclusion of assertions expressed in “pure” logic,  
 — without mentioning the state space  
 by *simp*  
 also have  $\Gamma \vdash \{ \ 'M + 1 \neq \ 'N \} \ 'M ::= \ 'M + 1 \ \{ \ 'M \neq \ 'N \}$   
 by *vcg*  
 finally show *?thesis* .  
 qed

**lemma**  $\Gamma \vdash \{ \ 'M = \ 'N \} \ 'M ::= \ 'M + 1 \ \{ \ 'M \neq \ 'N \}$   
 apply *vcg*  
 apply *simp*  
 done

### 17.3 Multiplication by Addition

We now do some basic examples of actual WHILE programs. This one is a loop for calculating the product of two natural numbers, by iterated addition. We first give detailed structured proof based on single-step Hoare rules.

**lemma**  $\Gamma \vdash \{ \ 'M = 0 \wedge \ 'S = 0 \}$   
     WHILE  $\ 'M \neq a$   
     DO  $\ 'S ::= \ 'S + b;; \ 'M ::= \ 'M + 1$  OD  
      $\{ \ 'S = a * b \}$   
**proof** –  
 let  $\Gamma \vdash - \ ?while - = \ ?thesis$   
 let  $\{ \ '?inv \} = \{ \ 'S = \ 'M * b \}$   
  
 have  $\{ \ 'M = 0 \ \& \ 'S = 0 \} \subseteq \{ \ '?inv \}$  by *auto*  
 also have  $\Gamma \vdash \dots \ ?while \ \{ \ '?inv \wedge \neg (\ 'M \neq a) \}$   
**proof**  
 let  $?c = \ 'S ::= \ 'S + b;; \ 'M ::= \ 'M + 1$   
 have  $\{ \ '?inv \wedge \ 'M \neq a \} \subseteq \{ \ 'S + b = (\ 'M + 1) * b \}$   
 by *auto*  
 also have  $\Gamma \vdash \dots \ ?c \ \{ \ '?inv \}$  by *vcg*  
 finally show  $\Gamma \vdash \{ \ '?inv \wedge \ 'M \neq a \} \ ?c \ \{ \ '?inv \}$  .  
 qed  
 also have  $\{ \ '?inv \wedge \neg (\ 'M \neq a) \} \subseteq \{ \ 'S = a * b \}$  by *auto*  
 finally show *?thesis* by *blast*  
 qed

The subsequent version of the proof applies the *vcg* method to reduce the



Hoare statement to a purely logical problem that can be solved fully automatically. Note that we have to specify the **WHILE** loop invariant in the original statement.

```

lemma  $\Gamma \vdash \{ \prime M = 0 \wedge \prime S = 0 \}$ 
  WHILE  $\prime M \neq a$ 
  INV  $\{ \prime S = \prime M * b \}$ 
  DO  $\prime S ::= \prime S + b;; \prime M ::= \prime M + 1$  OD
   $\{ \prime S = a * b \}$ 
apply vcg
apply auto
done

```

Here some examples of “breaking” out of a loop

```

lemma  $\Gamma \vdash \{ \prime M = 0 \wedge \prime S = 0 \}$ 
  TRY
    WHILE True
    INV  $\{ \prime S = \prime M * b \}$ 
    DO IF  $\prime M = a$  THEN THROW ELSE  $\prime S ::= \prime S + b;; \prime M ::= \prime M + 1$ 
  FI OD
  CATCH
    SKIP
  END
   $\{ \prime S = a * b \}$ 
apply vcg
apply auto
done

```

```

lemma  $\Gamma \vdash \{ \prime M = 0 \wedge \prime S = 0 \}$ 
  TRY
    WHILE True
    INV  $\{ \prime S = \prime M * b \}$ 
    DO IF  $\prime M = a$  THEN  $\prime Abr ::= \text{"Break"};;$  THROW
      ELSE  $\prime S ::= \prime S + b;; \prime M ::= \prime M + 1$ 
    FI
  OD
  CATCH
    IF  $\prime Abr = \text{"Break"}$  THEN SKIP ELSE Throw FI
  END
   $\{ \prime S = a * b \}$ 
apply vcg
apply auto
done

```

Some more syntactic sugar, the label statement  $\dots \cdot \dots$  as shorthand for the *TRY-CATCH* above, and the *RAISE* for an state-update followed by a *THROW*.

```

lemma  $\Gamma \vdash \{ \prime M = 0 \wedge \prime S = 0 \}$ 
   $\{ \prime Abr = \text{"Break"} \} \cdot$  WHILE True INV  $\{ \prime S = \prime M * b \}$ 

```

```

DO IF 'M = a THEN RAISE 'Abr ::= "Break"
  ELSE 'S ::= 'S + b;; 'M ::= 'M + 1
FI
OD
{ 'S = a * b }
apply vcg
apply auto
done

```

```

lemma  $\Gamma \vdash \{ 'M = 0 \wedge 'S = 0 \}$ 
  TRY
    WHILE True
      INV  $\{ 'S = 'M * b \}$ 
      DO IF 'M = a THEN RAISE 'Abr ::= "Break"
        ELSE 'S ::= 'S + b;; 'M ::= 'M + 1
      FI
    OD
  CATCH
    IF 'Abr = "Break" THEN SKIP ELSE Throw FI
  END
  { 'S = a * b }
apply vcg
apply auto
done

```

```

lemma  $\Gamma \vdash \{ 'M = 0 \wedge 'S = 0 \}$ 
  { 'Abr = "Break" } · WHILE True
  INV  $\{ 'S = 'M * b \}$ 
  DO IF 'M = a THEN RAISE 'Abr ::= "Break"
    ELSE 'S ::= 'S + b;; 'M ::= 'M + 1
  FI
  OD
  { 'S = a * b }
apply vcg
apply auto
done

```

Blocks

```

lemma  $\Gamma \vdash \{ 'T = i \} \text{LOC } 'T;; 'T ::= 2 \text{COL } \{ 'T \leq i \}$ 
  apply vcg
  by simp

```

```

lemma  $\Gamma \vdash \{ 'N = n \} \text{LOC } 'N ::= 10;; 'N ::= 'N + 2 \text{COL } \{ 'N = n \}$ 
  by vcg

```

```

lemma  $\Gamma \vdash \{ 'N = n \} \text{LOC } 'N ::= 10, 'M;; 'N ::= 'N + 2 \text{COL } \{ 'N = n \}$ 
  by vcg

```

## 17.4 Summing Natural Numbers

We verify an imperative program to sum natural numbers up to a given limit. First some functional definition for proper specification of the problem.

**primrec**

$sum :: (nat \Rightarrow nat) \Rightarrow nat \Rightarrow nat$

**where**

$sum\ f\ 0 = 0$

|  $sum\ f\ (Suc\ n) = f\ n + sum\ f\ n$

**syntax**

$-sum :: idt \Rightarrow nat \Rightarrow nat \Rightarrow nat$

$(\langle SUMM\ -<-. \rightarrow [0, 0, 10]\ 10)$

**syntax-consts**

$-sum == sum$

**translations**

$SUMM\ j<k. b == CONST\ sum\ (\lambda j. b)\ k$

The following proof is quite explicit in the individual steps taken, with the *vcg* method only applied locally to take care of assignment and sequential composition. Note that we express intermediate proof obligation in pure logic, without referring to the state space.

**theorem**  $\Gamma \vdash \{True\}$

$\ 'S ::= 0;;\ 'T ::= 1;;$

$\ WHILE\ 'T \neq n$

$\ DO$

$\ 'S ::= 'S + 'T;;$

$\ 'T ::= 'T + 1$

$\ OD$

$\ \{ 'S = (SUMM\ j<n. j) \}$

(is  $\Gamma \vdash - (-; ?while) -$ )

**proof** –

**let**  $?sum = \lambda k. SUMM\ j<k. j$

**let**  $?inv = \lambda s\ i. s = ?sum\ i$

**have**  $\Gamma \vdash \{True\}\ 'S ::= 0;;\ 'T ::= 1\ \{?inv\ 'S\ 'T\}$

**proof** –

**have**  $True \longrightarrow 0 = ?sum\ 1$

**by** *simp*

**also have**  $\Gamma \vdash \{...\}\ 'S ::= 0;;\ 'T ::= 1\ \{?inv\ 'S\ 'T\}$

**by** *vcg*

**finally show** *?thesis* .

**qed**

**also have**  $\Gamma \vdash \{?inv\ 'S\ 'T\}\ ?while\ \{?inv\ 'S\ 'T \wedge \neg\ 'T \neq n\}$

**proof**

**let**  $?body = 'S ::= 'S + 'T;;\ 'T ::= 'T + 1$

**have**  $\bigwedge s\ i. ?inv\ s\ i \wedge i \neq n \longrightarrow ?inv\ (s + i)\ (i + 1)$

**by** *simp*

```

also have  $\Gamma \vdash \{\! \{ 'S + 'I = ?sum ('I + 1) \} \! \}$  ?body  $\{\! \{ ?inv 'S 'I \} \! \}$ 
by vcg
finally show  $\Gamma \vdash \{\! \{ ?inv 'S 'I \wedge 'I \neq n \} \! \}$  ?body  $\{\! \{ ?inv 'S 'I \} \! \}$  .
qed
also have  $\bigwedge s i. s = ?sum i \wedge \neg i \neq n \longrightarrow s = ?sum n$ 
by simp
finally show ?thesis .
qed

```

The next version uses the *vcg* method, while still explaining the resulting proof obligations in an abstract, structured manner.

```

theorem  $\Gamma \vdash \{\! \{ True \} \! \}$ 
   $'S ::= 0;; 'I ::= 1;;$ 
  WHILE  $'I \neq n$ 
  INV  $\{\! \{ 'S = (SUMM j < 'I. j) \} \! \}$ 
  DO
     $'S ::= 'S + 'I;;$ 
     $'I ::= 'I + 1$ 
  OD
   $\{\! \{ 'S = (SUMM j < n. j) \} \! \}$ 

```

```

proof –
let  $?sum = \lambda k. SUMM j < k. j$ 
let  $?inv = \lambda s i. s = ?sum i$ 

show ?thesis
proof vcg
  show  $?inv 0 1$  by simp
next
  fix  $i s$  assume  $?inv s i i \neq n$ 
  thus  $?inv (s + i) (i + 1)$  by simp
next
  fix  $i s$  assume  $x: ?inv s i \neg i \neq n$ 
  thus  $s = ?sum n$  by simp
qed
qed

```

Certainly, this proof may be done fully automatically as well, provided that the invariant is given beforehand.

```

theorem  $\Gamma \vdash \{\! \{ True \} \! \}$ 
   $'S ::= 0;; 'I ::= 1;;$ 
  WHILE  $'I \neq n$ 
  INV  $\{\! \{ 'S = (SUMM j < 'I. j) \} \! \}$ 
  DO
     $'S ::= 'S + 'I;;$ 
     $'I ::= 'I + 1$ 
  OD
   $\{\! \{ 'S = (SUMM j < n. j) \} \! \}$ 
apply vcg
apply auto

```

**done**

## 17.5 SWITCH

```
lemma  $\Gamma \vdash \{\!| N = 5 \!\!| \} \text{ SWITCH } 'B$   
       $\{True\} \Rightarrow 'N ::= 6$   
       $| \{False\} \Rightarrow 'N ::= 7$   
      END  
       $\{\!| N > 5 \!\!| \}$ 
```

```
apply vcg  
apply simp  
done
```

```
lemma  $\Gamma \vdash \{\!| N = 5 \!\!| \} \text{ SWITCH } 'N$   
       $\{v. v < 5\} \Rightarrow 'N ::= 6$   
       $| \{v. v \geq 5\} \Rightarrow 'N ::= 7$   
      END  
       $\{\!| N > 5 \!\!| \}$ 
```

```
apply vcg  
apply simp  
done
```

## 17.6 (Mutually) Recursive Procedures

### 17.6.1 Factorial

We want to define a procedure for the factorial. We first define a HOL functions that calculates it to specify the procedure later on.

```
primrec fac:: nat  $\Rightarrow$  nat  
where  
fac 0 = 1 |  
fac (Suc n) = (Suc n) * fac n
```

```
lemma fac-simp [simp]: 0 < i  $\implies$  fac i = i * fac (i - 1)  
by (cases i) simp-all
```

Now we define the procedure

**procedures**

```
Fac (N|R) = IF 'N = 0 THEN 'R ::= 1  
           ELSE 'R ::= CALL Fac(N - 1);;  
           'R ::= 'N * 'R  
FI
```

A procedure is given by the signature of the procedure followed by the procedure body. The signature consists of the name of the procedure and a list of parameters. The parameters in front of the pipe | are value parameters and behind the pipe are the result parameters. Value parameters model call by value semantics. The value of a result parameter at the end of the procedure is passed back to the caller.

Behind the scenes the *procedures* command provides us convenient syntax for procedure calls, defines a constant for the procedure body (named *Fac-body*) and creates some locales. The purpose of locales is to set up logical contexts to support modular reasoning. A locale is named *Fac-impl* and extends the *hoare* locale with a theorem  $\Gamma \text{ "Fac" } = \text{Fac-body}$  that simply states how the procedure is defined in the procedure context. Check out the locales. The purpose of the locales is to give us easy means to setup the context in which we will prove programs correct. In these locales the procedure context  $\Gamma$  is fixed. So always use this letter in procedure specifications. This is crucial, if we later on prove some tuples under the assumption of some procedure specifications.

```
thm Fac-body.Fac-body-def
print-locale Fac-impl
```

To see how a call is syntactically translated you can switch off the printing translation via the configuration option *hoare-use-call-tr'*

```
context Fac-impl
begin
```

$'M ::= \text{CALL } \text{Fac}('N)$  is internally:

```
declare [[hoare-use-call-tr' = false]]
```

```
call ( $\lambda s. s(N' := N' s)$ ) Fac'-proc ( $\lambda s t. s(\text{globals} := \text{globals } t)$ ) ( $\lambda i t. 'M ::= R' t$ )
```

```
term CALL Fac('N, 'M)
```

```
declare [[hoare-use-call-tr' = true]]
```

```
end
```

Now let us prove that *Fac* meets its specification.

Procedure specifications are ordinary Hoare tuples. We use the parameterless call for the specification;  $'R ::= \text{PROC } \text{Fac}('N)$  is syntactic sugar for *Call "Fac"*. This emphasises that the specification describes the internal behaviour of the procedure, whereas parameter passing corresponds to the procedure call.

```
lemma (in Fac-impl)
```

```
shows  $\forall n. \Gamma, \Theta \vdash \{N=n\} \text{PROC } \text{Fac}('N, 'R) \{R = \text{fac } n\}$ 
```

```
apply (hoare-rule HoarePartial.ProcRec1)
```

```
apply vcg
```

```
apply simp
```

```
done
```

Since the factorial was implemented recursively, the main ingredient of this proof is, to assume that the specification holds for the recursive call of *Fac* and prove the body correct. The assumption for recursive calls is added to the context by the rule *HoarePartial.ProcRec1* (also derived from general rule for mutually recursive procedures):

$$\begin{aligned} & \llbracket \forall Z. \Gamma, \Theta \cup (\bigcup_Z \{(P\ Z, p, Q\ Z, A\ Z)\}) \vdash_{/F} (P\ Z) \text{ the } (\Gamma\ p) (Q\ Z), (A\ Z); \\ & \quad p \in \text{dom } \Gamma \rrbracket \\ & \implies \forall Z. \Gamma, \Theta \vdash_{/F} (P\ Z) \text{ Call } p (Q\ Z), (A\ Z) \end{aligned}$$

The verification condition generator will infer the specification out of the context when it encounters a recursive call of the factorial.

We can also step through verification condition generation. When the verification condition generator encounters a procedure call it tries to use the rule *ProcSpec*. To be successful there must be a specification of the procedure in the context.

```

lemma (in Fac-impl)
  shows  $\forall n. \Gamma \vdash \{N=n\} \ 'R ::= PROC\ Fac(N) \ \{R = fac\ n\}$ 
  apply (hoare-rule HoarePartial.ProcRec1)
  apply vcg-step
  apply vcg-step
  apply vcg-step
  apply vcg-step
  apply vcg-step
  apply simp
  done

```

Here some Isar style version of the proof

```

lemma (in Fac-impl)
  shows  $\forall n. \Gamma \vdash \{N=n\} \ 'R ::= PROC\ Fac(N) \ \{R = fac\ n\}$ 
proof (hoare-rule HoarePartial.ProcRec1)
  have Fac-spec:  $\forall n. \Gamma, (\bigcup n. \{(\{N=n\}, Fac-'proc, \{R = fac\ n\}, \{\})\})$ 
     $\vdash \{N=n\} \ 'R ::= PROC\ Fac(N) \ \{R = fac\ n\}$ 
    apply (rule allI)
    apply (rule hoarep.Asm)
    by auto
  show  $\forall n. \Gamma, (\bigcup n. \{(\{N=n\}, Fac-'proc, \{R = fac\ n\}, \{\})\})$ 
     $\vdash \{N=n\} \ IF\ 'N = 0\ THEN\ 'R ::= 1$ 
     $ELSE\ 'R ::= CALL\ Fac('N - 1);; \ 'R ::= 'N * 'R\ FI\ \{R = fac\ n\}$ 
    apply vcg
    apply simp
  done
qed

```

To avoid retyping of potentially large pre and postconditions in the previous proof we can use the casual term abbreviations of the Isar language.

```

lemma (in Fac-impl)
  shows  $\forall n. \Gamma \vdash \{N=n\} \ 'R ::= PROC\ Fac(N) \ \{R = fac\ n\}$ 
  (is  $\forall n. \Gamma \vdash (?Pre\ n) \ ?Fac\ (?Post\ n)$ )
proof (hoare-rule HoarePartial.ProcRec1)
  have Fac-spec:  $\forall n. \Gamma, (\bigcup n. \{(?Pre\ n, Fac-'proc, ?Post\ n, \{\})\})$ 
     $\vdash (?Pre\ n) \ ?Fac\ (?Post\ n)$ 
    apply (rule allI)

```

```

apply (rule hoarep.Asm)
by auto
show  $\forall n. \Gamma, (\bigcup n. \{(?Pre\ n, Fac\text{-}proc, ?Post\ n, \{\})\})$ 
       $\vdash (?Pre\ n) \text{ IF } 'N = 0 \text{ THEN } 'R ::= 1$ 
       $ELSE\ 'R ::= CALL\ Fac('N - 1);; 'R ::= 'N * 'R \text{ FI } (?Post\ n)$ 
apply vcg
apply simp
done
qed

```

The previous proof pattern has still some kind of inconvenience. The augmented context is always printed in the proof state. That can mess up the state, especially if we have large specifications. This may be annoying if we want to develop single step or structured proofs. In this case it can be a good idea to introduce a new variable for the augmented context.

```

lemma (in Fac-impl) Fac-spec:
  shows  $\forall n. \Gamma \vdash \{ 'N = n \} 'R ::= PROC\ Fac('N) \{ 'R = fac\ n \}$ 
  (is  $\forall n. \Gamma \vdash (?Pre\ n) ?Fac\ (?Post\ n)$ )
proof (hoare-rule HoarePartial.ProcRec1)
  define  $\Theta'$  where  $\Theta' = (\bigcup n. \{(?Pre\ n, Fac\text{-}proc, ?Post\ n, \{\}::('a, 'b)\ \text{vars-scheme set})\})$ 
  have Fac-spec:  $\forall n. \Gamma, \Theta' \vdash (?Pre\ n) ?Fac\ (?Post\ n)$ 
  by (unfold  $\Theta'$ -def, rule allI, rule hoarep.Asm) auto

```

We have to name the fact *Fac-spec*, so that the vcg can use the specification for the recursive call, since it cannot infer it from the opaque  $\Theta'$ .

```

show  $\forall \sigma. \Gamma, \Theta' \vdash (?Pre\ \sigma) \text{ IF } 'N = 0 \text{ THEN } 'R ::= 1$ 
       $ELSE\ 'R ::= CALL\ Fac('N - 1);; 'R ::= 'N * 'R \text{ FI } (?Post\ \sigma)$ 
apply vcg
apply simp
done
qed

```

There are different rules available to prove procedure calls, depending on the kind of postcondition and whether or not the procedure is recursive or even mutually recursive. See for example *HoarePartial.ProcRec1*, *HoarePartial.ProcNoRec1*. They are all derived from the most general rule *HoarePartial.ProcRec*. All of them have some side-condition concerning definedness of the procedure. They can be solved in a uniform fashion. That's why we have created the method *hoare-rule*, which behaves like the method *rule* but automatically tries to solve the side-conditions.

## 17.6.2 Odd and Even

Odd and even are defined mutually recursive here. In the *procedures* command we conjoin both definitions with *and*.

```

procedures

```



```

odd(N | A) = IF 'N=0 THEN 'A:=0
            ELSE IF 'N=1 THEN CALL even ('N - 1, 'A)
            ELSE CALL odd ('N - 2, 'A)
            FI
            FI

```

**and**

```

even(N | A) = IF 'N=0 THEN 'A:=1
             ELSE IF 'N=1 THEN CALL odd ('N - 1, 'A)
             ELSE CALL even ('N - 2, 'A)
             FI
             FI

```

**print-theorems**

```

thm odd-body.odd-body-def
thm even-body.even-body-def
print-locale odd-even-clique

```

To prove the procedure calls to *odd* respectively *even* correct we first derive a rule to justify that we can assume both specifications to verify the bodies. This rule can be derived from the general *HoarePartial.ProcRec* rule. An ML function does this work:

```

ML <ML-Thms.bind-thm (ProcRec2, Hoare.gen-proc-rec @ {context} Hoare.Partial
2)>

```

**lemma** (in *odd-even-clique*)

```

shows odd-spec:  $\forall n. \Gamma \vdash \{ 'N=n \} 'A ::= PROC\ odd('N)$ 
              $\{ (\exists b. n = 2 * b + 'A) \wedge 'A < 2 \} \text{ (is } ?P1)$ 
and even-spec:  $\forall n. \Gamma \vdash \{ 'N=n \} 'A ::= PROC\ even('N)$ 
              $\{ (\exists b. n + 1 = 2 * b + 'A) \wedge 'A < 2 \} \text{ (is } ?P2)$ 

```

**proof** –

```

have ?P1  $\wedge$  ?P2
apply (hoare-rule ProcRec2)
apply vcg
apply clarsimp
apply (rule-tac x=b + 1 in exI)
apply arith
apply vcg
apply clarsimp
apply arith
done
thus ?P1 ?P2
by iprover+
qed

```

## 17.7 Expressions With Side Effects

$R := N++ + M++$

**lemma**  $\Gamma \vdash \{True\}$   
 $'N \gg n. 'N ::= 'N + 1 \gg$   
 $'M \gg m. 'M ::= 'M + 1 \gg$   
 $'R ::= n + m$   
 $\{ 'R = 'N + 'M - 2 \}$   
**apply** *vcg*  
**apply** *simp*  
**done**

$R := \text{Fac } (N) + \text{Fac } (M)$

**lemma** (**in** *Fac-impl*) **shows**  
 $\Gamma \vdash \{True\}$   
 $CALL \text{Fac}('N) \gg n. CALL \text{Fac}('M) \gg m.$   
 $'R ::= n + m$   
 $\{ 'R = \text{fac } 'N + \text{fac } 'M \}$   
**apply** *vcg*  
**done**

$R := (\text{Fac}(\text{Fac } (N)))$

**lemma** (**in** *Fac-impl*) **shows**  
 $\Gamma \vdash \{True\}$   
 $CALL \text{Fac}('N) \gg n. CALL \text{Fac}(n) \gg m.$   
 $'R ::= m$   
 $\{ 'R = \text{fac } (\text{fac } 'N) \}$   
**apply** *vcg*  
**done**

## 17.8 Global Variables and Heap

Now we define and verify some procedures on heap-lists. We consider list structures consisting of two fields, a content element *cont* and a reference to the next list element *next*. We model this by the following state space where every field has its own heap.

**record** *globals-list* =  
 $next-' :: ref \Rightarrow ref$   
 $cont-' :: ref \Rightarrow nat$

**record** *'g list-vars* = *'g state* +  
 $p-' :: ref$   
 $q-' :: ref$   
 $r-' :: ref$   
 $root-' :: ref$   
 $tmp-' :: ref$

Updates to global components inside a procedure will always be propagated to the caller. This is implicitly done by the parameter passing syntax translations. The record containing the global variables must begin with the prefix "globals".

We first define an append function on lists. It takes two references as parameters. It appends the list referred to by the first parameter with the list referred to by the second parameter, and returns the result right into the first parameter.

**procedures**

```

append(p,q|p) =
  IF 'p=NULL THEN 'p ::= 'q ELSE 'p → 'next ::= CALL append('p → 'next, 'q)
FI

```

**context** *append-impl*

**begin**

**declare**  $[[\text{hoare-use-call-tr}' = \text{false}]]$

**term** *CALL* *append*('p, 'q, 'p → 'next)

**declare**  $[[\text{hoare-use-call-tr}' = \text{true}]]$

**end**

Below we give two specifications this time. One captures the functional behaviour and focuses on the entities that are potentially modified by the procedure, the other one is a pure frame condition. The list in the modifies clause has to list all global state components that may be changed by the procedure. Note that we know from the modifies clause that the *cont* parts of the lists will not be changed. Also a small side note on the syntax. We use ordinary brackets in the postcondition of the modifies clause, and also the state components do not carry the acute, because we explicitly note the state *t* here.

The functional specification now introduces two logical variables besides the state space variable  $\sigma$ , namely *Ps* and *Qs*. They are universally quantified and range over both the pre and the postcondition, so that we are able to properly instantiate the specification during the proofs. The syntax  $\{\{\sigma. \dots\}\}$  is a shorthand to fix the current state:  $\{s. \sigma = s \dots\}$ .

**lemma** (in *append-impl*) *append-spec*:

**shows**  $\forall \sigma \ Ps \ Qs. \Gamma \vdash$

$\{\{\sigma. \text{List } 'p \ 'next \ Ps \wedge \text{List } 'q \ 'next \ Qs \wedge \text{set } Ps \cap \text{set } Qs = \{\}\}\}$

$'p ::= \text{PROC } \text{append}('p, 'q)$

$\{\{\text{List } 'p \ 'next \ (Ps @ Qs) \wedge (\forall x. x \notin \text{set } Ps \longrightarrow 'next \ x = {}^\sigma \text{next } x)\}\}$

**apply** (*hoare-rule* *HoarePartial.ProcRec1*)

**apply** *vcg*

**apply** *fastforce*

**done**

The modifies clause is equal to a proper record update specification of the following form.

```

lemma {t. t may-only-modify-globals Z in [next]}
  =
  {t.  $\exists$  next. globals t=next'-update ( $\lambda$ -. next) (globals Z)}
  apply (unfold mex-def meq-def)
  apply (simp)
  done

```

If the verification condition generator works on a procedure call it checks whether it can find a modified clause in the context. If one is present the procedure call is simplified before the Hoare rule *HoarePartial.ProcSpec* is applied. Simplification of the procedure call means, that the “copy back” of the global components is simplified. Only those components that occur in the modifies clause will actually be copied back. This simplification is justified by the rule *HoarePartial.ProcModifyReturn*. So after this simplification all global components that do not appear in the modifies clause will be treated as local variables.

You can study the effect of the modifies clause on the following two examples, where we want to prove that (@) does not change the *cont* part of the heap.

```

lemma (in append-impl)
  shows  $\Gamma \vdash \{ \prime p = \text{Null} \wedge \prime cont = c \} \prime p ::= \text{CALL } \text{append}(\prime p, \text{Null}) \{ \prime cont = c \}$ 
  apply vcg
  oops

```

To prove the frame condition, we have to tell the verification condition generator to use only the modifies clauses and not to search for functional specifications by the parameter *spec=modifies*. It will also try to solve the verification conditions automatically.

```

lemma (in append-impl) append-modifies:
  shows
   $\forall \sigma. \Gamma \vdash \{ \sigma \} \prime p ::= \text{PROC } \text{append}(\prime p, \prime q) \{ t. t \text{ may-only-modify-globals } \sigma \text{ in } [next] \}$ 
  apply (hoare-rule HoarePartial.ProcRec1)
  apply (vcg spec=modifies)
  done

```

```

lemma (in append-impl)
  shows  $\Gamma \vdash \{ \prime p = \text{Null} \wedge \prime cont = c \} \prime p \rightarrow \prime next ::= \text{CALL } \text{append}(\prime p, \text{Null}) \{ \prime cont = c \}$ 
  apply vcg
  apply simp
  done

```

Of course we could add the modifies clause to the functional specification as well. But separating both has the advantage that we split up the verification

work. We can make use of the modifies clause before we apply the functional specification in a fully automatic fashion.

To verify the body of (@) we do not need the modifies clause, since the specification does not talk about *cont* at all, and we don't access *cont* inside the body. This may be different for more complex procedures.

To prove that a procedure respects the modifies clause, we only need the modifies clauses of the procedures called in the body. We do not need the functional specifications. So we can always prove the modifies clause without functional specifications, but we may need the modifies clause to prove the functional specifications.

### 17.8.1 Insertion Sort

**primrec** *sorted*:: ('a ⇒ 'a ⇒ bool) ⇒ 'a list ⇒ bool

**where**

*sorted le* [] = True |

*sorted le* (x#xs) = ((∀ y∈set xs. le x y) ∧ *sorted le* xs)

**procedures**

*insert*(*r*,*p* | *p*) =

IF *r*=Null THEN SKIP

ELSE IF *p*=Null THEN '*p* ::= '*r*;; '*p*→'*next* ::= Null

ELSE IF '*r*→'*cont* ≤ '*p*→'*cont*

THEN '*r*→'*next* ::= '*p*;; '*p*::=''*r*

ELSE '*p*→'*next* ::= CALL *insert*(''*r*, '*p*→'*next*)

FI

FI

FI

In the postcondition of the functional specification there is a small but important subtlety. Whenever we talk about the *cont* part we refer to the one of the pre-state, even in the conclusion of the implication. The reason is, that we have separated out, that *cont* is not modified by the procedure, to the modifies clause. So whenever we talk about unmodified parts in the postcondition we have to use the pre-state part, or explicitly state an equality in the postcondition. The reason is simple. If the postcondition would talk about '*cont* instead of  $\sigma cont$ , we get a new instance of *cont* during verification and the postcondition would only state something about this new instance. But as the verification condition generator uses the modifies clause the caller of *insert* instead still has the old *cont* after the call. That's the very reason for the modifies clause. So the caller and the specification will simply talk about two different things, without being able to relate them (unless an explicit equality is added to the specification).

**lemma** (*in insert-impl*) *insert-modifies*:  
 $\forall \sigma. \Gamma \vdash \{\sigma\} \text{'p} ::= \text{PROC insert}(\text{'r}, \text{'p}) \{t. t \text{ may-only-modify-globals } \sigma \text{ in } [\text{next}]\}$   
**apply** (*hoare-rule HoarePartial.ProcRec1*)  
**apply** (*vcg spec=modifies*)  
**done**

**lemma** (*in insert-impl*) *insert-spec*:  
 $\forall \sigma \text{ Ps} . \Gamma \vdash \{\sigma. \text{List 'p 'next Ps} \wedge \text{sorted} (\leq) (\text{map 'cont Ps}) \wedge$   
 $\text{'r} \neq \text{Null} \wedge \text{'r} \notin \text{set Ps}\}$   
 $\text{'p} ::= \text{PROC insert}(\text{'r}, \text{'p})$   
 $\{\exists \text{Qs}. \text{List 'p 'next Qs} \wedge \text{sorted} (\leq) (\text{map}^{\sigma \text{cont}} \text{Qs}) \wedge$   
 $\text{set Qs} = \text{insert}^{\sigma \text{r}} (\text{set Ps}) \wedge$   
 $(\forall x. x \notin \text{set Qs} \longrightarrow \text{'next } x = \sigma_{\text{next}} x)\}$

**apply** (*hoare-rule HoarePartial.ProcRec1*)  
**apply** *vcg*  
**apply** (*intro conjI impI*)  
**apply** *fastforce*  
**apply** *fastforce*  
**apply** *fastforce*  
**apply** (*clarsimp*)  
**apply** *force*  
**done**

**procedures**  
 $\text{insertSort}(p \mid p) =$   
 $\text{'r} ::= \text{Null};;$   
 $\text{WHILE } (\text{'p} \neq \text{Null}) \text{ DO}$   
 $\text{'q} ::= \text{'p};;$   
 $\text{'p} ::= \text{'p} \rightarrow \text{'next};;$   
 $\text{'r} ::= \text{CALL insert}(\text{'q}, \text{'r})$   
 $\text{OD};;$   
 $\text{'p} ::= \text{'r}$

**lemma** (*in insertSort-impl*) *insertSort-modifies*:  
**shows**  
 $\forall \sigma. \Gamma \vdash \{\sigma\} \text{'p} ::= \text{PROC insertSort}(\text{'p})$   
 $\{t. t \text{ may-only-modify-globals } \sigma \text{ in } [\text{next}]\}$   
**apply** (*hoare-rule HoarePartial.ProcRec1*)  
**apply** (*vcg spec=modifies*)  
**done**

Insertion sort is not implemented recursively here but with a while loop. Note that the while loop is not annotated with an invariant in the procedure definition. The invariant only comes into play during verification. There-

fore we will annotate the body during the proof with the rule *HoarePartial.annotateI*.

```

lemma (in insertSort-impl) insertSort-body-spec:
  shows  $\forall \sigma \ Ps. \Gamma, \Theta \vdash \{ \sigma. \text{List } 'p \ 'next \ Ps \}$ 
     $'p ::= PROC \text{insertSort}('p)$ 
     $\{ \exists Qs. \text{List } 'p \ 'next \ Qs \wedge \text{sorted } (\leq) (\text{map } \sigma_{cont} \ Qs) \wedge$ 
       $\text{set } Qs = \text{set } Ps \}$ 
  apply (hoare-rule HoarePartial.ProcRec1)
  apply (hoare-rule anno=
     $'r ::= Null;$ 
    WHILE  $'p \neq Null$ 
    INV  $\{ \exists Rs. \text{List } 'p \ 'next \ Qs \wedge \text{List } 'r \ 'next \ Rs \wedge$ 
       $\text{set } Qs \cap \text{set } Rs = \{ \} \wedge$ 
       $\text{sorted } (\leq) (\text{map } 'cont \ Rs) \wedge \text{set } Qs \cup \text{set } Rs = \text{set } Ps \wedge$ 
       $'cont = \sigma_{cont} \}$ 
    DO  $'q ::= 'p;; 'p ::= 'p \rightarrow 'next;; 'r ::= CALL \text{insert}('q, 'r) \ OD;;$ 
     $'p ::= 'r \text{ in } HoarePartial.annotateI$ )
  apply vcg
  apply fastforce
  prefer 2
  apply fastforce
  apply (clarsimp)
  apply (rule-tac x=ps in exI)
  apply (intro conjI)
  apply (rule heap-eq-ListI1)
  apply assumption
  apply clarsimp
  apply (subgoal-tac x≠p ∧ x ∉ set Rs)
  apply auto
  done

```

## 17.8.2 Memory Allocation and Deallocation

The basic idea of memory management is to keep a list of allocated references in the state space. Allocation of a new reference adds a new reference to the list deallocation removes a reference. Moreover we keep a counter "free" for the free memory.

```

record globals-list-alloc = globals-list +
  alloc-'::ref list
  free-'::nat

```

```

record 'g list-vars' = 'g list-vars +
  i-'::nat
  first-'::ref

```

```

definition sz = (2::nat)

```

Restrict locale *hoare* to the required type.

**locale** *hoare-ex* =

*hoare*  $\Gamma$  **for**  $\Gamma :: 'c \rightarrow (( 'a \text{ globals-list-alloc-scheme, } 'b) \text{ list-vars'-scheme, } 'c, 'd)$   
*com*

**lemma** (**in** *hoare-ex*)

$\Gamma \vdash \{ \{ 'i = 0 \wedge 'first = Null \wedge n * sz \leq 'free \}$   
  **WHILE**  $'i < n$   
  **INV**  $\{ \exists Ps. List 'first 'next Ps \wedge length Ps = 'i \wedge 'i \leq n \wedge$   
     $set Ps \subseteq set 'alloc \wedge (n - 'i) * sz \leq 'free \}$   
  **DO**  
     $'p ::= NEW sz [ 'cont ::= 0, 'next ::= Null ] ;;$   
     $'p \rightarrow 'next ::= 'first ;;$   
     $'first ::= 'p ;;$   
     $'i ::= 'i + 1$   
  **OD**  
   $\{ \exists Ps. List 'first 'next Ps \wedge length Ps = n \wedge set Ps \subseteq set 'alloc \}$

**apply** (*vcg*)

**apply** *simp*

**apply** *clarsimp*

**apply** (*rule conjI*)

**apply** *clarsimp*

**apply** (*rule-tac x=new (set alloc)#Ps in exI*)

**apply** *clarsimp*

**apply** (*rule conjI*)

**apply** *fastforce*

**apply** (*simp add: sz-def*)

**apply** (*simp add: sz-def*)

**apply** *fastforce*

**done**

**lemma** (**in** *hoare-ex*)

$\Gamma \vdash \{ \{ 'i = 0 \wedge 'first = Null \wedge n * sz \leq 'free \}$   
  **WHILE**  $'i < n$   
  **INV**  $\{ \exists Ps. List 'first 'next Ps \wedge length Ps = 'i \wedge 'i \leq n \wedge$   
     $set Ps \subseteq set 'alloc \wedge (n - 'i) * sz \leq 'free \}$   
  **DO**  
     $'p ::= NNEW sz [ 'cont ::= 0, 'next ::= Null ] ;;$   
     $'p \rightarrow 'next ::= 'first ;;$   
     $'first ::= 'p ;;$   
     $'i ::= 'i + 1$   
  **OD**  
   $\{ \exists Ps. List 'first 'next Ps \wedge length Ps = n \wedge set Ps \subseteq set 'alloc \}$

**apply** (*vcg*)

**apply** *simp*

**apply** *clarsimp*



```

apply (rule conjI)
apply clarsimp
apply (rule-tac x=new (set alloc)#Ps in exI)
apply clarsimp
apply (rule conjI)
apply fastforce
apply (simp add: sz-def)
apply (simp add: sz-def)
apply fastforce
done

```

## 17.9 Fault Avoiding Semantics

If we want to ensure that no runtime errors occur we can insert guards into the code. We will not be able to prove any nontrivial Hoare triple about code with guards, if we cannot show that the guards will never fail. A trivial hoare triple is one with an empty precondition.

```

lemma  $\Gamma \vdash \{\{True\}\} \{\{p \neq Null\}\} \longrightarrow 'p \rightarrow 'next ::= 'p \{\{True\}\}$ 
apply vcg
oops

```

```

lemma  $\Gamma \vdash \{\} \{\{p \neq Null\}\} \longrightarrow 'p \rightarrow 'next ::= 'p \{\{True\}\}$ 
apply vcg
done

```

Let us consider this small program that reverts a list. At first without guards.

```

lemma (in hoare-ex) rev-strip:
 $\Gamma \vdash \{\{List 'p 'next Ps \wedge List 'q 'next Qs \wedge set Ps \cap set Qs = \{\}\} \wedge$ 
 $set Ps \subseteq set 'alloc \wedge set Qs \subseteq set 'alloc\}$ 
  WHILE  $'p \neq Null$ 
  INV  $\{\{\exists ps qs. List 'p 'next ps \wedge List 'q 'next qs \wedge set ps \cap set qs = \{\} \wedge$ 
 $rev ps @ qs = rev Ps @ Qs \wedge$ 
 $set ps \subseteq set 'alloc \wedge set qs \subseteq set 'alloc\}$ 
  DO  $'r ::= 'p;$ 
 $'p ::= 'p \rightarrow 'next;;$ 
 $'r \rightarrow 'next ::= 'q;;$ 
 $'q ::= 'r$  OD
 $\{\{List 'q 'next (rev Ps @ Qs) \wedge set Ps \subseteq set 'alloc \wedge set Qs \subseteq set 'alloc\}$ 
apply (vcg)
apply fastforce+
done

```

If we want to ensure that we do not dereference *Null* or access unallocated memory, we have to add some guards.

```

locale hoare-ex-guard =
  hoare  $\Gamma$  for  $\Gamma :: 'c \rightarrow (( 'a$  globals-list-alloc-scheme, 'b) list-vars'-scheme, 'c, bool)
  com

```

**lemma**

(in *hoare-ex-guard*)  
 $\Gamma \vdash \{ \text{List } 'p \text{ 'next } Ps \wedge \text{List } 'q \text{ 'next } Qs \wedge \text{set } Ps \cap \text{set } Qs = \{ \} \wedge$   
 $\text{set } Ps \subseteq \text{set } 'alloc \wedge \text{set } Qs \subseteq \text{set } 'alloc \}$   
**WHILE**  $'p \neq \text{Null}$   
**INV**  $\{ \exists ps \ qs. \text{List } 'p \text{ 'next } ps \wedge \text{List } 'q \text{ 'next } qs \wedge \text{set } ps \cap \text{set } qs = \{ \} \wedge$   
 $\text{rev } ps @ qs = \text{rev } Ps @ Qs \wedge$   
 $\text{set } ps \subseteq \text{set } 'alloc \wedge \text{set } qs \subseteq \text{set } 'alloc \}$   
**DO**  $'r ::= 'p;$   
 $\{ 'p \neq \text{Null} \wedge 'p \in \text{set } 'alloc \} \mapsto 'p ::= 'p \rightarrow 'next;;$   
 $\{ 'r \neq \text{Null} \wedge 'r \in \text{set } 'alloc \} \mapsto 'r \rightarrow 'next ::= 'q;;$   
 $'q ::= 'r$  **OD**  
 $\{ \text{List } 'q \text{ 'next } (\text{rev } Ps @ Qs) \wedge \text{set } Ps \subseteq \text{set } 'alloc \wedge \text{set } Qs \subseteq \text{set } 'alloc \}$   
**apply** (*vcg*)  
**apply** *fastforce+*  
**done**

We can also just prove that no faults will occur, by giving the trivial post-condition.

**lemma** (in *hoare-ex-guard*) *rev-noFault*:

$\Gamma \vdash \{ \text{List } 'p \text{ 'next } Ps \wedge \text{List } 'q \text{ 'next } Qs \wedge \text{set } Ps \cap \text{set } Qs = \{ \} \wedge$   
 $\text{set } Ps \subseteq \text{set } 'alloc \wedge \text{set } Qs \subseteq \text{set } 'alloc \}$   
**WHILE**  $'p \neq \text{Null}$   
**INV**  $\{ \exists ps \ qs. \text{List } 'p \text{ 'next } ps \wedge \text{List } 'q \text{ 'next } qs \wedge \text{set } ps \cap \text{set } qs = \{ \} \wedge$   
 $\text{rev } ps @ qs = \text{rev } Ps @ Qs \wedge$   
 $\text{set } ps \subseteq \text{set } 'alloc \wedge \text{set } qs \subseteq \text{set } 'alloc \}$   
**DO**  $'r ::= 'p;$   
 $\{ 'p \neq \text{Null} \wedge 'p \in \text{set } 'alloc \} \mapsto 'p ::= 'p \rightarrow 'next;;$   
 $\{ 'r \neq \text{Null} \wedge 'r \in \text{set } 'alloc \} \mapsto 'r \rightarrow 'next ::= 'q;;$   
 $'q ::= 'r$  **OD**  
**UNIV, UNIV**  
**apply** (*vcg*)  
**apply** *fastforce+*  
**done**

**lemma** (in *hoare-ex-guard*) *rev-moduloGuards*:

$\Gamma \vdash / \{ \text{True} \} \{ \text{List } 'p \text{ 'next } Ps \wedge \text{List } 'q \text{ 'next } Qs \wedge \text{set } Ps \cap \text{set } Qs = \{ \} \wedge$   
 $\text{set } Ps \subseteq \text{set } 'alloc \wedge \text{set } Qs \subseteq \text{set } 'alloc \}$   
**WHILE**  $'p \neq \text{Null}$   
**INV**  $\{ \exists ps \ qs. \text{List } 'p \text{ 'next } ps \wedge \text{List } 'q \text{ 'next } qs \wedge \text{set } ps \cap \text{set } qs = \{ \} \wedge$   
 $\text{rev } ps @ qs = \text{rev } Ps @ Qs \wedge$   
 $\text{set } ps \subseteq \text{set } 'alloc \wedge \text{set } qs \subseteq \text{set } 'alloc \}$   
**DO**  $'r ::= 'p;$   
 $\{ 'p \neq \text{Null} \wedge 'p \in \text{set } 'alloc \} \checkmark \mapsto 'p ::= 'p \rightarrow 'next;;$   
 $\{ 'r \neq \text{Null} \wedge 'r \in \text{set } 'alloc \} \checkmark \mapsto 'r \rightarrow 'next ::= 'q;;$   
 $'q ::= 'r$  **OD**  
 $\{ \text{List } 'q \text{ 'next } (\text{rev } Ps @ Qs) \wedge \text{set } Ps \subseteq \text{set } 'alloc \wedge \text{set } Qs \subseteq \text{set } 'alloc \}$   
**apply** *vcg*

apply fastforce+  
done

**lemma** *CombineStrip'*:  
 assumes *deriv*:  $\Gamma, \Theta \vdash /_F P \ c' \ Q, A$   
 assumes *deriv-strip*:  $\Gamma, \Theta \vdash /_{\{\}} P \ c'' \ UNIV, UNIV$   
 assumes *c''*:  $c'' = \text{mark-guards False (strip-guards } (-F) \ c')$   
 assumes *c*:  $c = \text{mark-guards False } c'$   
 shows  $\Gamma, \Theta \vdash /_{\{\}} P \ c \ Q, A$   
**proof** –  
 from *deriv-strip* [*simplified c''*]  
 have  $\Gamma, \Theta \vdash P \ (\text{strip-guards } (-F) \ c') \ UNIV, UNIV$   
 by (rule *HoarePartialProps.MarkGuardsD*)  
 with *deriv*  
 have  $\Gamma, \Theta \vdash P \ c' \ Q, A$   
 by (rule *HoarePartialProps.CombineStrip*)  
 hence  $\Gamma, \Theta \vdash P \ \text{mark-guards False } c' \ Q, A$   
 by (rule *HoarePartialProps.MarkGuardsI*)  
 thus *thesis*  
 by (*simp add: c*)  
**qed**

We can then combine the prove that no fault will occur with the functional proof of the programme without guards to get the full prove by the rule  
 $\llbracket ?\Gamma, ?\Theta \vdash /_{?F} ?P \ ?c \ ?Q, ?A; ?\Gamma, ?\Theta \vdash ?P \ \text{strip-guards } (- ?F) \ ?c \ UNIV, UNIV \rrbracket$   
 $\implies ?\Gamma, ?\Theta \vdash ?P \ ?c \ ?Q, ?A$

**lemma**  
 (*in hoare-ex-guard*)  
 $\Gamma \vdash \{ \text{List } 'p \ 'next \ Ps \wedge \text{List } 'q \ 'next \ Qs \wedge \text{set } Ps \cap \text{set } Qs = \{\} \wedge$   
 $\text{set } Ps \subseteq \text{set } 'alloc \wedge \text{set } Qs \subseteq \text{set } 'alloc \}$   
 WHILE  $'p \neq \text{Null}$   
 INV  $\{ \exists ps \ qs. \text{List } 'p \ 'next \ ps \wedge \text{List } 'q \ 'next \ qs \wedge \text{set } ps \cap \text{set } qs = \{\} \wedge$   
 $\text{rev } ps \ @ \ qs = \text{rev } Ps \ @ \ Qs \wedge$   
 $\text{set } ps \subseteq \text{set } 'alloc \wedge \text{set } qs \subseteq \text{set } 'alloc \}$   
 DO  $'r ::= 'p;$   
 $\{ 'p \neq \text{Null} \wedge 'p \in \text{set } 'alloc \} \mapsto 'p ::= 'p \rightarrow 'next;$   
 $\{ 'r \neq \text{Null} \wedge 'r \in \text{set } 'alloc \} \mapsto 'r \rightarrow 'next ::= 'q;$   
 $'q ::= 'r \ \text{OD}$   
 $\{ \text{List } 'q \ 'next \ (\text{rev } Ps \ @ \ Qs) \wedge \text{set } Ps \subseteq \text{set } 'alloc \wedge \text{set } Qs \subseteq \text{set } 'alloc \}$

apply (rule *CombineStrip'* [*OF rev-moduloGuards rev-noFault*])  
 apply *simp*  
 apply *simp*  
 done

In the previous example the effort to split up the prove did not really pay

off. But when we think of programs with a lot of guards and complicated specifications it may be better to first focus on a prove without the messy guards. Maybe it is possible to automate the no fault proofs so that it suffices to focus on the stripped program.

The purpose of guards is to watch for faults that can occur during evaluation of expressions. In the example before we watched for null pointer dereferencing or memory faults. We can also look for array index bounds or division by zero. As the condition of a while loop is evaluated in each iteration we cannot just add a guard before the while loop. Instead we need a special guard for the condition. Example:  $WHILE (False, \{ \{ p \neq Null \} \}) \mapsto p \rightarrow next \neq Null DO SKIP OD$

## 17.10 Circular Lists

### definition

$distPath :: ref \Rightarrow (ref \Rightarrow ref) \Rightarrow ref \Rightarrow ref\ list \Rightarrow bool$  **where**  
 $distPath\ x\ next\ y\ as = (Path\ x\ next\ y\ as \wedge\ distinct\ as)$

**lemma** *neq-dP*:  $\llbracket p \neq q; Path\ p\ h\ q\ Ps; distinct\ Ps \rrbracket \implies$   
 $\exists\ Qs. p \neq Null \wedge Ps = p \# Qs \wedge p \notin set\ Qs$   
**by** (*cases Ps, auto*)

### lemma

*circular-list-rev-I*:  
 $\Gamma \vdash \{ \{ root = r \wedge distPath\ 'root\ 'next\ 'root\ (r \# Ps) \} \}$   
 $\{ p := 'root;; 'q := 'root \rightarrow 'next;;$   
 $WHILE\ 'q \neq 'root$   
 $INV\ \{ \{ \exists\ ps\ qs. distPath\ 'p\ 'next\ 'root\ ps \wedge distPath\ 'q\ 'next\ 'root\ qs \wedge$   
 $'root = r \wedge r \neq Null \wedge r \notin set\ Ps \wedge set\ ps \cap set\ qs = \{ \} \wedge$   
 $Ps = (rev\ ps) @ qs \} \}$   
 $DO\ 'tmp := 'q;; 'q := 'q \rightarrow 'next;; 'tmp \rightarrow 'next := 'p;; 'p := 'tmp\ OD;;$   
 $'root \rightarrow 'next := 'p$   
 $\{ 'root = r \wedge distPath\ 'root\ 'next\ 'root\ (r \# rev\ Ps) \} \}$

**apply** (*simp only: distPath-def*)  
**apply** *vcg*  
**apply** (*rule-tac x=[] in exI*)  
**apply** *fastforce*  
**apply** *clarsimp*  
**apply** (*drule (2) neq-dP*)  
**apply** (*rule-tac x=q # ps in exI*)  
**apply** *clarsimp*  
**apply** *fastforce*  
**done**

**lemma** *path-is-list*:  $\wedge a\ next\ b. \llbracket Path\ b\ next\ a\ Ps; a \notin set\ Ps; a \neq Null \rrbracket$   
 $\implies List\ b\ (next(a := Null))\ (Ps @ [a])$

```

apply (induct Ps)
apply (auto simp add:fun-upd-apply)
done

```

The simple algorithm for acyclic list reversal, with modified annotations, works for cyclic lists as well.:

**lemma** *circular-list-rev-II*:

```

 $\Gamma \vdash$ 
 $\{ \{ p = r \wedge \text{distPath } 'p \ 'next \ 'p \ (r \# Ps) \} \}$ 
 $'q := \text{Null};$ 
WHILE  $'p \neq \text{Null}$ 
INV
 $\{ (( 'q = \text{Null} ) \longrightarrow (\exists ps. \text{distPath } 'p \ 'next \ r \ ps \ \wedge \ ps = r \# Ps)) \wedge$ 
 $(( 'q \neq \text{Null} ) \longrightarrow (\exists ps \ qs. \text{distPath } 'q \ 'next \ r \ qs \ \wedge \ \text{List } 'p \ 'next \ ps \ \wedge$ 
 $\text{set } ps \cap \text{set } qs = \{ \} \wedge \text{rev } qs \ @ \ ps = Ps@[r])) \wedge$ 
 $\neg ( 'p = \text{Null} \wedge 'q = \text{Null} \wedge r = \text{Null} )$ 
 $\}$ 
DO
 $'tmp := 'p;; 'p := 'p \rightarrow 'next;; 'tmp \rightarrow 'next := 'q;; 'q := 'tmp$ 
OD
 $\{ 'q = r \wedge \text{distPath } 'q \ 'next \ 'q \ (r \# \text{rev } Ps) \}$ 

```

```

apply (simp only:distPath-def)
apply vcg
apply clarsimp
apply clarsimp
apply (case-tac (q = Null))
apply (fastforce intro: path-is-list)
apply clarify
apply (rule-tac x=psa in exI)
apply (rule-tac x= p # qs in exI)
apply force
apply fastforce
done

```

Although the above algorithm is more succinct, its invariant looks more involved. The reason for the case distinction on  $q$  is due to the fact that during execution, the pointer variables can point to either cyclic or acyclic structures.

When working on lists, its sometimes better to remove *fun-upd-apply* from the simpset, and instead include *fun-upd-same* and *fun-upd-other* to the simpset

```

lemma  $\Gamma \vdash \{ \sigma \}$ 
 $'I := 'M;$ 
ANNO  $\tau. \{ \tau. 'I = {}^\sigma M \}$ 
 $'M := 'N;; 'N := 'I$ 
 $\{ 'M = {}^\tau N \wedge 'N = {}^\tau I \}$ 

```

```

       $\{\! \{ 'M = \sigma N \wedge 'N = \sigma M \} \!\}$ 
apply vcg
apply auto
done

lemma  $\Gamma \vdash (\{\sigma\} \cap \{\! \{ 'M = 0 \wedge 'S = 0 \} \!\})$ 
  (ANNO  $\tau. (\{\tau\} \cap \{\! \{ 'A = \sigma A \wedge 'I = \sigma I \wedge 'M = 0 \wedge 'S = 0 \} \!\})$ 
  WHILE  $'M \neq 'A$ 
  INV  $\{\! \{ 'S = 'M * 'I \wedge 'A = \tau A \wedge 'I = \tau I \} \!\}$ 
  DO  $'S ::= 'S + 'I;; 'M ::= 'M + 1$  OD
   $\{\! \{ 'S = \tau A * \tau I \} \!\}$ 
   $\{\! \{ 'S = \sigma A * \sigma I \} \!\}$ 
apply vcg-step
apply vcg-step
apply simp
apply vcg-step
apply vcg-step
apply simp
apply vcg
apply simp
apply simp
apply vcg-step
apply auto
done

```

Instead of annotations one can also directly use previously proven lemmas.

```

lemma foo-lemma:  $\forall n m. \Gamma \vdash \{\! \{ 'N = n \wedge 'M = m \} \!\} 'N ::= 'N + 1;; 'M ::= 'M + 1$ 
   $M + 1$ 
       $\{\! \{ 'N = n + 1 \wedge 'M = m + 1 \} \!\}$ 
by vcg

```

```

lemma  $\Gamma \vdash \{\! \{ 'N = n \wedge 'M = m \} \!\}$  LEMMA foo-lemma
   $'N ::= 'N + 1;; 'M ::= 'M + 1$ 
  END;;
   $'N ::= 'N + 1$ 
   $\{\! \{ 'N = n + 2 \wedge 'M = m + 1 \} \!\}$ 
apply vcg
apply simp
done

```

```

lemma  $\Gamma \vdash \{\! \{ 'N = n \wedge 'M = m \} \!\}$ 
  LEMMA foo-lemma
   $'N ::= 'N + 1;; 'M ::= 'M + 1$ 
  END;;
  LEMMA foo-lemma
   $'N ::= 'N + 1;; 'M ::= 'M + 1$ 
  END

```

```

    { 'N = n + 2 ∧ 'M = m + 2 }
apply vcg
apply simp
done

lemma  $\Gamma \vdash \{ 'N = n \wedge 'M = m \}$ 
    'N ::= 'N + 1;; 'M ::= 'M + 1;;
    'N ::= 'N + 1;; 'M ::= 'M + 1
    { 'N = n + 2 ∧ 'M = m + 2 }
apply (hoare-rule anno=
    LEMMA foo-lemma
    'N ::= 'N + 1;; 'M ::= 'M + 1
    END;;
    LEMMA foo-lemma
    'N ::= 'N + 1;; 'M ::= 'M + 1
    END
    in HoarePartial.annotate-normI)
apply vcg
apply simp
done

Just some test on marked, guards

lemma  $\Gamma \vdash \{ True \}$  WHILE {P 'N }√, {Q 'M }#, {R 'N } $\mapsto$  'N < 'M
    INV { 'N < 2 } DO
    'N ::= 'M
    OD
    {hard}
apply vcg
oops

lemma  $\Gamma \vdash / \{ True \}$  {True} WHILE {P 'N }√, {Q 'M }#, {R 'N } $\mapsto$  'N < 'M
    INV { 'N < 2 } DO
    'N ::= 'M
    OD
    {hard}
apply vcg
oops

term  $\Gamma \vdash / \{ True \}$  {True} WHILEg 'N < 'Arr!i
    FIX Z.
    INV { 'N < 2 }

    DO
    'N ::= 'M
    OD
    {hard}

```

```

lemma  $\Gamma \vdash \{True\} \{True\} \text{WHILE}_g \ 'N < 'Arr!i$ 
      FIX Z.
      INV  $\{ 'N < 2 \}$ 
      VAR arbitrary
      DO
      'N ::= 'M
      OD
      {hard}
apply vcg
oops

lemma  $\Gamma \vdash \{True\} \{True\} \text{WHILE} \{P \ 'N \} \surd, \{Q \ 'M \} \#, \{R \ 'N \} \mapsto \ 'N < \ 'M$ 
      FIX Z.
      INV  $\{ 'N < 2 \}$ 
      VAR arbitrary
      DO
      'N ::= 'M
      OD
      {hard}
apply vcg
oops

end

```

## 18 Examples using Statespaces

```

theory VcgExSP imports ../HeapList ../Vcg begin

```

### 18.1 State Spaces

First of all we provide a store of program variables that occur in the programs considered later. Slightly unexpected things may happen when attempting to work with undeclared variables.

```

hoarestate state-space =

```

```

  A :: nat
  I :: nat
  M :: nat
  N :: nat
  R :: nat
  S :: nat
  B :: bool
  Abr :: string

```

```

lemma (in state-space)  $\Gamma \vdash \{ 'N = n \} \text{LOC } 'N ::= 10;; 'N ::= 'N + 2 \text{COL } \{ 'N = n \}$ 
by vcg

```

Internally we decorate the state components in the statespace with the suffix



-', to avoid cluttering the namespace with the simple names that could no longer be used for logical variables otherwise.

We will first consider programs without procedures, later on we will regard procedures without global variables and finally we will get the full pictures: mutually recursive procedures with global variables (including heap).

## 18.2 Basic Examples

We look at few trivialities involving assignment and sequential composition, in order to get an idea of how to work with our formulation of Hoare Logic.

Using the basic rule directly is a bit cumbersome.

```
lemma (in state-space)  $\Gamma \vdash \{|N = 5|\} \text{'N} ::= 2 * \text{'N} \{|N = 10|\}$ 
apply (rule HoarePartial.Basic)
apply simp
done
```

```
lemma (in state-space)  $\Gamma \vdash \{\text{True}\} \text{'N} ::= 10 \{\text{'N} = 10\}$ 
by vcg
```

```
lemma (in state-space)  $\Gamma \vdash \{2 * \text{'N} = 10\} \text{'N} ::= 2 * \text{'N} \{\text{'N} = 10\}$ 
by vcg
```

```
lemma (in state-space)  $\Gamma \vdash \{\text{'N} = 5\} \text{'N} ::= 2 * \text{'N} \{\text{'N} = 10\}$ 
apply vcg
apply simp
done
```

```
lemma (in state-space)  $\Gamma \vdash \{\text{'N} + 1 = a + 1\} \text{'N} ::= \text{'N} + 1 \{\text{'N} = a + 1\}$ 
by vcg
```

```
lemma (in state-space)  $\Gamma \vdash \{\text{'N} = a\} \text{'N} ::= \text{'N} + 1 \{\text{'N} = a + 1\}$ 
apply vcg
apply simp
done
```

```
lemma (in state-space)
shows  $\Gamma \vdash \{a = a \wedge b = b\} \text{'M} ::= a;; \text{'N} ::= b \{\text{'M} = a \wedge \text{'N} = b\}$ 
by vcg
```

```
lemma (in state-space)
shows  $\Gamma \vdash \{\text{True}\} \text{'M} ::= a;; \text{'N} ::= b \{\text{'M} = a \wedge \text{'N} = b\}$ 
by vcg
```

```
lemma (in state-space)
shows  $\Gamma \vdash \{\text{'M} = a \wedge \text{'N} = b\}$ 
```

```

      'I ::= 'M;; 'M ::= 'N;; 'N ::= 'I
    { 'M = b ∧ 'N = a }
apply vcg
apply simp
done

```

We can also perform verification conditions generation step by step by using the *vcg-step* method.

```

lemma (in state-space)
  shows  $\Gamma \vdash \{ 'M = a \wedge 'N = b \}$ 
    'I ::= 'M;; 'M ::= 'N;; 'N ::= 'I
    { 'M = b ∧ 'N = a }
  apply vcg-step
  apply vcg-step
  apply vcg-step
  apply vcg-step
  apply simp
done

```

In the following assignments we make use of the consequence rule in order to achieve the intended precondition. Certainly, the *vcg* method is able to handle this case, too.

```

lemma (in state-space)
  shows  $\Gamma \vdash \{ 'M = 'N \} 'M ::= 'M + 1 \{ 'M \neq 'N \}$ 
proof –
  have  $\{ 'M = 'N \} \subseteq \{ 'M + 1 \neq 'N \}$ 
    by auto
  also have  $\Gamma \vdash \dots 'M ::= 'M + 1 \{ 'M \neq 'N \}$ 
    by vcg
  finally show ?thesis .
qed

```

```

lemma (in state-space)
  shows  $\Gamma \vdash \{ 'M = 'N \} 'M ::= 'M + 1 \{ 'M \neq 'N \}$ 
proof –
  have  $\bigwedge m n :: nat. m = n \longrightarrow m + 1 \neq n$ 
    – inclusion of assertions expressed in “pure” logic,
    – without mentioning the state space
  by simp
  also have  $\Gamma \vdash \{ 'M + 1 \neq 'N \} 'M ::= 'M + 1 \{ 'M \neq 'N \}$ 
    by vcg
  finally show ?thesis .
qed

```

```

lemma (in state-space)
  shows  $\Gamma \vdash \{ 'M = 'N \} 'M ::= 'M + 1 \{ 'M \neq 'N \}$ 
  apply vcg
  apply simp
done

```

### 18.3 Multiplication by Addition

We now do some basic examples of actual WHILE programs. This one is a loop for calculating the product of two natural numbers, by iterated addition. We first give detailed structured proof based on single-step Hoare rules.

**lemma** (in *state-space*)  
**shows**  $\Gamma \vdash \{\! \{ 'M = 0 \wedge 'S = 0 \} \}$   
     *WHILE*  $'M \neq a$   
     *DO*  $'S ::= 'S + b;; 'M ::= 'M + 1$  *OD*  
      $\{\! \{ 'S = a * b \} \}$   
**proof** –  
 let  $\Gamma \vdash - ?while - = ?thesis$   
 let  $\{\! \{ ?inv \} \} = \{\! \{ 'S = 'M * b \} \}$   
  
 have  $\{\! \{ 'M = 0 \ \& \ 'S = 0 \} \} \subseteq \{\! \{ ?inv \} \}$  **by** *auto*  
 also have  $\Gamma \vdash \dots ?while \{\! \{ ?inv \wedge \neg ('M \neq a) \} \}$   
**proof**  
 let  $?c = 'S ::= 'S + b;; 'M ::= 'M + 1$   
 have  $\{\! \{ ?inv \wedge 'M \neq a \} \} \subseteq \{\! \{ 'S + b = ('M + 1) * b \} \}$   
     **by** *auto*  
 also have  $\Gamma \vdash \dots ?c \{\! \{ ?inv \} \}$  **by** *vcg*  
 finally show  $\Gamma \vdash \{\! \{ ?inv \wedge 'M \neq a \} \} ?c \{\! \{ ?inv \} \} .$   
**qed**  
 also have  $\{\! \{ ?inv \wedge \neg ('M \neq a) \} \} \subseteq \{\! \{ 'S = a * b \} \}$  **by** *auto*  
 finally show *?thesis* **by** *blast*  
**qed**

The subsequent version of the proof applies the *vcg* method to reduce the Hoare statement to a purely logical problem that can be solved fully automatically. Note that we have to specify the WHILE loop invariant in the original statement.

**lemma** (in *state-space*)  
**shows**  $\Gamma \vdash \{\! \{ 'M = 0 \wedge 'S = 0 \} \}$   
     *WHILE*  $'M \neq a$   
     *INV*  $\{\! \{ 'S = 'M * b \} \}$   
     *DO*  $'S ::= 'S + b;; 'M ::= 'M + 1$  *OD*  
      $\{\! \{ 'S = a * b \} \}$   
**apply** *vcg*  
**apply** *auto*  
**done**

Here some examples of “breaking” out of a loop

**lemma** (in *state-space*)  
**shows**  $\Gamma \vdash \{\! \{ 'M = 0 \wedge 'S = 0 \} \}$   
     *TRY*  
     *WHILE* *True*  
     *INV*  $\{\! \{ 'S = 'M * b \} \}$

```

    DO IF 'M = a THEN THROW ELSE 'S ::= 'S + b;; 'M ::= 'M + 1
FI OD
    CATCH
    SKIP
    END
    { 'S = a * b }
apply vcg
apply auto
done

```

```

lemma (in state-space)
  shows  $\Gamma \vdash \{ 'M = 0 \wedge 'S = 0 \}$ 
    TRY
      WHILE True
      INV  $\{ 'S = 'M * b \}$ 
      DO IF 'M = a THEN 'Abr ::= "Break";; THROW
        ELSE 'S ::= 'S + b;; 'M ::= 'M + 1
      FI
    OD
    CATCH
      IF 'Abr = "Break" THEN SKIP ELSE Throw FI
    END
    { 'S = a * b }
apply vcg
apply auto
done

```

Some more syntactic sugar, the label statement  $\dots \cdot \dots$  as shorthand for the *TRY-CATCH* above, and the *RAISE* for an state-update followed by a *THROW*.

```

lemma (in state-space)
  shows  $\Gamma \vdash \{ 'M = 0 \wedge 'S = 0 \}$ 
    { 'Abr = "Break" } • WHILE True INV  $\{ 'S = 'M * b \}$ 
    DO IF 'M = a THEN RAISE 'Abr ::= "Break"
      ELSE 'S ::= 'S + b;; 'M ::= 'M + 1
    FI
    OD
    { 'S = a * b }
apply vcg
apply auto
done

```

```

lemma (in state-space)
  shows  $\Gamma \vdash \{ 'M = 0 \wedge 'S = 0 \}$ 
    TRY
      WHILE True
      INV  $\{ 'S = 'M * b \}$ 
      DO IF 'M = a THEN RAISE 'Abr ::= "Break"
        ELSE 'S ::= 'S + b;; 'M ::= 'M + 1
      FI
    END

```

```

      FI
    OD
  CATCH
    IF 'Abr = "Break" THEN SKIP ELSE Throw FI
  END
  { 'S = a * b }
apply vcg
apply auto
done

```

```

lemma (in state-space)
  shows  $\Gamma \vdash \{ 'M = 0 \wedge 'S = 0 \}$ 
     $\{ 'Abr = "Break" \} \cdot \text{WHILE True}$ 
    INV  $\{ 'S = 'M * b \}$ 
    DO IF 'M = a THEN RAISE 'Abr ::= "Break"
      ELSE 'S ::= 'S + b;; 'M ::= 'M + 1
    FI
    OD
  { 'S = a * b }
apply vcg
apply auto
done

```

Blocks

```

lemma (in state-space)
  shows  $\Gamma \vdash \{ T = i \} \text{LOC } T;; T ::= 2 \text{ COL } \{ T \leq i \}$ 
  apply vcg
  by simp

```

## 18.4 Summing Natural Numbers

We verify an imperative program to sum natural numbers up to a given limit. First some functional definition for proper specification of the problem.

```

primrec
  sum :: (nat => nat) => nat => nat
where
  sum f 0 = 0
| sum f (Suc n) = f n + sum f n

```

```

syntax
  -sum :: idt => nat => nat => nat
  (⟨SUMM -<- . -> [0, 0, 10] 10)

```

```

syntax-consts
  -sum == sum

```

```

translations
  SUMM j<k. b == CONST sum ( $\lambda j. b$ ) k

```

The following proof is quite explicit in the individual steps taken, with the *vcg* method only applied locally to take care of assignment and sequential

composition. Note that we express intermediate proof obligation in pure logic, without referring to the state space.

**theorem** (in *state-space*)

**shows**  $\Gamma \vdash \{True\}$   
 $\quad 'S ::= 0;; 'I ::= 1;;$   
 $\quad WHILE 'I \neq n$   
 $\quad DO$   
 $\quad \quad 'S ::= 'S + 'I;;$   
 $\quad \quad 'I ::= 'I + 1$   
 $\quad OD$   
 $\quad \{ 'S = (SUMM j < n. j) \}$   
(is  $\Gamma \vdash - (-;; ?while) -$ )

**proof** –

**let**  $?sum = \lambda k. SUMM j < k. j$   
**let**  $?inv = \lambda s i. s = ?sum i$

**have**  $\Gamma \vdash \{True\} 'S ::= 0;; 'I ::= 1 \{?inv 'S 'I\}$

**proof** –

**have**  $True \longrightarrow 0 = ?sum 1$

**by** *simp*

**also have**  $\Gamma \vdash \{...\} 'S ::= 0;; 'I ::= 1 \{?inv 'S 'I\}$

**by** *vcg*

**finally show** *?thesis* .

**qed**

**also have**  $\Gamma \vdash \{?inv 'S 'I\} ?while \{?inv 'S 'I \wedge \neg 'I \neq n\}$

**proof**

**let**  $?body = 'S ::= 'S + 'I;; 'I ::= 'I + 1$

**have**  $\bigwedge s i. ?inv s i \wedge i \neq n \longrightarrow ?inv (s + i) (i + 1)$

**by** *simp*

**also have**  $\Gamma \vdash \{ 'S + 'I = ?sum ('I + 1) \} ?body \{?inv 'S 'I\}$

**by** *vcg*

**finally show**  $\Gamma \vdash \{?inv 'S 'I \wedge 'I \neq n\} ?body \{?inv 'S 'I\}$  .

**qed**

**also have**  $\bigwedge s i. s = ?sum i \wedge \neg i \neq n \longrightarrow s = ?sum n$

**by** *simp*

**finally show** *?thesis* .

**qed**

The next version uses the *vcg* method, while still explaining the resulting proof obligations in an abstract, structured manner.

**theorem** (in *state-space*)

**shows**  $\Gamma \vdash \{True\}$   
 $\quad 'S ::= 0;; 'I ::= 1;;$   
 $\quad WHILE 'I \neq n$   
 $\quad INV \{ 'S = (SUMM j < 'I. j) \}$   
 $\quad DO$   
 $\quad \quad 'S ::= 'S + 'I;;$   
 $\quad \quad 'I ::= 'I + 1$   
 $\quad OD$

```

    { 'S = (SUMM j<n. j) }
proof -
  let ?sum = λk. SUMM j<k. j
  let ?inv = λs i. s = ?sum i

  show ?thesis
  proof vcg
    show ?inv 0 1 by simp
  next
    fix i s assume ?inv s i i ≠ n
    thus ?inv (s + i) (i + 1) by simp
  next
    fix i s assume x: ?inv s i ¬ i ≠ n
    thus s = ?sum n by simp
  qed
qed

```

Certainly, this proof may be done fully automatically as well, provided that the invariant is given beforehand.

```

theorem (in state-space)
  shows Γ ⊢ { True }
    'S ::= 0;; 'I ::= 1;;
    WHILE 'I ≠ n
    INV { 'S = (SUMM j<'I. j) }
    DO
      'S ::= 'S + 'I;;
      'I ::= 'I + 1
    OD
    { 'S = (SUMM j<n. j) }
  apply vcg
  apply auto
  done

```

## 18.5 SWITCH

```

lemma (in state-space)
  shows Γ ⊢ { 'N = 5 } SWITCH 'B
    { True } ⇒ 'N ::= 6
    | { False } ⇒ 'N ::= 7
  END
  { 'N > 5 }
  apply vcg
  apply simp
  done

```

```

lemma (in state-space)
  shows Γ ⊢ { 'N = 5 } SWITCH 'N
    { v. v < 5 } ⇒ 'N ::= 6
    | { v. v ≥ 5 } ⇒ 'N ::= 7

```

```

      END
    } 'N > 5 }
  apply vcg
  apply simp
done

```

## 18.6 (Mutually) Recursive Procedures

### 18.6.1 Factorial

We want to define a procedure for the factorial. We first define a HOL functions that calculates it to specify the procedure later on.

```

primrec fac:: nat  $\Rightarrow$  nat
where
fac 0 = 1 |
fac (Suc n) = (Suc n) * fac n

lemma fac-simp [simp]: 0 < i  $\implies$  fac i = i * fac (i - 1)
  by (cases i) simp-all

```

Now we define the procedure

```

procedures
  Fac (N::nat|R::nat)
  IF 'N = 0 THEN 'R ::= 1
  ELSE 'R ::= CALL Fac('N - 1);;
  'R ::= 'N * 'R
  FI

```

**print-locale** *Fac-impl*

To see how a call is syntactically translated you can switch off the printing translation via the configuration option *hoare-use-call-tr'*

```

context Fac-impl
begin

```

'R ::= CALL *Fac*() is internally:

```

declare [[hoare-use-call-tr' = false]]

```

```

call ( $\lambda s$ . s(locals := update project-Nat-nat inject-Nat-nat N-'Fac-' (K-statefun
(lookup project-Nat-nat N-'Fac-' (locals s))) (locals s))) Fac-'proc ( $\lambda s$  t.
s(globals := globals t)) ( $\lambda i$  t. 'R ::= lookup project-Nat-nat R-'Fac-' (locals
t))

```

```

term CALL Fac('N,'R)
declare [[hoare-use-call-tr' = true]]

```

Now let us prove that *Fac* meets its specification.

```

end

```



```

lemma (in Fac-impl) Fac-spec':
  shows  $\forall \sigma. \Gamma, \Theta \vdash \{\sigma\} \text{PROC Fac}('N, 'R) \{\{R = \text{fac } \sigma N\}\}$ 
  apply (hoare-rule HoarePartial.ProcRec1)
  apply vcg
  apply simp
  done

```

Since the factorial was implemented recursively, the main ingredient of this proof is, to assume that the specification holds for the recursive call of *Fac* and prove the body correct. The assumption for recursive calls is added to the context by the rule *HoarePartial.ProcRec1* (also derived from general rule for mutually recursive procedures):

$$\llbracket \forall Z. \Gamma, \Theta \cup (\bigcup_Z \{(P\ Z, p, Q\ Z, A\ Z)\}) \vdash_{/F} (P\ Z) \text{ the } (\Gamma\ p) (Q\ Z), (A\ Z); \\ p \in \text{dom } \Gamma \rrbracket \\ \implies \forall Z. \Gamma, \Theta \vdash_{/F} (P\ Z) \text{ Call } p (Q\ Z), (A\ Z)$$

The verification condition generator will infer the specification out of the context when it encounters a recursive call of the factorial.

We can also step through verification condition generation. When the verification condition generator encounters a procedure call it tries to use the rule *ProcSpec*. To be successful there must be a specification of the procedure in the context.

```

lemma (in Fac-impl) Fac-spec1:
  shows  $\forall \sigma. \Gamma, \Theta \vdash \{\sigma\} 'R ::= \text{PROC Fac}('N) \{\{R = \text{fac } \sigma N\}\}$ 
  apply (hoare-rule HoarePartial.ProcRec1)
  apply vcg-step
  apply vcg-step
  apply vcg-step
  apply vcg-step
  apply vcg-step
  apply simp
  done

```

Here some Isar style version of the proof

```

lemma (in Fac-impl) Fac-spec2:

```

```

  shows  $\forall \sigma. \Gamma, \Theta \vdash \{\sigma\} 'R ::= \text{PROC Fac}('N) \{\{R = \text{fac } \sigma N\}\}$ 
proof (hoare-rule HoarePartial.ProcRec1)
  have Fac-spec:  $\forall \sigma. \Gamma, (\Theta \cup (\bigcup \sigma. \{(\{\sigma\}, \text{Fac-}'proc, \{\{R = \text{fac } \sigma N\}, \{\})\})) \vdash \{\sigma\} 'R ::= \text{PROC Fac}('N) \{\{R = \text{fac } \sigma N\}\}$ 
    apply (rule allI)
    apply (rule hoarep.Asm)
    by simp
  show  $\forall \sigma. \Gamma, (\Theta \cup (\bigcup \sigma. \{(\{\sigma\}, \text{Fac-}'proc, \{\{R = \text{fac } \sigma N\}, \{\})\}))$ 

```

```

       $\vdash \{\sigma\} \text{ IF } 'N = 0 \text{ THEN } 'R ::= 1$ 
       $\text{ ELSE } 'R ::= \text{ CALL } \text{Fac}('N - 1);; 'R ::= 'N * 'R \text{ FI } \{\{ 'R = \text{fac } \sigma N \}$ 
apply vcg
apply simp
done
qed

```

To avoid retyping of potentially large pre and postconditions in the previous proof we can use the casual term abbreviations of the Isar language.

```

lemma (in Fac-impl) Fac-spec3:
  shows  $\forall \sigma. \Gamma, \Theta \vdash \{\sigma\} 'R ::= \text{PROC } \text{Fac}('N) \{\{ 'R = \text{fac } \sigma N \}$ 
    (is  $\forall \sigma. \Gamma, \Theta \vdash (?Pre \sigma) ?Fac (?Post \sigma)$ )
proof (hoare-rule HoarePartial.ProcRec1)
  have Fac-spec:  $\forall \sigma. \Gamma, (\Theta \cup (\bigcup \sigma. \{ (?Pre \sigma, \text{Fac-}^l \text{proc}, ?Post \sigma, \{ \}) \}))$ 
     $\vdash (?Pre \sigma) ?Fac (?Post \sigma)$ 
    apply (rule allI)
    apply (rule hoarep.Asm)
    by simp
  show  $\forall \sigma. \Gamma, (\Theta \cup (\bigcup \sigma. \{ (?Pre \sigma, \text{Fac-}^l \text{proc}, ?Post \sigma, \{ \}) \}))$ 
     $\vdash (?Pre \sigma) \text{ IF } 'N = 0 \text{ THEN } 'R ::= 1$ 
     $\text{ ELSE } 'R ::= \text{ CALL } \text{Fac}('N - 1);; 'R ::= 'N * 'R \text{ FI } (?Post \sigma)$ 
    apply vcg
    apply simp
    done
qed

```

The previous proof pattern has still some kind of inconvenience. The augmented context is always printed in the proof state. That can mess up the state, especially if we have large specifications. This may be annoying if we want to develop single step or structured proofs. In this case it can be a good idea to introduce a new variable for the augmented context.

```

lemma (in Fac-impl) Fac-spec4:
  shows  $\forall \sigma. \Gamma, \Theta \vdash \{\sigma\} 'R ::= \text{PROC } \text{Fac}('N) \{\{ 'R = \text{fac } \sigma N \}$ 
    (is  $\forall \sigma. \Gamma, \Theta \vdash (?Pre \sigma) ?Fac (?Post \sigma)$ )
proof (hoare-rule HoarePartial.ProcRec1)
  define  $\Theta'$  where  $\Theta' = \Theta \cup (\bigcup \sigma. \{ (?Pre \sigma, \text{Fac-}^l \text{proc}, ?Post \sigma, \{ \}) \})$ 
  have Fac-spec:  $\forall \sigma. \Gamma, \Theta' \vdash (?Pre \sigma) ?Fac (?Post \sigma)$ 
    by (unfold  $\Theta'$ -def, rule allI, rule hoarep.Asm) simp

```

We have to name the fact *Fac-spec*, so that the *vcg* can use the specification for the recursive call, since it cannot infer it from the opaque  $\Theta'$ .

```

  show  $\forall \sigma. \Gamma, \Theta' \vdash (?Pre \sigma) \text{ IF } 'N = 0 \text{ THEN } 'R ::= 1$ 
     $\text{ ELSE } 'R ::= \text{ CALL } \text{Fac}('N - 1);; 'R ::= 'N * 'R \text{ FI } (?Post \sigma)$ 
    apply vcg
    apply simp
    done
qed

```

There are different rules available to prove procedure calls, depending on

the kind of postcondition and whether or not the procedure is recursive or even mutually recursive. See for example *HoareTotal.ProcRec1*, *HoareTotal.ProcNoRec1*. They are all derived from the most general rule *HoareTotal.ProcRec*. All of them have some side-conditions concerning the parameter passing protocol and its relation to the pre and postcondition. They can be solved in a uniform fashion. That's why we have created the method *hoare-rule*, which behaves like the method *rule* but automatically tries to solve the side-conditions.

### 18.6.2 Odd and Even

Odd and even are defined mutually recursive here. In the *procedures* command we conjoin both definitions with *and*.

**procedures**

```

odd(N::nat | A::nat) IF 'N=0 THEN 'A==0
                    ELSE IF 'N=1 THEN CALL even ('N - 1, 'A)
                    ELSE CALL odd ('N - 2, 'A)
                    FI
FI

```

**and**

```

even(N::nat | A::nat) IF 'N=0 THEN 'A==1
                     ELSE IF 'N=1 THEN CALL odd ('N - 1, 'A)
                     ELSE CALL even ('N - 2, 'A)
                     FI
FI

```

**print-theorems**

**print-locale!** *odd-even-clique*

To prove the procedure calls to *odd* respectively *even* correct we first derive a rule to justify that we can assume both specifications to verify the bodies. This rule can be derived from the general *HoareTotal.ProcRec* rule. An ML function will do this work:

**ML**  $\langle ML\text{-Thms.bind-thm } (ProcRec2, Hoare.gen\text{-}proc\text{-}rec \text{ @}\{context\} Hoare.Partial\ 2) \rangle$

**lemma** (in *odd-even-clique*)

```

shows odd-spec:  $\forall \sigma. \Gamma \vdash \{\sigma\} 'A ::= PROC\ odd('N)$ 
                 $\{\{\exists b. \sigma N = 2 * b + 'A \wedge 'A < 2\}\} \text{ (is } ?P1)$ 
and even-spec:  $\forall \sigma. \Gamma \vdash \{\sigma\} 'A ::= PROC\ even('N)$ 
                 $\{\{\exists b. \sigma N + 1 = 2 * b + 'A \wedge 'A < 2\}\} \text{ (is } ?P2)$ 

```

**proof** –

```

have ?P1  $\wedge$  ?P2
apply (hoare-rule ProcRec2)
apply vcg
apply clarsimp

```

```

apply (rule-tac x=b + 1 in exI)
apply arith
apply vcg
apply clarsimp
apply arith
done
thus ?P1 ?P2
  by iprover+
qed

```

## 18.7 Expressions With Side Effects

```

lemma (in state-space) shows  $\Gamma \vdash \{\!| True |\!\}$ 
  'N >> n. 'N ::= 'N + 1 >>
  'M >> m. 'M ::= 'M + 1 >>
  'R ::= n + m
   $\{\!| R = 'N + 'M - 2 |\!\}$ 
apply vcg
apply simp
done

```

```

lemma (in Fac-impl) shows
   $\Gamma \vdash \{\!| True |\!\}$ 
  CALL Fac('N) >> n. CALL Fac('N) >> m.
  'R ::= n + m
   $\{\!| R = fac 'N + fac 'N |\!\}$ 
proof -
  note Fac-spec = Fac-spec4
  show ?thesis
  apply vcg
  done
qed

```

```

lemma (in Fac-impl) shows
   $\Gamma \vdash \{\!| True |\!\}$ 
  CALL Fac('N) >> n. CALL Fac(n) >> m.
  'R ::= m
   $\{\!| R = fac (fac 'N) |\!\}$ 
proof -
  note Fac-spec = Fac-spec4
  show ?thesis
  apply vcg
  done
qed

```

## 18.8 Global Variables and Heap

Now we will define and verify some procedures on heap-lists. We consider list structures consisting of two fields, a content element *cont* and a reference to the next list element *next*. We model this by the following state space where every field has its own heap.

```
hoarestate globals-list =
  next :: ref ⇒ ref
  cont :: ref ⇒ nat
```

Updates to global components inside a procedure will always be propagated to the caller. This is implicitly done by the parameter passing syntax translations. The record containing the global variables must begin with the prefix "globals".

We will first define an append function on lists. It takes two references as parameters. It appends the list referred to by the first parameter with the list referred to by the second parameter, and returns the result right into the first parameter.

```
procedures (imports globals-list)
  append(p::ref,q::ref|p::ref)
    IF 'p=Null THEN 'p ::= 'q ELSE 'p → 'next::= CALL append('p→'next, 'q)
  FI
```

```
declare [[hoare-use-call-tr' = false]]
context append-impl
begin
term CALL append('p, 'q, 'p→'next)
end
declare [[hoare-use-call-tr' = true]]
```

Below we give two specifications this time.. The first one captures the functional behaviour and focuses on the entities that are potentially modified by the procedure, the second one is a pure frame condition. The list in the modifies clause has to list all global state components that may be changed by the procedure. Note that we know from the modifies clause that the *cont* parts of the lists will not be changed. Also a small side note on the syntax. We use ordinary brackets in the postcondition of the modifies clause, and also the state components do not carry the acute, because we explicitly note the state *t* here.

The functional specification now introduces two logical variables besides the state space variable  $\sigma$ , namely *Ps* and *Qs*. They are universally quantified and range over both the pre and the postcondition, so that we are able to properly instantiate the specification during the proofs. The syntax  $\{\sigma. \dots\}$  is a shorthand to fix the current state:  $\{s. \sigma = s \dots\}$ .

```

lemma (in append-impl) append-spec:
shows  $\forall \sigma \ Ps \ Qs. \Gamma \vdash$ 
 $\{\sigma. \text{List } 'p \ 'next \ Ps \wedge \text{List } 'q \ 'next \ Qs \wedge \text{set } Ps \cap \text{set } Qs = \{\}\}$ 
 $\quad 'p ::= \text{PROC } \text{append}('p, 'q)$ 
 $\quad \{\text{List } 'p \ 'next \ (Ps @ Qs) \wedge (\forall x. x \notin \text{set } Ps \longrightarrow 'next \ x = \sigma_{next} \ x)\}$ 
apply (hoare-rule HoarePartial.ProcRec1)
apply vcg
apply fastforce
done

```

The modifies clause is equal to a proper record update specification of the following form.

```

lemma (in append-impl) shows  $\{t. t \text{ may-only-modify-globals } Z \text{ in } [next]\}$ 
 $=$ 
 $\{t. \exists next. \text{globals } t = \text{update } id \ id \ next \text{' } (K\text{-statefun } next) (\text{globals } Z)\}$ 
apply (unfold mex-def meq-def)
apply simp
done

```

If the verification condition generator works on a procedure call it checks whether it can find a modifies clause in the context. If one is present the procedure call is simplified before the Hoare rule *HoareTotal.ProcSpec* is applied. Simplification of the procedure call means, that the “copy back” of the global components is simplified. Only those components that occur in the modifies clause will actually be copied back. This simplification is justified by the rule *HoareTotal.ProcModifyReturn*. So after this simplification all global components that do not appear in the modifies clause will be treated as local variables.

You can study the effect of the modifies clause on the following two examples, where we want to prove that (@) does not change the *cont* part of the heap.

```

lemma (in append-impl)
shows  $\Gamma \vdash \{\ 'p = \text{Null} \wedge 'cont = c \} 'p ::= \text{CALL } \text{append}('p, \text{Null}) \{\ 'cont = c \}$ 
apply vcg
oops

```

To prove the frame condition, we have to tell the verification condition generator to use only the modifies clauses and not to search for functional specifications by the parameter *spec=modifies* It will also try to solve the verification conditions automatically.

```

lemma (in append-impl) append-modifies:
shows
 $\forall \sigma. \Gamma \vdash \{\sigma\} 'p ::= \text{PROC } \text{append}('p, 'q) \{t. t \text{ may-only-modify-globals } \sigma \text{ in } [next]\}$ 
apply (hoare-rule HoarePartial.ProcRec1)
apply (vcg spec=modifies)
done

```

```

lemma (in append-impl)
  shows  $\Gamma \vdash \{\!| p = \text{Null} \wedge cont = c \!\!| \} p \rightarrow next ::= \text{CALL } \text{append}(p, \text{Null}) \{\!| cont = c \!\!| \}$ 
  apply vcg
  apply simp
  done

```

Of course we could add the modifies clause to the functional specification as well. But separating both has the advantage that we split up the verification work. We can make use of the modifies clause before we apply the functional specification in a fully automatic fashion.

To verify the body of (@) we do not need the modifies clause, since the specification does not talk about *cont* at all, and we don't access *cont* inside the body. This may be different for more complex procedures.

To prove that a procedure respects the modifies clause, we only need the modifies clauses of the procedures called in the body. We do not need the functional specifications. So we can always prove the modifies clause without functional specifications, but we may need the modifies clause to prove the functional specifications.

### 18.8.1 Insertion Sort

```

primrec sorted:: ('a  $\Rightarrow$  'a  $\Rightarrow$  bool)  $\Rightarrow$  'a list  $\Rightarrow$  bool
where
  sorted le [] = True |
  sorted le (x#xs) = (( $\forall y \in \text{set } xs. le x y$ )  $\wedge$  sorted le xs)

```

```

procedures (imports globals-list)
  insert(r::ref, p::ref | p::ref)
  IF r = Null THEN SKIP
  ELSE IF p = Null THEN p ::= r;; p  $\rightarrow$  next ::= Null
  ELSE IF r  $\rightarrow$  cont  $\leq$  p  $\rightarrow$  cont
    THEN r  $\rightarrow$  next ::= p;; p ::= r
    ELSE p  $\rightarrow$  next ::= CALL insert(r, p  $\rightarrow$  next)
  FI
  FI
  FI

```

In the postcondition of the functional specification there is a small but important subtlety. Whenever we talk about the *cont* part we refer to the one of the pre-state, even in the conclusion of the implication. The reason is, that we have separated out, that *cont* is not modified by the procedure, to the modifies clause. So whenever we talk about unmodified parts in the postcondition we have to use the pre-state part, or explicitly state an equality in the postcondition. The reason is simple. If the postcondition

would talk about  $\prime cont$  instead of  $\sigma cont$ , we will get a new instance of  $cont$  during verification and the postcondition would only state something about this new instance. But as the verification condition generator will use the modifies clause the caller of  $insert$  instead will still have the old  $cont$  after the call. That's the sense of the modifies clause. So the caller and the specification will simply talk about two different things, without being able to relate them (unless an explicit equality is added to the specification).

```

lemma (in insert-impl) insert-modifies:
   $\forall \sigma. \Gamma \vdash \{ \sigma \} \prime p ::= PROC \textit{insert}(\prime r, \prime p) \{ t. t \textit{ may-only-modify-globals } \sigma \textit{ in } [next] \}$ 
apply (hoare-rule HoarePartial.ProcRec1)
apply (vcg spec=modifies)
done

```

```

lemma (in insert-impl) insert-spec:
   $\forall \sigma Ps. \Gamma \vdash \{ \sigma. List \prime p \prime next Ps \wedge sorted (\leq) (map \prime cont Ps) \wedge$ 
     $\prime r \neq Null \wedge \prime r \notin set Ps \}$ 
     $\prime p ::= PROC \textit{insert}(\prime r, \prime p)$ 
     $\{ \exists Qs. List \prime p \prime next Qs \wedge sorted (\leq) (map \sigma cont Qs) \wedge$ 
     $set Qs = \textit{insert} \sigma r (set Ps) \wedge$ 
     $(\forall x. x \notin set Qs \longrightarrow \prime next x = \sigma next x) \}$ 

```

```

apply (hoare-rule HoarePartial.ProcRec1)
apply vcg
apply (intro conjI impI)
apply fastforce
apply fastforce
apply fastforce
apply (clarsimp)
apply force
done

```

```

procedures (imports globals-list)
  insertSort(p::ref | p::ref)
  where r::ref q::ref
  in
     $\prime r ::= Null;$ 
    WHILE ( $\prime p \neq Null$ ) DO
       $\prime q ::= \prime p;$ 
       $\prime p ::= \prime p \rightarrow \prime next;$ 
       $\prime r ::= CALL \textit{insert}(\prime q, \prime r)$ 
    OD;
     $\prime p ::= \prime r$ 

```

```

print-locale insertSort-impl

```

```

lemma (in insertSort-impl) insertSort-modifies:

```



```

shows
   $\forall \sigma. \Gamma \vdash \{ \sigma \} \text{'p} ::= \text{PROC insertSort}(\text{'p})$ 
     $\{ t. t \text{ may-only-modify-globals } \sigma \text{ in } [\text{next}] \}$ 
apply (hoare-rule HoarePartial.ProcRec1)
apply (vcg spec=modifies)
done

```

Insertion sort is not implemented recursively here but with a while loop. Note that the while loop is not annotated with an invariant in the procedure definition. The invariant only comes into play during verification. Therefore we will annotate the body during the proof with the rule *Hoare-Total.annotateI*.

```

lemma (in insertSort-impl) insertSort-body-spec:
shows  $\forall \sigma Ps. \Gamma, \Theta \vdash \{ \sigma. \text{List 'p 'next Ps} \}$ 
   $\text{'p} ::= \text{PROC insertSort}(\text{'p})$ 
   $\{ \exists Qs. \text{List 'p 'next Qs} \wedge \text{sorted} (\leq) (\text{map } \sigma_{\text{cont}} Qs) \wedge$ 
     $\text{set } Qs = \text{set } Ps \}$ 
apply (hoare-rule HoarePartial.ProcRec1)
apply (hoare-rule anno=
   $\text{'r} ::= \text{Null};;$ 
  WHILE 'p ≠ Null
  INV  $\{ \exists Qs Rs. \text{List 'p 'next Qs} \wedge \text{List 'r 'next Rs} \wedge$ 
     $\text{set } Qs \cap \text{set } Rs = \{ \} \wedge$ 
     $\text{sorted} (\leq) (\text{map 'cont } Rs) \wedge \text{set } Qs \cup \text{set } Rs = \text{set } Ps \wedge$ 
     $\text{'cont} = \sigma_{\text{cont}} \}$ 
  DO 'q ::= 'p;; 'p ::= 'p → 'next;; 'r ::= CALL insert('q, 'r) OD;;
   $\text{'p} ::= \text{'r}$  in HoarePartial.annotateI)
apply vcg
apply fastforce
prefer 2
apply fastforce
apply (clarsimp)
apply (rule-tac x=ps in exI)
apply (intro conjI)
apply (rule heap-eq-ListI1)
apply assumption
apply clarsimp
apply (subgoal-tac x≠p ∧ x ∉ set Rs)
apply auto
done

```

## 18.8.2 Memory Allocation and Deallocation

The basic idea of memory management is to keep a list of allocated references in the state space. Allocation of a new reference adds a new reference to the list deallocation removes a reference. Moreover we keep a counter "free" for the free memory.

```

hoarestate globals-list-alloc =

```

```

alloc::ref list
free::nat
next::ref  $\Rightarrow$  ref
cont::ref  $\Rightarrow$  nat
hoarestate locals-list-alloc =
  i::nat
  first::ref
  p::ref
  q::ref
  r::ref
  root::ref
  tmp::ref
locale list-alloc = globals-list-alloc + locals-list-alloc

definition sz = (2::nat)

lemma (in list-alloc)
shows
   $\Gamma, \Theta \vdash \{i = 0 \wedge 'first = Null \wedge n * sz \leq 'free\}$ 
    WHILE  $'i < n$ 
      INV  $\{\exists Ps. List\ 'first\ 'next\ Ps \wedge length\ Ps = 'i \wedge 'i \leq n \wedge$ 
         $set\ Ps \subseteq set\ 'alloc \wedge (n - 'i) * sz \leq 'free\}$ 
      DO
         $'p ::= NEW\ sz\ ['cont ::= 0, 'next ::= Null];;$ 
         $'p \rightarrow 'next ::= 'first;;$ 
         $'first ::= 'p;;$ 
         $'i ::= 'i + 1$ 
      OD
     $\{\exists Ps. List\ 'first\ 'next\ Ps \wedge length\ Ps = n \wedge set\ Ps \subseteq set\ 'alloc\}$ 
apply (vcg)
apply simp
apply clarsimp
apply (rule conjI)
apply clarsimp
apply (rule-tac x=new (set alloc)#Ps in exI)
apply clarsimp
apply (rule conjI)
apply fastforce
apply (simp add: sz-def)
apply (simp add: sz-def)
apply fastforce
done

```

```

lemma (in list-alloc)
shows
   $\Gamma \vdash \{i = 0 \wedge 'first = Null \wedge n * sz \leq 'free\}$ 
    WHILE  $'i < n$ 
      INV  $\{\exists Ps. List\ 'first\ 'next\ Ps \wedge length\ Ps = 'i \wedge 'i \leq n \wedge$ 

```

```

      set Ps ⊆ set 'alloc ∧ (n - 'i)*sz ≤ 'free}
DO
  'p := NNEW sz ['cont:=0, 'next:= Null];;
  'p→'next := 'first;;
  'first := 'p;;
  'i := 'i + 1
OD
{∃ Ps. List 'first 'next Ps ∧ length Ps = n ∧ set Ps ⊆ set 'alloc}

```

```

apply (vcg)
apply simp
apply clarsimp
apply (rule conjI)
apply clarsimp
apply (rule-tac x=new (set alloc)#Ps in exI)
apply clarsimp
apply (rule conjI)
apply fastforce
apply (simp add: sz-def)
apply (simp add: sz-def)
apply fastforce
done

```

## 18.9 Fault Avoiding Semantics

If we want to ensure that no runtime errors occur we can insert guards into the code. We will not be able to prove any nontrivial Hoare triple about code with guards, if we cannot show that the guards will never fail. A trivial Hoare triple is one with an empty precondition.

```

lemma (in list-alloc)  $\Gamma, \Theta \vdash \{True\} \{p \neq Null\} \mapsto p \rightarrow next := p \{True\}$ 
apply vcg
oops

```

```

lemma (in list-alloc)  $\Gamma, \Theta \vdash \{\} \{p \neq Null\} \mapsto p \rightarrow next := p \{True\}$ 
apply vcg
done

```

Let us consider this small program that reverts a list. At first without guards.

```

lemma (in list-alloc)
  shows
   $\Gamma, \Theta \vdash \{List\ p\ next\ Ps \wedge List\ q\ next\ Qs \wedge set\ Ps \cap set\ Qs = \{\} \wedge$ 
     $set\ Ps \subseteq set\ 'alloc \wedge set\ Qs \subseteq set\ 'alloc\}$ 
  WHILE  $p \neq Null$ 
  INV  $\{\exists ps\ qs. List\ p\ next\ ps \wedge List\ q\ next\ qs \wedge set\ ps \cap set\ qs = \{\} \wedge$ 
     $rev\ ps\ @\ qs = rev\ Ps\ @\ Qs \wedge$ 
     $set\ ps \subseteq set\ 'alloc \wedge set\ qs \subseteq set\ 'alloc\}$ 
  DO  $r := p;$ 

```

```

    'p ::= 'p → 'next;;
    'r → 'next ::= 'q;;
    'q ::= 'r OD
  {List 'q 'next (rev Ps @ Qs) ∧ set Ps ⊆ set 'alloc ∧ set Qs ⊆ set 'alloc}
apply (vcg)
apply fastforce+
done

```

If we want to ensure that we do not dereference *Null* or access unallocated memory, we have to add some guards.

```

lemma (in list-alloc)
  shows
   $\Gamma, \Theta \vdash \{List\ 'p\ 'next\ Ps \wedge List\ 'q\ 'next\ Qs \wedge set\ Ps \cap set\ Qs = \{\} \wedge$ 
     $set\ Ps \subseteq set\ 'alloc \wedge set\ Qs \subseteq set\ 'alloc\}$ 
  WHILE 'p ≠ Null
  INV  $\{\exists ps\ qs. List\ 'p\ 'next\ ps \wedge List\ 'q\ 'next\ qs \wedge set\ ps \cap set\ qs = \{\} \wedge$ 
     $rev\ ps @ qs = rev\ Ps @ Qs \wedge$ 
     $set\ ps \subseteq set\ 'alloc \wedge set\ qs \subseteq set\ 'alloc\}$ 
  DO 'r ::= 'p;;
     $\{\ 'p \neq Null \wedge 'p \in set\ 'alloc \} \mapsto 'p ::= 'p \rightarrow 'next;;$ 
     $\{\ 'r \neq Null \wedge 'r \in set\ 'alloc \} \mapsto 'r \rightarrow 'next ::= 'q;;$ 
    'q ::= 'r OD
  {List 'q 'next (rev Ps @ Qs) ∧ set Ps ⊆ set 'alloc ∧ set Qs ⊆ set 'alloc}
apply (vcg)
apply fastforce+
done

```

We can also just prove that no faults will occur, by giving the trivial post-condition.

```

lemma (in list-alloc) rev-noFault:
  shows
   $\Gamma, \Theta \vdash \{List\ 'p\ 'next\ Ps \wedge List\ 'q\ 'next\ Qs \wedge set\ Ps \cap set\ Qs = \{\} \wedge$ 
     $set\ Ps \subseteq set\ 'alloc \wedge set\ Qs \subseteq set\ 'alloc\}$ 
  WHILE 'p ≠ Null
  INV  $\{\exists ps\ qs. List\ 'p\ 'next\ ps \wedge List\ 'q\ 'next\ qs \wedge set\ ps \cap set\ qs = \{\} \wedge$ 
     $rev\ ps @ qs = rev\ Ps @ Qs \wedge$ 
     $set\ ps \subseteq set\ 'alloc \wedge set\ qs \subseteq set\ 'alloc\}$ 
  DO 'r ::= 'p;;
     $\{\ 'p \neq Null \wedge 'p \in set\ 'alloc \} \mapsto 'p ::= 'p \rightarrow 'next;;$ 
     $\{\ 'r \neq Null \wedge 'r \in set\ 'alloc \} \mapsto 'r \rightarrow 'next ::= 'q;;$ 
    'q ::= 'r OD
  UNIV, UNIV
apply (vcg)
apply fastforce+
done

```

**lemma** (in list-alloc) rev-moduloGuards:

**shows**

$\Gamma, \Theta \vdash_{\{\text{True}\}} \{ \text{List } 'p \text{ 'next } Ps \wedge \text{List } 'q \text{ 'next } Qs \wedge \text{set } Ps \cap \text{set } Qs = \{\} \wedge$   
 $\text{set } Ps \subseteq \text{set } 'alloc \wedge \text{set } Qs \subseteq \text{set } 'alloc \}$   
**WHILE**  $'p \neq \text{Null}$   
**INV**  $\{ \exists ps \ qs. \text{List } 'p \text{ 'next } ps \wedge \text{List } 'q \text{ 'next } qs \wedge \text{set } ps \cap \text{set } qs = \{\} \wedge$   
 $\text{rev } ps @ qs = \text{rev } Ps @ Qs \wedge$   
 $\text{set } ps \subseteq \text{set } 'alloc \wedge \text{set } qs \subseteq \text{set } 'alloc \}$   
**DO**  $'r := 'p;$   
 $\{ 'p \neq \text{Null} \wedge 'p \in \text{set } 'alloc \} \checkmark \mapsto 'p := 'p \rightarrow 'next;;$   
 $\{ 'r \neq \text{Null} \wedge 'r \in \text{set } 'alloc \} \checkmark \mapsto 'r \rightarrow 'next := 'q;;$   
 $'q := 'r$  **OD**  
 $\{ \text{List } 'q \text{ 'next } (\text{rev } Ps @ Qs) \wedge \text{set } Ps \subseteq \text{set } 'alloc \wedge \text{set } Qs \subseteq \text{set } 'alloc \}$   
**apply** *vcg*  
**apply** *fastforce+*  
**done**

**lemma** *CombineStrip'*:  
**assumes** *deriv*:  $\Gamma, \Theta \vdash_{/F} P \ c' \ Q, A$   
**assumes** *deriv-strip*:  $\Gamma, \Theta \vdash_{\{\}} P \ c'' \ UNIV, UNIV$   
**assumes** *c''*:  $c'' = \text{mark-guards } \text{False} \ (\text{strip-guards } (-F) \ c')$   
**assumes** *c*:  $c = \text{mark-guards } \text{False} \ c'$   
**shows**  $\Gamma, \Theta \vdash_{\{\}} P \ c \ Q, A$

**proof** –  
**from** *deriv-strip* [*simplified c''*]  
**have**  $\Gamma, \Theta \vdash P \ (\text{strip-guards } (-F) \ c') \ UNIV, UNIV$   
**by** (*rule HoarePartialProps.MarkGuardsD*)  
**with** *deriv*  
**have**  $\Gamma, \Theta \vdash P \ c' \ Q, A$   
**by** (*rule HoarePartialProps.CombineStrip*)  
**hence**  $\Gamma, \Theta \vdash P \ \text{mark-guards } \text{False} \ c' \ Q, A$   
**by** (*rule HoarePartialProps.MarkGuardsI*)  
**thus** *?thesis*  
**by** (*simp add: c*)  
**qed**

We can then combine the prove that no fault will occur with the functional prove of the programm without guards to get the full prove by the rule  
 $\llbracket ?\Gamma, ?\Theta \vdash_{/?F} ?P \ ?c \ ?Q, ?A; ?\Gamma, ?\Theta \vdash ?P \ \text{strip-guards } (- ?F) \ ?c \ UNIV, UNIV \rrbracket$   
 $\implies ?\Gamma, ?\Theta \vdash ?P \ ?c \ ?Q, ?A$

**lemma** (*in list-alloc*)  
**shows**  
 $\Gamma, \Theta \vdash \{ \text{List } 'p \text{ 'next } Ps \wedge \text{List } 'q \text{ 'next } Qs \wedge \text{set } Ps \cap \text{set } Qs = \{\} \wedge$   
 $\text{set } Ps \subseteq \text{set } 'alloc \wedge \text{set } Qs \subseteq \text{set } 'alloc \}$   
**WHILE**  $'p \neq \text{Null}$   
**INV**  $\{ \exists ps \ qs. \text{List } 'p \text{ 'next } ps \wedge \text{List } 'q \text{ 'next } qs \wedge \text{set } ps \cap \text{set } qs = \{\} \wedge$   
 $\text{rev } ps @ qs = \text{rev } Ps @ Qs \wedge$

$$\begin{array}{l}
\text{set } ps \subseteq \text{set } 'alloc \wedge \text{set } qs \subseteq \text{set } 'alloc\} \\
\text{DO } 'r ::= 'p;; \\
\quad \{\{ 'p \neq \text{Null} \wedge 'p \in \text{set } 'alloc\} \mapsto 'p ::= 'p \rightarrow 'next;; \\
\quad \{\{ 'r \neq \text{Null} \wedge 'r \in \text{set } 'alloc\} \mapsto 'r \rightarrow 'next ::= 'q;; \\
\quad 'q ::= 'r \text{ OD} \\
\{\{ \text{List } 'q 'next (\text{rev } Ps @ Qs) \wedge \text{set } Ps \subseteq \text{set } 'alloc \wedge \text{set } Qs \subseteq \text{set } 'alloc\}
\end{array}$$

**apply** (*rule CombineStrip'* [*OF rev-moduloGuards rev-noFault*])  
**apply** *simp*  
**apply** *simp*  
**done**

In the previous example the effort to split up the prove did not really pay off. But when we think of programs with a lot of guards and complicated specifications it may be better to first focus on a prove without the messy guards. Maybe it is possible to automate the no fault proofs so that it suffices to focus on the stripped program.

**context** *list-alloc*  
**begin**

The purpose of guards is to watch for faults that can occur during evaluation of expressions. In the example before we watched for null pointer dereferencing or memory faults. We can also look for array index bounds or division by zero. As the condition of a while loop is evaluated in each iteration we cannot just add a guard before the while loop. Instead we need a special guard for the condition. Example: *WHILE* (*False*,  $\{\{ 'p \neq \text{Null}\} \mapsto 'p \rightarrow 'next \neq \text{Null} \text{ DO } \text{SKIP} \text{ OD}$

**end**

## 18.10 Cicular Lists

**definition**

$$\begin{array}{l}
\text{distPath} :: \text{ref} \Rightarrow (\text{ref} \Rightarrow \text{ref}) \Rightarrow \text{ref} \Rightarrow \text{ref list} \Rightarrow \text{bool} \text{ \textbf{where}} \\
\text{distPath } x \text{ next } y \text{ as} = (\text{Path } x \text{ next } y \text{ as} \wedge \text{distinct } \text{as})
\end{array}$$

**lemma** *neq-dP*:  $\llbracket p \neq q; \text{Path } p \text{ h } q \text{ Ps}; \text{distinct } Ps \rrbracket \implies$   
 $\exists Qs. p \neq \text{Null} \wedge Ps = p \# Qs \wedge p \notin \text{set } Qs$   
**by** (*cases Ps, auto*)

**lemma** (**in** *list-alloc*) *circular-list-rev-I*:

$$\begin{array}{l}
\Gamma, \Theta \vdash \{\{ 'root = r \wedge \text{distPath } 'root 'next 'root (r \# Ps) \}\} \\
'p ::= 'root;; 'q ::= 'root \rightarrow 'next;; \\
\text{WHILE } 'q \neq 'root \\
\text{INV } \{\{ \exists ps \text{ qs. distPath } 'p 'next 'root ps \wedge \text{distPath } 'q 'next 'root qs \wedge \\
'root = r \wedge r \neq \text{Null} \wedge r \notin \text{set } Ps \wedge \text{set } ps \cap \text{set } qs = \{\} \wedge \\
Ps = (\text{rev } ps) @ qs \}\} \\
\text{DO } 'tmp ::= 'q;; 'q ::= 'q \rightarrow 'next;; 'tmp \rightarrow 'next ::= 'p;; 'p ::= 'tmp \text{ OD};
\end{array}$$

```

    'root→'next ::= 'p
    { 'root = r ∧ distPath 'root 'next 'root (r#rev Ps) }
apply (simp only:distPath-def)
apply vcg
apply (rule-tac x=[] in exI)
apply fastforce
apply clarsimp
apply (drule (2) neq-dP)
apply (rule-tac x=q # ps in exI)
apply clarsimp
apply fastforce
done

```

```

lemma path-is-list:  $\wedge a \text{ next } b. \llbracket \text{Path } b \text{ next } a \text{ Ps} ; a \notin \text{set Ps} ; a \neq \text{Null} \rrbracket$ 
 $\implies \text{List } b \text{ (next(a := Null)) (Ps @ [a])}$ 
apply (induct Ps)
apply (auto simp add:fun-upd-apply)
done

```

The simple algorithm for acyclic list reversal, with modified annotations, works for cyclic lists as well.:

```

lemma (in list-alloc) circular-list-rev-II:
   $\Gamma, \Theta \vdash$ 
  { 'p = r ∧ distPath 'p 'next 'p (r#Ps) }
  'q ::= Null;;
  WHILE 'p ≠ Null
  INV
  { (( 'q = Null )  $\longrightarrow$  (  $\exists ps. \text{distPath 'p 'next } r \text{ ps} \wedge ps = r\#Ps$  ))  $\wedge$ 
    (( 'q ≠ Null )  $\longrightarrow$  (  $\exists ps \ qs. \text{distPath 'q 'next } r \text{ qs} \wedge \text{List 'p 'next } ps \wedge$ 
      set ps  $\cap$  set qs = { }  $\wedge \text{rev } qs @ ps = Ps@[r]$  ))  $\wedge$ 
     $\neg$  ( 'p = Null  $\wedge$  'q = Null  $\wedge$  r = Null )
  }
  DO
  'tmp ::= 'p;; 'p ::= 'p→'next;; 'tmp→'next ::= 'q;; 'q ::= 'tmp
  OD
  { 'q = r ∧ distPath 'q 'next 'q (r # rev Ps) }

apply (simp only:distPath-def)
apply vcg
apply clarsimp
apply clarsimp
apply (case-tac (q = Null))
apply (fastforce intro: path-is-list)
apply clarify
apply (rule-tac x=psa in exI)
apply (rule-tac x= p # qs in exI)
apply force

```

**apply** *fastforce*  
**done**

Although the above algorithm is more succinct, its invariant looks more involved. The reason for the case distinction on  $q$  is due to the fact that during execution, the pointer variables can point to either cyclic or acyclic structures.

When working on lists, its sometimes better to remove *fun-upd-apply* from the simpset, and instead include *fun-upd-same* and *fun-upd-other* to the simpset

**lemma** (in *state-space*)  $\Gamma \vdash \{\sigma\}$   
 $I ::= 'M;;$   
 $ANNO \tau. \{\tau. 'I = {}^\sigma M\}$   
 $'M ::= 'N;; 'N ::= 'I$   
 $\{\{ 'M = {}^\tau N \wedge 'N = {}^\tau I \}$   
 $\{ 'M = {}^\sigma N \wedge 'N = {}^\sigma M \}$

**apply** *vcg*  
**apply** *auto*  
**done**

**context** *state-space*

**begin**

**term**  $ANNO (\tau, m, k). (\{\tau. 'M = m\}) 'M ::= 'N;; 'N ::= 'I \{ 'M = {}^\tau N \ \& \ 'N = {}^\tau I \}, \{\}$

**end**

**lemma** (in *state-space*)  $\Gamma \vdash (\{\sigma\} \cap \{ 'M = 0 \wedge 'S = 0 \})$   
 $(ANNO \tau. (\{\tau\} \cap \{ 'A = {}^\sigma A \wedge 'I = {}^\sigma I \wedge 'M = 0 \wedge 'S = 0 \}))$   
 $WHILE 'M \neq 'A$   
 $INV \{ 'S = 'M * 'I \wedge 'A = {}^\tau A \wedge 'I = {}^\tau I \}$   
 $DO 'S ::= 'S + 'I;; 'M ::= 'M + 1 OD$   
 $\{ 'S = {}^\tau A * {}^\tau I \}$   
 $\{ 'S = {}^\sigma A * {}^\sigma I \}$

**apply** *vcg-step*  
**apply** *vcg-step*  
**apply** *simp*  
**apply** *vcg-step*  
**apply** *vcg-step*  
**apply** *simp*  
**apply** *vcg*  
**apply** *simp*  
**apply** *simp*  
**apply** *vcg-step*  
**apply** *auto*  
**done**

Just some test on marked, guards

**lemma** (in *state-space*)  $\Gamma \vdash \{ True \} WHILE \{ P 'N \} \checkmark, \{ Q 'M \} \#, \{ R 'N \} \mapsto 'N$



```

< 'M
      INV {'N < 2} DO
      'N ::= 'M
    OD
  {hard}
apply vcg
oops

lemma (in state-space)  $\Gamma \vdash \{True\} \{True\} \text{ WHILE } \{P \ 'N \} \checkmark, \{Q \ 'M \} \#, \{R \ 'N\} \mapsto 'N < 'M$ 
      INV {'N < 2} DO
      'N ::= 'M
    OD
  {hard}
apply vcg
oops

end

```

## 19 Examples for Total Correctness

```

theory VcgExTotal imports ../HeapList ../Vcg begin

```

```

record 'g vars = 'g state +

```

```

  A-' :: nat
  I-' :: nat
  M-' :: nat
  N-' :: nat
  R-' :: nat
  S-' :: nat
  Abr-': string

```

```

lemma  $\Gamma \vdash_t \{ 'M = 0 \wedge 'S = 0 \}$ 
      WHILE 'M  $\neq$  a
      INV {'S = 'M * b  $\wedge$  'M  $\leq$  a}
      VAR MEASURE a - 'M
      DO 'S ::= 'S + b;; 'M ::= 'M + 1 OD
      {'S = a * b}

```

```

apply vcg
apply (auto)
done

```

```

lemma  $\Gamma \vdash_t \{ 'I \leq 3 \}$ 
      WHILE 'I < 10 INV {'I  $\leq$  10} VAR MEASURE 10 - 'I
      DO
      'I ::= 'I + 1
      OD
      {'I = 10}

```

**apply** *vcg*  
**apply** *auto*  
**done**

Total correctness of a nested loop. In the inner loop we have to express that the loop variable of the outer loop is not changed. We use *FIX* to introduce a new logical variable

**lemma**  $\Gamma \vdash_t \{\{ 'M=0 \wedge 'N=0 \}$   
      $WHILE ('M < i)$   
      $INV \{\{ 'M \leq i \wedge ('M \neq 0 \longrightarrow 'N = j) \wedge 'N \leq j \}$   
      $VAR MEASURE (i - 'M)$   
      $DO$   
          $'N ::= 0;;$   
          $WHILE ('N < j)$   
          $FIX m.$   
          $INV \{\{ 'M=m \wedge 'N \leq j \}$   
          $VAR MEASURE (j - 'N)$   
          $DO$   
              $'N ::= 'N + 1$   
          $OD;;$   
      $'M ::= 'M + 1$   
      $OD$   
      $\{\{ 'M=i \wedge ('M \neq 0 \longrightarrow 'N=j) \}$

**apply** *vcg*  
**apply** *simp-all*  
**apply** *arith*  
**done**

**primrec** *fac*:: *nat*  $\Rightarrow$  *nat*  
**where**  
*fac* 0 = 1 |  
*fac* (Suc *n*) = (Suc *n*) \* *fac* *n*

**lemma** *fac-simp* [*simp*]:  $0 < i \implies fac\ i = i * fac\ (i - 1)$   
**by** (*cases* *i*) *simp-all*

**procedures**

$Fac\ (N\ |\ R) = IF\ 'N = 0\ THEN\ 'R ::= 1$   
                    $ELSE\ CALL\ Fac('N - 1, 'R);;$   
                    $'R ::= 'N * 'R$   
**FI**

**lemma** (**in** *Fac-impl*) *Fac-spec*:  
**shows**  $\forall n. \Gamma \vdash_t \{\{ 'N=n \} 'R ::= PROC\ Fac('N) \{\{ 'R = fac\ n \}$   
**apply** (*hoare-rule* *HoareTotal.ProcRec1* [**where** *r=measure* ( $\lambda(s,p).\ ^sN$ )])  
**apply** *vcg*  
**apply** *simp*  
**done**

**procedures**

$$p91(R, N \mid R) = \text{IF } 100 < 'N \text{ THEN } 'R ::= 'N - 10 \\ \text{ELSE } 'R ::= \text{CALL } p91('R, 'N+11);; \\ 'R ::= \text{CALL } p91('R, 'R) \text{ FI}$$

$$p91\text{-spec: } \forall n. \Gamma \vdash_t \{ \{ 'N=n \} 'R ::= \text{PROC } p91('R, 'N) \\ \{ \text{if } 100 < n \text{ then } 'R = n - 10 \text{ else } 'R = 91 \}, \{ \}$$
**lemma (in p91-impl) p91-spec:**

$$\text{shows } \forall \sigma. \Gamma \vdash_t \{ \sigma \} 'R ::= \text{PROC } p91('R, 'N) \\ \{ \text{if } 100 < {}^\sigma N \text{ then } 'R = {}^\sigma N - 10 \text{ else } 'R = 91 \}, \{ \}$$

$$\text{apply (hoare-rule HoareTotal.ProcRec1 [where } r=\text{measure } (\lambda(s,p). 101 - {}^s N)])$$

$$\text{apply vcg}$$

$$\text{apply clarsimp}$$

$$\text{apply arith}$$

$$\text{done}$$
**record globals-list =**

$$\text{next-}' :: \text{ref} \Rightarrow \text{ref}$$

$$\text{cont-}' :: \text{ref} \Rightarrow \text{nat}$$
**record 'g list-vars = 'g state +**

$$p\text{'}' :: \text{ref}$$

$$q\text{'}' :: \text{ref}$$

$$r\text{'}' :: \text{ref}$$

$$\text{root-}' :: \text{ref}$$

$$\text{tmp-}' :: \text{ref}$$
**procedures**

$$\text{append}(p, q \mid p) =$$

$$\text{IF } 'p = \text{Null} \text{ THEN } 'p ::= 'q \text{ ELSE } 'p \rightarrow 'next ::= \text{CALL } \text{append}('p \rightarrow 'next, 'q)$$

$$\text{FI}$$
**lemma (in append-impl)**

$$\text{shows}$$

$$\forall \sigma \ Ps \ Qs. \Gamma \vdash_t$$

$$\{ \sigma. \text{List } 'p \ 'next \ Ps \wedge \text{List } 'q \ 'next \ Qs \wedge \text{set } Ps \cap \text{set } Qs = \{ \} \}$$

$$'p ::= \text{PROC } \text{append}('p, 'q)$$

$$\{ \text{List } 'p \ 'next \ (Ps @ Qs) \wedge (\forall x. x \notin \text{set } Ps \longrightarrow 'next \ x = {}^\sigma next \ x) \}$$

$$\text{apply (hoare-rule HoareTotal.ProcRec1}$$

$$[\text{where } r=\text{measure } (\lambda(s,p). \text{length } (\text{list } {}^s p \ {}^s next))])$$

$$\text{apply vcg}$$

$$\text{apply (fastforce simp add: List-list)}$$

$$\text{done}$$

**lemma** (in *append-impl*)  
**shows**  
 $\forall \sigma \ Ps \ Qs. \Gamma \vdash_t$   
 $\{\sigma. \text{List } 'p \ 'next \ Ps \wedge \text{List } 'q \ 'next \ Qs \wedge \text{set } Ps \cap \text{set } Qs = \{\} \}$   
 $'p ::= \text{PROC } \text{append}('p, 'q)$   
 $\{\text{List } 'p \ 'next \ (Ps @ Qs) \wedge (\forall x. x \notin \text{set } Ps \longrightarrow 'next \ x = \sigma_{next} \ x)\}$   
**apply** (*hoare-rule HoareTotal.ProcRec1*  
[**where**  $r = \text{measure } (\lambda(s,p). \text{length } (\text{list } ^s p \ ^s next))$ ])  
**apply** *vcg*  
**apply** (*fastforce simp add: List-list*)  
**done**

**lemma** (in *append-impl*)  
**shows**  
*append-spec*:  
 $\forall \sigma. \Gamma \vdash_t (\{\sigma\} \cap \{\text{islist } 'p \ 'next\}) \ 'p ::= \text{PROC } \text{append}('p, 'q)$   
 $\{\forall Ps \ Qs. \text{List } ^\sigma p \ ^\sigma next \ Ps \wedge \text{List } ^\sigma q \ ^\sigma next \ Qs \wedge \text{set } Ps \cap \text{set } Qs = \{\}$   
 $\longrightarrow$   
 $\text{List } 'p \ 'next \ (Ps @ Qs) \wedge (\forall x. x \notin \text{set } Ps \longrightarrow 'next \ x = \sigma_{next} \ x)\}$   
**apply** (*hoare-rule HoareTotal.ProcRec1*  
[**where**  $r = \text{measure } (\lambda(s,p). \text{length } (\text{list } ^s p \ ^s next))$ ])  
**apply** *vcg*  
**apply** *fastforce*  
**done**

**lemma**  $\Gamma \vdash \{\text{List } 'p \ 'next \ Ps\}$   
 $'q ::= \text{Null};;$   
 $\text{WHILE } 'p \neq \text{Null } \text{INV } \{\exists Ps' \ Qs'. \text{List } 'p \ 'next \ Ps' \wedge \text{List } 'q \ 'next \ Qs' \wedge$   
 $\text{set } Ps' \cap \text{set } Qs' = \{\} \wedge$   
 $\text{rev } Ps' @ Qs' = \text{rev } Ps\}$   
  
 $\text{DO}$   
 $'r ::= 'p;; 'p ::= 'p \rightarrow 'next;;$   
 $'r \rightarrow 'next ::= 'q;; 'q ::= 'r$   
 $\text{OD};;$   
 $'p ::= 'q$   
 $\{\text{List } 'p \ 'next \ (\text{rev } Ps)\}$   
**apply** *vcg*  
**apply** *clarsimp*  
**apply** *clarsimp*  
**apply** *force*  
**apply** *simp*  
**done**

**lemma** *conjI2*:  $\llbracket Q; Q \implies P \rrbracket \implies P \wedge Q$   
**by** *blast*

**procedures** *Rev(p|p)* =  
 $'q ::= \text{Null};;$   
 $\text{WHILE } 'p \neq \text{Null}$

```

DO
  'r := 'p;; {'p ≠ Null}⟶ 'p := 'p→'next;;
  {'r ≠ Null}⟶ 'r→'next := 'q;; 'q := 'r
OD;;
'p := 'q
Rev-spec:
∀ Ps. Γ ⊢t {'List 'p 'next Ps} 'p := PROC Rev('p) {'List 'p 'next (rev Ps)}
Rev-modifies:
∀ σ. Γ ⊢ /UNIV {σ} 'p := PROC Rev('p) {t. t may-only-modify-globals σ in
[next]}

```

We only need partial correctness of modifies clause!

```

lemma upd-hd-next:
  assumes p-ps: List p next (p#ps)
  shows List (next p) (next(p := q)) ps
proof -
  from p-ps
  have p ∉ set ps
  by auto
  with p-ps show ?thesis
  by simp
qed

```

lemma (in Rev-impl) shows

```

Rev-spec:
∀ Ps. Γ ⊢t {'List 'p 'next Ps} 'p := PROC Rev('p) {'List 'p 'next (rev Ps)}
apply (hoare-rule HoareTotal.ProcNoRec1)
apply (hoare-rule anno =
  'q := Null;;
  WHILE 'p ≠ Null INV {'∃ Ps' Qs'. List 'p 'next Ps' ∧ List 'q 'next Qs' ∧
    set Ps' ∩ set Qs' = {} ∧
    rev Ps' @ Qs' = rev Ps}
  VAR MEASURE (length (list 'p 'next) )
  DO
    'r := 'p;; {'p ≠ Null}⟶ 'p := 'p→'next;;
    {'r ≠ Null}⟶ 'r→'next := 'q;; 'q := 'r
  OD;;
  'p := 'q in HoareTotal.annotateI)
apply vcg
apply clarsimp
apply clarsimp
apply (rule conjI2)
apply force
apply clarsimp
apply (subgoal-tac List p next (p#ps))
prefer 2 apply simp
apply (frule-tac q=q in upd-hd-next)
apply (simp only: List-list)
apply simp

```

**apply** *simp*  
**done**

**lemma** (in *Rev-impl*) **shows**

*Rev-modifies:*  
 $\forall \sigma. \Gamma \vdash_{UNIV} \{\sigma\} \text{'p} ::= \text{PROC } \text{Rev}(\text{'p}) \{t. t \text{ may-only-modify-globals } \sigma \text{ in } [\text{next}]\}$   
**apply** (*hoare-rule HoarePartial.ProcNoRec1*)  
**apply** (*vcg spec=modifies*)  
**done**

**lemma**  $\Gamma \vdash_t \{\text{List 'p 'next Ps}\}$

$\text{'q} ::= \text{Null};;$   
 $\text{WHILE 'p} \neq \text{Null INV } \{\exists \text{Ps' Qs'}. \text{List 'p 'next Ps}' \wedge \text{List 'q 'next Qs}' \wedge$   
 $\text{set Ps}' \cap \text{set Qs}' = \{\} \wedge$   
 $\text{rev Ps}' @ \text{Qs}' = \text{rev Ps}\}$   
**VAR MEASURE** (*length (list 'p 'next)*)  
**DO**  
 $\text{'r} ::= \text{'p};; \text{'p} ::= \text{'p} \rightarrow \text{'next};;$   
 $\text{'r} \rightarrow \text{'next} ::= \text{'q};; \text{'q} ::= \text{'r}$   
**OD**;;  
 $\text{'p} ::= \text{'q}$   
 $\{\text{List 'p 'next (rev Ps)}\}$   
**apply** *vcg*  
**apply** *clarsimp*  
**apply** *clarsimp*  
**apply** (*rule conjI2*)  
**apply** *force*  
**apply** *clarsimp*  
**apply** (*subgoal-tac List p next (p#ps)*)  
**prefer** 2 **apply** *simp*  
**apply** (*frule-tac q=q in upd-hd-next*)  
**apply** (*simp only: List-list*)  
**apply** *simp*  
**apply** *simp*  
**done**

**procedures**

*pedal(N,M) = IF 0 < 'N THEN*  
 $\text{IF } 0 < \text{'M THEN CALL } \text{coast}(\text{'N} - 1, \text{'M} - 1) \text{ FI};;$   
 $\text{CALL } \text{pedal}(\text{'N} - 1, \text{'M})$   
*FI*

**and**

$$\text{coast}(N, M) = \text{CALL } \text{pedal}('N, 'M);;$$

$$\text{IF } 0 < 'M \text{ THEN CALL } \text{coast}('N, 'M - 1) \text{ FI}$$

**ML**  $\langle \text{ML-Thms.bind-thm } (\text{HoareTotal-ProcRec2}, \text{Hoare.gen-proc-rec } @\{\text{context}\} \text{Hoare.Total } 2) \rangle$

**lemma** (in *pedal-coast-clique*)  
**shows**  $(\Gamma \vdash_t \{ \text{True} \} \text{PROC } \text{pedal}('N, 'M) \{ \text{True} \}) \wedge$   
 $(\Gamma \vdash_t \{ \text{True} \} \text{PROC } \text{coast}('N, 'M) \{ \text{True} \})$   
**apply** (*hoare-rule HoareTotal-ProcRec2*  
[**where**  $?r = \text{inv-image } (\text{measure } (\lambda m. m) < *lex* >$   
 $\text{measure } (\lambda p. \text{if } p = \text{coast-}'\text{proc then } 1 \text{ else } 0))$   
 $(\lambda(s, p). (^sN + ^sM, p))$ ])  
**apply** *simp-all*  
**apply** *vcg*  
**apply** *simp*  
**apply** *vcg*  
**apply** *simp*  
**done**

**lemma** (in *pedal-coast-clique*)  
**shows**  $(\Gamma \vdash_t \{ \text{True} \} \text{PROC } \text{pedal}('N, 'M) \{ \text{True} \}) \wedge$   
 $(\Gamma \vdash_t \{ \text{True} \} \text{PROC } \text{coast}('N, 'M) \{ \text{True} \})$   
**apply** (*hoare-rule HoareTotal-ProcRec2*  
[**where**  $?r = \text{inv-image } (\text{measure } (\lambda m. m) < *lex* >$   
 $\text{measure } (\lambda p. \text{if } p = \text{coast-}'\text{proc then } 1 \text{ else } 0))$   
 $(\lambda(s, p). (^sN + ^sM, p))$ ])  
**apply** *simp-all*  
**apply** *vcg*  
**apply** *simp*  
**apply** *vcg*  
**apply** *simp*  
**done**

**lemma** (in *pedal-coast-clique*)  
**shows**  $(\Gamma \vdash_t \{ \text{True} \} \text{PROC } \text{pedal}('N, 'M) \{ \text{True} \}) \wedge$   
 $(\Gamma \vdash_t \{ \text{True} \} \text{PROC } \text{coast}('N, 'M) \{ \text{True} \})$   
**apply** (*hoare-rule HoareTotal-ProcRec2*  
[**where**  $?r = \text{measure } (\lambda(s, p). ^sN + ^sM + (\text{if } p = \text{coast-}'\text{proc then } 1 \text{ else } 0))$ ])  
**apply** *simp-all*  
**apply** *vcg*  
**apply** *simp*  
**apply** *arith*  
**apply** *vcg*

```

apply simp
done

```

```

lemma (in pedal-coast-clique)
shows ( $\Gamma \vdash_t \{ \text{True} \}$  PROC pedal('N, 'M) { True })  $\wedge$ 
        ( $\Gamma \vdash_t \{ \text{True} \}$  PROC coast('N, 'M) { True })
apply (hoare-rule HoareTotal-ProcRec2
        [ where ?r = ( $\lambda(s,p). {}^sN$ ) <*mlex*> ( $\lambda(s,p). {}^sM$ ) <*mlex*>
          measure ( $\lambda(s,p). \text{if } p = \text{coast-}'\text{proc then } 1 \text{ else } 0$ ) ])
apply simp-all
apply vcg
apply simp
apply vcg
apply simp
done

```

```

lemma (in pedal-coast-clique)
shows ( $\Gamma \vdash_t \{ \text{True} \}$  PROC pedal('N, 'M) { True })  $\wedge$ 
        ( $\Gamma \vdash_t \{ \text{True} \}$  PROC coast('N, 'M) { True })
apply (hoare-rule HoareTotal-ProcRec2
        [ where ?r = measure ( $\lambda s. {}^sN + {}^sM$ ) <*lex*>
          measure ( $\lambda p. \text{if } p = \text{coast-}'\text{proc then } 1 \text{ else } 0$ ) ])
apply simp-all
apply vcg
apply simp
apply vcg
apply simp
done

```

```

end

```

## 20 Example: Quicksort on Heap Lists

```

theory Quicksort
imports ../Vcg ../HeapList HOL-Library.Multiset
begin

```

```

record globals-heap =
  next-' :: ref  $\Rightarrow$  ref
  cont-' :: ref  $\Rightarrow$  nat

```

```

record 'g vars = 'g state +
  p-' :: ref
  q-' :: ref
  le-' :: ref
  gt-' :: ref

```



$hd-' :: ref$   
 $tl-' :: ref$

**procedures**

$append(p, q | p) =$   
 $IF \ 'p = Null \ THEN \ 'p ::= 'q \ ELSE \ 'p \rightarrow 'next ::= CALL \ append('p \rightarrow 'next, 'q)$   
 $FI$

*append-spec:*

$\forall \sigma \ Ps \ Qs.$   
 $\Gamma \vdash \{ \sigma. List \ 'p \ 'next \ Ps \wedge List \ 'q \ 'next \ Qs \wedge set \ Ps \cap set \ Qs = \{ \} \}$   
 $\ 'p ::= PROC \ append('p, 'q)$   
 $\{ List \ 'p \ 'next \ (Ps @ Qs) \wedge (\forall x. x \notin set \ Ps \longrightarrow 'next \ x = \sigma_{next} \ x) \}$

*append-modifies:*

$\forall \sigma. \Gamma \vdash \{ \sigma \} \ 'p ::= PROC \ append('p, 'q) \{ t. t \text{ may-only-modify-globals } \sigma \text{ in } [next] \}$

**lemma** (in *append-impl*) *append-modifies:*

**shows**

$\forall \sigma. \Gamma \vdash \{ \sigma \} \ 'p ::= PROC \ append('p, 'q) \{ t. t \text{ may-only-modify-globals } \sigma \text{ in } [next] \}$

**apply** (*hoare-rule HoarePartial.ProcRec1*)

**apply** (*vcg spec=modifies*)

**done**

**lemma** (in *append-impl*) *append-spec:*

**shows**  $\forall \sigma \ Ps \ Qs. \Gamma \vdash$

$\{ \sigma. List \ 'p \ 'next \ Ps \wedge List \ 'q \ 'next \ Qs \wedge set \ Ps \cap set \ Qs = \{ \} \}$

$\ 'p ::= PROC \ append('p, 'q)$

$\{ List \ 'p \ 'next \ (Ps @ Qs) \wedge (\forall x. x \notin set \ Ps \longrightarrow 'next \ x = \sigma_{next} \ x) \}$

**apply** (*hoare-rule HoarePartial.ProcRec1*)

**apply** *vcg*

**apply** *fastforce*

**done**

**primrec** *sorted:: ('a  $\Rightarrow$  'a  $\Rightarrow$  bool)  $\Rightarrow$  'a list  $\Rightarrow$  bool*

**where**

*sorted le [] = True |*

*sorted le (x#xs) = (( $\forall y \in set \ xs. le \ x \ y$ )  $\wedge$  sorted le xs)*

**lemma** *sorted-append[simp]:*

$sorted \ le \ (xs @ ys) = (sorted \ le \ xs \wedge sorted \ le \ ys \wedge$

$(\forall x \in set \ xs. \forall y \in set \ ys. le \ x \ y))$

**by** (*induct xs, auto*)

**procedures** *quickSort(p|p) =*

*IF 'p = Null THEN SKIP*

*ELSE 'tl ::= 'p  $\rightarrow$  'next;;*

```

'le ::= Null;;
'gt ::= Null;;
WHILE 'tl ≠ Null DO
  'hd ::= 'tl;;
  'tl ::= 'tl → 'next;;
  IF 'hd → 'cont ≤ 'p → 'cont
  THEN 'hd → 'next ::= 'le;;
    'le ::= 'hd
  ELSE 'hd → 'next ::= 'gt;;
    'gt ::= 'hd
  FI
OD;;
'le ::= CALL quickSort('le);;
'gt ::= CALL quickSort('gt);;
'p → 'next ::= 'gt;;
'le ::= CALL append('le, 'p);;
'p ::= 'le
FI

```

*quickSort-spec:*

$$\forall \sigma \text{ Ps. } \Gamma \vdash \{\sigma. \text{List } 'p \text{ 'next Ps}\} 'p ::= \text{PROC quickSort}('p)$$

$$\{\{\exists \text{sortedPs. List } 'p \text{ 'next sortedPs} \wedge$$

$$\text{sorted } (\leq) (\text{map } \sigma_{\text{cont}} \text{ sortedPs}) \wedge$$

$$\text{mset Ps} = \text{mset sortedPs}\} \wedge$$

$$(\forall x. x \notin \text{set Ps} \longrightarrow \text{'next } x = \sigma_{\text{next}} x)\}$$

*quickSort-modifies:*

$$\forall \sigma. \Gamma \vdash \{\sigma\} 'p ::= \text{PROC quickSort}('p) \{t. t \text{ may-only-modify-globals } \sigma \text{ in } [\text{next}]\}$$

**lemma** (in *quickSort-impl*) *quickSort-modifies:*

**shows**

$$\forall \sigma. \Gamma \vdash \{\sigma\} 'p ::= \text{PROC quickSort}('p) \{t. t \text{ may-only-modify-globals } \sigma \text{ in } [\text{next}]\}$$

**apply** (*hoare-rule HoarePartial.ProcRec1*)

**apply** (*vcg spec=modifies*)

**done**

**lemma** (in *quickSort-impl*) *quickSort-spec:*

**shows**

$$\forall \sigma \text{ Ps. } \Gamma \vdash \{\sigma. \text{List } 'p \text{ 'next Ps}\}$$

$$'p ::= \text{PROC quickSort}('p)$$

$$\{\{\exists \text{sortedPs. List } 'p \text{ 'next sortedPs} \wedge$$

$$\text{sorted } (\leq) (\text{map } \sigma_{\text{cont}} \text{ sortedPs}) \wedge$$

$$\text{mset Ps} = \text{mset sortedPs}\} \wedge$$

$$(\forall x. x \notin \text{set Ps} \longrightarrow \text{'next } x = \sigma_{\text{next}} x)\}$$

**apply** (*hoare-rule HoarePartial.ProcRec1*)

**apply** (*hoare-rule anno =*

*IF 'p=Null THEN SKIP*

*ELSE 'tl ::= 'p → 'next;;*

```

'le ::= Null;;
'gt ::= Null;;
WHILE 'tl ≠ Null
INV { (∃ les grs tls. List 'le 'next les ∧ List 'gt 'next grs ∧
      List 'tl 'next tls ∧
      mset Ps = mset ('p#tls@les@grs) ∧
      distinct('p#tls@les@grs) ∧
      (∀ x ∈ set les. x → 'cont ≤ 'p → 'cont) ∧
      (∀ x ∈ set grs. 'p → 'cont < x → 'cont)) ∧
      'p =σp ∧
      'cont =σcont ∧
      List σp σnext Ps ∧
      (∀ x. x ∉ set Ps → 'next x = σnext x)}
DO
  'hd ::= 'tl;;
  'tl ::= 'tl → 'next;;
  IF 'hd → 'cont ≤ 'p → 'cont
  THEN 'hd → 'next ::= 'le;;
    'le ::= 'hd
  ELSE 'hd → 'next ::= 'gt;;
    'gt ::= 'hd
  FI
OD;;
'le ::= CALL quickSort('le);;
'gt ::= CALL quickSort('gt);;
'p → 'next ::= 'gt;;
'le ::= CALL append('le, 'p);;
'p ::= 'le
FI in HoarePartial.annotateI
apply vcg
  apply fastforce
  apply clarsimp
  apply (rule conjI)
  apply clarify
  apply (rule conjI)
  apply (rule-tac x=tl#les in exI)
  apply simp
  apply (rule-tac x=grs in exI)
  apply simp
  apply (rule-tac x=ps in exI)
  apply simp
  apply (metis insertCI set-mset-add-mset-insert set-mset-mset)
  apply clarify
  apply (rule conjI)
  apply (rule-tac x=les in exI)
  apply simp
  apply (rule-tac x=tl#grs in exI)
  apply simp
  apply (rule-tac x=ps in exI)

```

```

  apply simp
  apply (metis insertCI set-mset-add-mset-insert set-mset-mset)
  apply clarsimp
  apply (rule-tac ?x=grs in exI)
  apply (rule conjI)
  apply (erule heap-eq-ListI1)
  apply clarify
  apply (erule-tac x=x in allE) back
  apply blast
  apply clarsimp
  apply (rule-tac x=sortedPs in exI)
  apply (rule conjI)
  apply (erule heap-eq-ListI1)
  apply (clarsimp)
  apply (erule-tac x=x in allE) back back
  apply (metis IntI empty-iff set-mset-mset)
  apply (rule-tac x=p#sortedPsa in exI)
  apply (rule conjI)
  apply (metis List-cons List-updateI Null-notin-List fun-upd-same insert-iff set-mset-add-mset-insert
set-mset-mset)
  apply (rule conjI)
  apply (metis disjoint-iff mset-eq-setD set-ConsD)
  apply clarsimp
  apply (rule conjI)
  apply (metis less-or-eq-imp-le mset-eq-setD)
  apply (rule conjI)
  apply (metis leD less-le-trans mset-eq-setD nat-le-linear)
  apply clarsimp
  apply (erule-tac x=x in allE)+
  apply (metis Un-iff insert-iff list.set(2) mset.simps(2) mset-append set-append
set-mset-mset)
done

```

end

```

theory XVcg
imports Vcg

```

```

begin

```

We introduce a syntactic variant of the let-expression so that we can safely unfold it during verification condition generation. With the new theorem attribute *vcg-simp* we can declare equalities to be used by the verification condition generator, while simplifying assertions.

```

syntax

```

```

-Let' :: [letbinds, basicblock] => basicblock
  (⟨⟨notation=⟨mixfix LET expression⟩⟩LET (-)/ IN (-)⟩ 23)

```

**syntax-consts**

*-Let'* == *Let'*

**translations**

*-Let' (-binds b bs) e* == *-Let' b (-Let' bs e)*

*-Let' (-bind x a) e* == *CONST Let' a (%x. e)*

**lemma** *Let'-unfold* [*vcg-simp*]: *Let' x f = f x*

**by** (*simp add: Let'-def Let-def*)

**lemma** *Let'-split-conv* [*vcg-simp*]:

*(Let' x (λp. (case-prod (f p) (g p)))) =*

*(Let' x (λp. (f p) (fst (g p)) (snd (g p))))*

**by** (*simp add: split-def*)

**end**

## 21 Examples for Parallel Assignments

**theory** *XVcgEx*

**imports** *../XVcg*

**begin**

**record** *globals* =

*G-':nat*

*H-':nat*

**record** *'g vars* = *'g state* +

*A-': nat*

*B-': nat*

*C-': nat*

*I-': nat*

*M-': nat*

*N-': nat*

*R-': nat*

*S-': nat*

*Arr-': nat list*

*Abr-': string*

**term** *BASIC*

*'A := x,*

*'B := y*

*END*

**term** *BASIC*

*'G := 'H,*

*'H := 'G*

*END*

**term** *BASIC*  
   $LET (x,y) = ('A,b);$   
     $z = 'B$   
   $IN 'A ::= x,$   
     $'G ::= 'A + y + z$   
*END*

**lemma**  $\Gamma \vdash \{\!| 'A = 0 |\!\}$   
   $\{\!| 'A < 0 |\!\} \mapsto BASIC$   
   $LET (a,b,c) = foo 'A$   
   $IN$   
     $'A ::= a,$   
     $'B ::= b,$   
     $'C ::= c$   
*END*  
 $\{\!| 'A = x \wedge 'B = y \wedge 'C = c |\!\}$

**apply** *vcg*

**oops**

**lemma**  $\Gamma \vdash \{\!| 'A = 0 |\!\}$   
   $\{\!| 'A < 0 |\!\} \mapsto BASIC$   
   $LET (a,b,c) = foo 'A$   
   $IN$   
     $'A ::= a,$   
     $'G ::= b + 'B,$   
     $'H ::= c$   
*END*  
 $\{\!| 'A = x \wedge 'G = y \wedge 'H = c |\!\}$

**apply** *vcg*

**oops**

**definition**  $foo :: nat \Rightarrow (nat \times nat \times nat)$   
  **where**  $foo\ n = (n,n+1,n+2)$

**lemma**  $\Gamma \vdash \{\!| 'A = 0 |\!\}$   
   $\{\!| 'A < 0 |\!\} \mapsto BASIC$   
   $LET (a,b,c) = foo 'A$   
   $IN$   
     $'A ::= a,$   
     $'G ::= b + 'B,$   
     $'H ::= c$   
*END*  
 $\{\!| 'A = x \wedge 'G = y \wedge 'H = c |\!\}$

**apply** (*vcg add: foo-def snd-conv fst-conv*)

**oops**

end

## 22 Examples for Procedures as Parameters

theory *ProcParEx* imports *../Vcg* begin

**lemma** *DynProcProcPar'*:  
assumes *adapt*:  $P \subseteq \{s. p\ s = q \wedge$   
     $(\exists Z. \text{init } s \in P' Z \wedge$   
         $(\forall t \in Q' Z. \text{return } s\ t \in R\ s\ t) \wedge$   
         $(\forall t \in A' Z. \text{return } s\ t \in A))\}$   
assumes *result*:  $\forall s\ t. \Gamma, \Theta \vdash_{/F} (R\ s\ t)\ \text{result } s\ t\ Q, A$   
assumes *q*:  $\forall Z. \Gamma, \Theta \vdash_{/F} (P' Z)\ \text{Call } q\ (Q' Z), (A' Z)$   
shows  $\Gamma, \Theta \vdash_{/F} P\ \text{dynCall } \text{init } p\ \text{return } \text{result } Q, A$   
apply (*rule* *HoarePartial.DynProcProcPar* [*OF* - *result* *q*])  
apply (*insert* *adapt*)  
apply *fast*  
done

**lemma** *conseq-exploit-pre'*:  
     $\llbracket \forall s \in S. \Gamma, \Theta \vdash (\{s\} \cap P)\ c\ Q, A \rrbracket$   
     $\implies$   
     $\Gamma, \Theta \vdash (P \cap S)\ c\ Q, A$   
apply (*rule* *HoarePartialDef.Conseq*)  
apply *clarify*  
by (*metis* *IntI* *insertI1* *subset-refl*)

**lemma** *conseq-exploit-pre''*:  
     $\llbracket \forall Z. \forall s \in S\ Z. \Gamma, \Theta \vdash (\{s\} \cap P\ Z)\ c\ (Q\ Z), (A\ Z) \rrbracket$   
     $\implies$   
     $\forall Z. \Gamma, \Theta \vdash (P\ Z \cap S\ Z)\ c\ (Q\ Z), (A\ Z)$   
apply (*rule* *allI*)  
apply (*rule* *conseq-exploit-pre'*)  
apply *blast*  
done

**lemma** *conseq-exploit-pre'''*:  
     $\llbracket \forall s \in S. \forall Z. \Gamma, \Theta \vdash (\{s\} \cap P\ Z)\ c\ (Q\ Z), (A\ Z) \rrbracket$   
     $\implies$   
     $\forall Z. \Gamma, \Theta \vdash (P\ Z \cap S)\ c\ (Q\ Z), (A\ Z)$   
apply (*rule* *allI*)  
apply (*rule* *conseq-exploit-pre'*)  
apply *blast*  
done

```

record 'g vars = 'g state +
  compare-' :: string
  n-' :: nat
  m-' :: nat
  b-' :: bool
  k-' :: nat

```

```

procedures compare(n,m|b) = NoBody
print-locale! compare-signature

```

```

context compare-signature
begin
declare [[hoare-use-call-tr' = false]]
term 'b ::= CALL compare('n,'m)
term 'b ::= DYNCALL 'compare('n,'m)
declare [[hoare-use-call-tr' = true]]
term 'b ::= DYNCALL 'compare('n,'m)
end

```

```

procedures
  LEQ (n,m | b) = 'b ::= 'n ≤ 'm
  LEQ-spec: ∀σ. Γ⊢ {σ} PROC LEQ('n,'m,'b) { 'b = (σn ≤ σm) }
  LEQ-modifies: ∀σ. Γ⊢ {σ} PROC LEQ('n,'m,'b) { t. t may-only-modify-globals σ
in [] }

```

```

definition mx:: ('a ⇒ 'a ⇒ bool) ⇒ 'a ⇒ 'a ⇒ 'a
  where mx leq a b = (if leq a b then a else b)

```

```

procedures
  Max (compare, n, m | k) =
    'b ::= DYNCALL 'compare('n,'m);;
    IF 'b THEN 'k ::= 'n ELSE 'k ::= 'm FI

  Max-spec: ∧leq. ∀σ. Γ⊢
    ({σ} ∩ {s. (∀τ. Γ⊢ {τ} 'b ::= PROC scompare('n,'m) { 'b = (leq τn τm) } ) ∧
      (∀τ. Γ⊢ {τ} 'b ::= PROC scompare('n,'m) { t. t may-only-modify-globals
τ in [] } ) } )
    PROC Max('compare,'n,'m,'k)
    { 'k = mx leq σn σm }

```



```

lemma (in Max-impl) Max-spec1:
shows
 $\forall \sigma \text{ leq. } \Gamma \vdash$ 
  ( $\{\sigma\} \cap \{\{(\forall \tau. \Gamma \vdash \{\tau\} \text{ 'b := PROC 'compare('n, 'm) \{b = (leq \tau_n \tau_m)\}) \wedge$ 
    ( $\forall \tau. \Gamma \vdash \{\tau\} \text{ 'b := PROC 'compare('n, 'm) \{t. t \text{ may-only-modify-globals } \tau$ 
    in  $\{\}\})\}$ )
    'k := PROC Max('compare, 'n, 'm)
     $\{\{k = mx \text{ leq } \sigma_n \sigma_m\}$ 
apply (hoare-rule HoarePartial.ProcNoRec1)
apply (intro allI)
apply (rule conseq-exploit-pre')
apply (rule)
apply clarify
proof –
  fix  $\sigma :: ('a, 'b) \text{ vars-scheme and } s :: ('a, 'b) \text{ vars-scheme and leq}$ 
  assume compare-spec:
     $\forall \tau. \Gamma \vdash \{\tau\} \text{ 'b := PROC } s\text{compare}('n, 'm) \{\{b = leq \tau_n \tau_m\}$ 

  assume compare-modifies:
     $\forall \tau. \Gamma \vdash \{\tau\} \text{ 'b := PROC } s\text{compare}('n, 'm)$ 
     $\{t. t \text{ may-only-modify-globals } \tau \text{ in } \{\}\}$ 

  show  $\Gamma \vdash (\{s\} \cap \{\sigma\})$ 
    'b := DYNCALL 'compare ('n, 'm);;
    IF 'b THEN 'k := 'n ELSE 'k := 'm FI
     $\{\{k = mx \text{ leq } \sigma_n \sigma_m\}$ 
  apply vcg
  apply (clarsimp simp add: mx-def)
  done
qed

```

```

lemma (in Max-impl) Max-spec2:
shows
 $\forall \sigma \text{ leq. } \Gamma \vdash$ 
  ( $\{\sigma\} \cap \{\{(\forall \tau. \Gamma \vdash \{\tau\} \text{ 'b := PROC 'compare('n, 'm) \{b = (leq \tau_n \tau_m)\}) \wedge$ 
    ( $\forall \tau. \Gamma \vdash \{\tau\} \text{ 'b := PROC 'compare('n, 'm) \{t. t \text{ may-only-modify-globals } \tau$ 
    in  $\{\}\})\}$ )
    'k := PROC Max('compare, 'n, 'm)
     $\{\{k = mx \text{ leq } \sigma_n \sigma_m\}$ 
apply (hoare-rule HoarePartial.ProcNoRec1)
apply (intro allI)
apply (rule conseq-exploit-pre')
apply (rule)
apply clarify
apply vcg
apply (clarsimp simp add: mx-def)
done

```

**lemma** (in *Max-impl*) *Max-spec3*:

**shows**

```

∀ n m leq. Γ ⊢
  ( { 'n=n ∧ 'm=m } ∩
    { (∀ τ. Γ ⊢ {τ} 'b ::= PROC 'compare('n,'m) { 'b = (leq τn τm) } ) ∧
      (∀ τ. Γ ⊢ {τ} 'b ::= PROC 'compare('n,'m) { t. t may-only-modify-globals τ in
        [] } ) } )
    'k ::= PROC Max('compare,'n,'m)
    { 'k = mx leq n m }
apply (hoare-rule HoarePartial.ProcNoRec1)
apply (intro allI)
apply (rule conseq-exploit-pre')
apply (rule)
apply clarify
apply vcg
apply (clarsimp simp add: mx-def)
done

```

**lemma** (in *Max-impl*) *Max-spec4*:

**shows**

```

∀ n m leq. Γ ⊢
  ( { 'n=n ∧ 'm=m } ∩ { ∀ τ. Γ ⊢ {τ} 'b ::= PROC 'compare('n,'m) { 'b = (leq τn
    τm) } } )
    'k ::= PROC Max('compare,'n,'m)
    { 'k = mx leq n m }
apply (hoare-rule HoarePartial.ProcNoRec1)
apply (intro allI)
apply (rule conseq-exploit-pre')
apply (rule)
apply clarify
apply vcg
apply (clarsimp simp add: mx-def)
done

```

**locale** *Max-test* = *Max-spec* + *LEQ-spec* + *LEQ-modifies*

**lemma** (in *Max-test*)

**shows**

$\Gamma \vdash \{ \sigma \} 'k ::= \text{CALL Max}(\text{LEQ-}'proc, 'n, 'm) \{ 'k = mx (\leq) \sigma_n \sigma_m \}$

**proof** –

**note** *Max-spec* = *Max-spec* [ **where** *leq*=( $\leq$ ) ]

**show** ?thesis

**apply** vcg

**apply** (clarsimp)

**apply** (rule conjI)

**apply** (rule *LEQ-spec* [simplified])

**apply** (rule *LEQ-modifies* [simplified])

**done**

**qed**

```

lemma (in Max-impl) Max-spec5:
shows
 $\forall n\ m\ leq. \Gamma \vdash$ 
  ( $\{\{n=n \wedge m=m\} \cap \{\forall n'\ m'. \Gamma \vdash \{n=n' \wedge m=m'\} \} b := PROC\ compare('n, 'm) \{b = (leq\ n'\ m')\}\}$ )
   $\{k := PROC\ Max(compare, 'n, 'm) \{k = mx\ leq\ n\ m\}$ 
term  $\{\{s. s_n = n' \wedge s_m = m'\} = X\}$ 
apply (hoare-rule HoarePartial.ProcNoRec1)
apply (intro allI)
apply (rule conseq-exploit-pre')
apply (rule)
apply (clarify)
apply (vcg)
apply (clarsimp)
apply (clarsimp simp add: mx-def)
done

```

```

lemma (in LEQ-impl)
  LEQ-spec:  $\forall n\ m. \Gamma \vdash \{n=n \wedge m=m\} PROC\ LEQ('n, 'm, 'b) \{b = (n \leq m)\}$ 
apply (vcg)
done

```

**locale** *Max-test'* = *Max-impl* + *LEQ-impl*

```

lemma (in Max-test')
shows
 $\forall n\ m. \Gamma \vdash \{n=n \wedge m=m\} k := CALL\ Max(LEQ-'proc, 'n, 'm) \{k = mx\ (\leq) n\ m\}$ 
proof –
note Max-spec = Max-spec5
show ?thesis
apply (vcg)
apply (rule-tac x=( $\leq$ ) in exI)
apply (clarsimp)
apply (rule LEQ-spec [rule-format])
done
qed

```

**end**

## 23 Examples for Procedures as Parameters using Statespaces

```

theory ProcParExSP imports ../Vcg begin

```

**lemma** *DynProcProcPar'*:  
**assumes** *adapt*:  $P \subseteq \{s. p \ s = q \wedge$   
 $(\exists Z. \text{init } s \in P' \ Z \wedge$   
 $(\forall t \in Q' \ Z. \text{return } s \ t \in R \ s \ t) \wedge$   
 $(\forall t \in A' \ Z. \text{return } s \ t \in A))\}$   
**assumes** *result*:  $\forall s \ t. \Gamma, \Theta \vdash_{/F} (R \ s \ t) \ \text{result } s \ t \ Q, A$   
**assumes** *q*:  $\forall Z. \Gamma, \Theta \vdash_{/F} (P' \ Z) \ \text{Call } q \ (Q' \ Z), (A' \ Z)$   
**shows**  $\Gamma, \Theta \vdash_{/F} P \ \text{dynCall } \text{init } p \ \text{return } \text{result } Q, A$   
**apply** (*rule* *HoarePartial.DynProcProcPar* [*OF* - *result* *q*])  
**apply** (*insert* *adapt*)  
**apply** *fast*  
**done**

**lemma** *conseq-exploit-pre'*:  
 $\llbracket \forall s \in S. \Gamma, \Theta \vdash (\{s\} \cap P) \ c \ Q, A \rrbracket$   
 $\implies$   
 $\Gamma, \Theta \vdash (P \cap S) \ c \ Q, A$   
**apply** (*rule* *HoarePartialDef.Conseq*)  
**apply** *clarify*  
**by** (*metis* *IntI* *insertI1* *subset-refl*)

**lemma** *conseq-exploit-pre''*:  
 $\llbracket \forall Z. \forall s \in S \ Z. \Gamma, \Theta \vdash (\{s\} \cap P \ Z) \ c \ (Q \ Z), (A \ Z) \rrbracket$   
 $\implies$   
 $\forall Z. \Gamma, \Theta \vdash (P \ Z \cap S \ Z) \ c \ (Q \ Z), (A \ Z)$   
**apply** (*rule* *allI*)  
**apply** (*rule* *conseq-exploit-pre'*)  
**apply** *blast*  
**done**

**lemma** *conseq-exploit-pre'''*:  
 $\llbracket \forall s \in S. \forall Z. \Gamma, \Theta \vdash (\{s\} \cap P \ Z) \ c \ (Q \ Z), (A \ Z) \rrbracket$   
 $\implies$   
 $\forall Z. \Gamma, \Theta \vdash (P \ Z \cap S) \ c \ (Q \ Z), (A \ Z)$   
**apply** (*rule* *allI*)  
**apply** (*rule* *conseq-exploit-pre'*)  
**apply** *blast*  
**done**

**procedures** *compare*(*i::nat*,*j::nat*|*r::bool*) *NoBody*

**print-locale!** *compare-signature*

```

context compare-impl
begin
declare [[hoare-use-call-tr' = false]]
term 'r ::= CALL compare('i, 'j)
declare [[hoare-use-call-tr' = true]]
end

```

**procedures**

```

LEQ (i::nat, j::nat | r::bool) 'r ::= 'i ≤ 'j
LEQ-spec: ∀σ. Γ ⊢ {σ} PROC LEQ('i, 'j, 'r) {r = (σi ≤ σj)}

LEQ-modifies: ∀σ. Γ ⊢ {σ} PROC LEQ('i, 'j, 'r) {t. t may-only-modify-globals σ
in []}

```

**definition** *mx*:: ('a ⇒ 'a ⇒ *bool*) ⇒ 'a ⇒ 'a ⇒ 'a  
**where** *mx leq a b* = (*if leq a b then a else b*)

**procedures** (**imports** *compare-signature*)

```

Max (compare::string, n::nat, m::nat | k::nat)
where b::bool
in
  'b ::= DYNCALL 'compare('n, 'm);;
  IF 'b THEN 'k ::= 'n ELSE 'k ::= 'm FI

Max-spec: ∧leq. ∀σ. Γ ⊢
  ({σ} ∩ {s. (∀τ. Γ ⊢ {τ} 'r ::= PROC scompare('i, 'j) {r = (leq τi τj)})} ∧
  (∀τ. Γ ⊢ {τ} 'r ::= PROC scompare('i, 'j) {t. t may-only-modify-globals
τ in []}))}
  PROC Max('compare, 'n, 'm, 'k)
  {k = mx leq σn σm}

```

**context** *Max-spec*

```

begin
thm Max-spec
end

```

**context** *Max-impl*

```

begin
term 'b ::= DYNCALL 'compare('n, 'm)
declare [[hoare-use-call-tr' = false]]
term 'b ::= DYNCALL 'compare('n, 'm)
declare [[hoare-use-call-tr' = true]]
end

```

**lemma** (in *Max-impl*) *Max-spec1*:  
**shows**  
 $\forall \sigma \text{ leq. } \Gamma \vdash$   
 $(\{\sigma\} \cap \{\!(\forall \tau. \Gamma \vdash \{\tau\} \text{ 'r := PROC 'compare('i, 'j) \{!r = (leq }^{\tau_i} \tau_j)\}\} \wedge$   
 $(\forall \tau. \Gamma \vdash \{\tau\} \text{ 'r := PROC 'compare('i, 'j) \{t. t may-only-modify-globals } \tau \text{ in$   
 $\{\!\}\}\})$   
 $\text{ 'k := PROC Max('compare, 'n, 'm)}$   
 $\{\!k = mx \text{ leq } \sigma_n \sigma_m\}$   
**apply** (hoare-rule *HoarePartial.ProcNoRec1*)  
**apply** (intro allI)  
**apply** (rule conseq-exploit-pre')  
**apply** (rule)  
**apply** clarify  
**proof** –  
**fix** s:: ('a, 'b, 'c, 'd) stateSP-scheme **and** s:: ('a, 'b, 'c, 'd) stateSP-scheme **and**  
leq  
**assume** compare-spec:  
 $\forall \tau. \Gamma \vdash \{\tau\} \text{ 'r := PROC }^s \text{compare('i, 'j) \{!r = leq }^{\tau_i} \tau_j\}$   
  
**assume** compare-modifies:  
 $\forall \tau. \Gamma \vdash \{\tau\} \text{ 'r := PROC }^s \text{compare('i, 'j)$   
 $\{t. t \text{ may-only-modify-globals } \tau \text{ in } \{\!\}\}$   
  
**show**  $\Gamma \vdash (\{\sigma\} \cap \{\sigma\})$   
 $\text{ 'b := DYNCALL 'compare ('n, 'm);;$   
 $\text{ IF 'b THEN 'k := 'n ELSE 'k := 'm FI}$   
 $\{\!k = mx \text{ leq } \sigma_n \sigma_m\}$   
**apply** vcg  
**apply** (clarsimp simp add: mx-def)  
**done**  
**qed**

**lemma** (in *Max-impl*) *Max-spec2*:  
**shows**  
 $\forall \sigma \text{ leq. } \Gamma \vdash$   
 $(\{\sigma\} \cap \{\!(\forall \tau. \Gamma \vdash \{\tau\} \text{ 'r := PROC 'compare('i, 'j) \{!r = (leq }^{\tau_i} \tau_j)\}\} \wedge$   
 $(\forall \tau. \Gamma \vdash \{\tau\} \text{ 'r := PROC 'compare('i, 'j) \{t. t may-only-modify-globals } \tau \text{ in$   
 $\{\!\}\}\})$   
 $\text{ 'k := PROC Max('compare, 'n, 'm)}$   
 $\{\!k = mx \text{ leq } \sigma_n \sigma_m\}$   
**apply** (hoare-rule *HoarePartial.ProcNoRec1*)  
**apply** (intro allI)  
**apply** (rule conseq-exploit-pre')  
**apply** (rule)  
**apply** clarify  
**apply** vcg  
**apply** (clarsimp simp add: mx-def)

**done**

**lemma** (in *Max-impl*) *Max-spec3*:

**shows**

$\forall n\ m\ leq. \Gamma \vdash$   
     $(\{\{ 'n=n \wedge 'm=m \} \cap$   
         $\{\{ (\forall \tau. \Gamma \vdash \{\tau\} 'r ::= PROC\ 'compare(i, j) \{\{ 'r = (leq\ \tau_i\ \tau_j) \} \} \} \} \wedge$   
         $(\forall \tau. \Gamma \vdash \{\tau\} 'r ::= PROC\ 'compare(i, j) \{t. t\ may-only-modify-globals\ \tau\ in$   
     $\{\}\} \})$   
     $'k ::= PROC\ Max('compare, 'n, 'm)$   
     $\{\{ 'k = mx\ leq\ n\ m \}$   
**apply** (*hoare-rule HoarePartial.ProcNoRec1*)  
**apply** (*intro allI*)  
**apply** (*rule conseq-exploit-pre'*)  
**apply** (*rule*)  
**apply** (*clarify*)  
**apply** (*vcg*)  
**apply** (*clarsimp simp add: mx-def*)  
**done**

**lemma** (in *Max-impl*) *Max-spec4*:

**shows**

$\forall n\ m\ leq. \Gamma \vdash$   
     $(\{\{ 'n=n \wedge 'm=m \} \cap \{\{ \forall \tau. \Gamma \vdash \{\tau\} 'r ::= PROC\ 'compare(i, j) \{\{ 'r = (leq\ \tau_i\ \tau_j) \} \} \} \} \})$   
     $'k ::= PROC\ Max('compare, 'n, 'm)$   
     $\{\{ 'k = mx\ leq\ n\ m \}$   
**apply** (*hoare-rule HoarePartial.ProcNoRec1*)  
**apply** (*intro allI*)  
**apply** (*rule conseq-exploit-pre'*)  
**apply** (*rule*)  
**apply** (*clarify*)  
**apply** (*vcg*)  
**apply** (*clarsimp simp add: mx-def*)  
**done**

**print-locale** *Max-spec*

**locale** *Max-test* = *Max-spec* **where**

*i*-*'compare*-*'* = *i*-*'LEQ*-*'* **and**  
*j*-*'compare*-*'* = *j*-*'LEQ*-*'* **and**  
*r*-*'compare*-*'* = *r*-*'LEQ*-*'*  
+ *LEQ-spec* + *LEQ-modifies*

**lemma** (in *Max-test*)

**shows**

$\Gamma \vdash \{\sigma\} 'k ::= CALL\ Max(LEQ-'proc, 'n, 'm) \{\{ 'k = mx\ (\leq)\ \sigma_n\ \sigma_m \}$   
**proof** –

```

note Max-spec = Max-spec [where leq=( $\leq$ )]
show ?thesis
  apply vcg
  apply (clarsimp)
  apply (rule conjI)
  apply (rule LEQ-spec)
  apply (rule LEQ-modifies)
  done
qed

```

```

lemma (in Max-impl) Max-spec5:
shows
 $\forall n\ m\ leq.\ \Gamma \vdash$ 
  ( $\{\{n=n \wedge m=m\} \cap \{\{\forall n'\ m'.\ \Gamma \vdash \{i=n' \wedge j=m'\} \ r := PROC\ compare(i, j)\} \{r = (leq\ n'\ m')\}\}\}$ )
   $\{k := PROC\ Max(compare, n, m)\}$ 
   $\{k = mx\ leq\ n\ m\}$ 
apply (hoare-rule HoarePartial.ProcNoRec1)
apply (intro allI)
apply (rule conseq-exploit-pre')
apply (rule)
apply clarify
apply vcg
apply clarsimp
apply (clarsimp simp add: mx-def)
done

```

```

lemma (in LEQ-impl)
  LEQ-spec:  $\forall n\ m.\ \Gamma \vdash \{i=n \wedge j=m\}\ PROC\ LEQ(i, j, r)\ \{r = (n \leq m)\}$ 
  apply vcg
  apply simp
  done

```

```

print-locale Max-impl

```

```

locale Max-test' = Max-impl where
  i'-compare-' = i'-LEQ-' and
  j'-compare-' = j'-LEQ-' and
  r'-compare-' = r'-LEQ-'
  + LEQ-impl

```

```

lemma (in Max-test')
shows
 $\forall n\ m.\ \Gamma \vdash \{n=n \wedge m=m\}\ k := CALL\ Max(LEQ-'proc, n, m)\ \{k = mx\ (\leq)\ n\ m\}$ 

```



```

proof –
  note Max-spec = Max-spec5
  show ?thesis
    apply vcg
    apply (rule-tac x=(≤) in exI)
    apply clarsimp
    apply (rule LEQ-spec [rule-format])
    done
qed

end

```

## 24 Experiments with Closures

```

theory Closure
imports ../Hoare
begin

```

### definition

```

callClosure upd cl = Seq (Basic (upd (fst cl))) (Call (snd cl))

```

### definition

```

dynCallClosure init upd cl return c =
  DynCom ( $\lambda s. \text{call } (\text{upd } (\text{fst } (\text{cl } s)) \circ \text{init}) (\text{snd } (\text{cl } s)) \text{ return } c$ )

```

### lemma *dynCallClosure-sound*:

**assumes** *adapt*:

$$P \subseteq \{s. \exists P' Q' A'. \forall n. \Gamma, \Theta \models n: /_F P' (\text{callClosure } \text{upd } (\text{cl } s)) Q', A' \wedge$$

$$\text{init } s \in P' \wedge$$

$$(\forall t \in Q'. \text{return } s \ t \in R \ s \ t) \wedge$$

$$(\forall t \in A'. \text{return } s \ t \in A)\}$$

**assumes** *res*:  $\forall s \ t \ n. \Gamma, \Theta \models n: /_F (R \ s \ t) (c \ s \ t) Q, A$

**shows**

$$\Gamma, \Theta \models n: /_F P (\text{dynCallClosure } \text{init } \text{upd } \text{cl } \text{return } c) Q, A$$

**proof** (*rule cinvalidI*)

**fix** *s t*

**assume** *ctxt*:  $\forall (P, p, Q, A) \in \Theta. \Gamma \models n: /_F P \text{ Call } p \ Q, A$

**assume** *exec*:  $\Gamma \vdash \langle \text{dynCallClosure } \text{init } \text{upd } \text{cl } \text{return } c, \text{Normal } s \rangle = n \Rightarrow t$

**from** *execn.Basic* [**where**  $f = (\text{upd } (\text{fst } (\text{cl } s)))$  **and**  $s = (\text{init } s)$ ]

**have** *exec-upd*:  $\Gamma \vdash \langle \text{Basic } (\text{upd } (\text{fst } (\text{cl } s))), \text{Normal } (\text{init } s) \rangle = n \Rightarrow$

$$\text{Normal } (((\text{upd } (\text{fst } (\text{cl } s))) \circ \text{init}) \ s)$$

**by** *auto*

```

assume  $P: s \in P$ 
from  $P$  adapt obtain  $P' Q' A'$ 
  where
    valid:  $\forall n. \Gamma, \Theta \models n: /_F P' (\text{callClosure upd } (cl\ s)) Q', A'$  and
    init-P':  $\text{init } s \in P'$  and
     $R: (\forall t \in Q'. \text{return } s\ t \in R\ s\ t)$  and
     $A: (\forall t \in A'. \text{return } s\ t \in A)$ 
  by auto
assume  $t\text{-notin-}F: t \notin \text{Fault } F$ 
from exec [simplified dynCallClosure-def]
have exec-call:
   $\Gamma \vdash \langle \text{call } (\text{upd } (\text{fst } (cl\ s)) \circ \text{init}) (\text{snd } (cl\ s)) \text{return } c, \text{Normal } s \rangle = n \Rightarrow t$ 
  by cases
then
show  $t \in \text{Normal } Q \cup \text{Abrupt } A$ 
proof (cases rule: execn-call-Normal-elim)
  fix bdy m t'
  assume bdy:  $\Gamma (\text{snd } (cl\ s)) = \text{Some } bdy$ 
  assume exec-body:  $\Gamma \vdash \langle bdy, \text{Normal } ((\text{upd } (\text{fst } (cl\ s)) \circ \text{init})\ s) \rangle = m \Rightarrow \text{Normal } t'$ 
  assume exec-c:  $\Gamma \vdash \langle c\ s\ t', \text{Normal } (\text{return } s\ t') \rangle = \text{Suc } m \Rightarrow t$ 
  assume  $n: n = \text{Suc } m$ 
  have  $\Gamma \vdash \langle \text{Basic } \text{init}, \text{Normal } s \rangle = m \Rightarrow \text{Normal } (\text{init } s)$ 
  by (rule execn.Basic)
  from bdy exec-body
  have exec-callC:
   $\Gamma \vdash \langle \text{Call } (\text{snd } (cl\ s)), \text{Normal } ((\text{upd } (\text{fst } (cl\ s)) \circ \text{init})\ s) \rangle = \text{Suc } m \Rightarrow \text{Normal } t'$ 
  by (rule execn.Call)
  from execn.Seq [OF exec-upd [simplified n] exec-callC]
  have exec-closure:  $\Gamma \vdash \langle \text{callClosure upd } (cl\ s), \text{Normal } (\text{init } s) \rangle = n \Rightarrow \text{Normal } t'$ 
  by (simp add: callClosure-def n)
  from cvalidD [OF valid [rule-format] ctxt exec-closure init-P']
  have  $t' \in Q'$ 
  by auto
  with  $R$  have  $\text{return } s\ t' \in R\ s\ t'$ 
  by auto
  from cvalidD [OF res [rule-format] ctxt exec-c [simplified n] symmetric] this
   $t\text{-notin-}F$ 
  show ?thesis
  by auto
next
  fix bdy m t'
  assume bdy:  $\Gamma (\text{snd } (cl\ s)) = \text{Some } bdy$ 
  assume exec-body:  $\Gamma \vdash \langle bdy, \text{Normal } ((\text{upd } (\text{fst } (cl\ s)) \circ \text{init})\ s) \rangle = m \Rightarrow \text{Abrupt } t'$ 
  assume  $t: t = \text{Abrupt } (\text{return } s\ t')$ 
  assume  $n: n = \text{Suc } m$ 
  from bdy exec-body
  have exec-callC:
   $\Gamma \vdash \langle \text{Call } (\text{snd } (cl\ s)), \text{Normal } ((\text{upd } (\text{fst } (cl\ s)) \circ \text{init})\ s) \rangle = \text{Suc } m \Rightarrow \text{Abrupt } t'$ 

```

```

    by (rule execn.Call)
  from execn.Seq [OF exec-upd [simplified n] exec-callC]
  have exec-closure:  $\Gamma \vdash \langle \text{callClosure upd (cl s), Normal (init s)} \rangle =n \Rightarrow \text{Abrupt } t'$ 
    by (simp add: callClosure-def n)

  from cinvalidD [OF valid [rule-format] ctxt exec-closure init-P↑]
  have  $t' \in A'$ 
    by auto
  with A have return s  $t' \in A$ 
    by auto
  with t show ?thesis
    by auto
next
fix bdy m f
assume bdy:  $\Gamma (\text{snd (cl s)}) = \text{Some bdy}$ 
assume exec-body:  $\Gamma \vdash \langle \text{bdy, Normal ((upd (fst (cl s)) \circ \text{init}) s)} \rangle =m \Rightarrow \text{Fault } f$ 
assume t:  $t = \text{Fault } f$ 
assume n:  $n = \text{Suc } m$ 
from bdy exec-body
have exec-callC:
   $\Gamma \vdash \langle \text{Call (snd (cl s)), Normal ((upd (fst (cl s)) \circ \text{init}) s)} \rangle =\text{Suc } m \Rightarrow \text{Fault } f$ 
  by (rule execn.Call)
from execn.Seq [OF exec-upd [simplified n] exec-callC]
have exec-closure:  $\Gamma \vdash \langle \text{callClosure upd (cl s), Normal (init s)} \rangle =n \Rightarrow \text{Fault } f$ 
  by (simp add: callClosure-def n)
from cinvalidD [OF valid [rule-format] ctxt exec-closure init-P↑] t-notin-F t
have False
  by auto
thus ?thesis ..
next
fix bdy m
assume bdy:  $\Gamma (\text{snd (cl s)}) = \text{Some bdy}$ 
assume exec-body:  $\Gamma \vdash \langle \text{bdy, Normal ((upd (fst (cl s)) \circ \text{init}) s)} \rangle =m \Rightarrow \text{Stuck}$ 
assume t:  $t = \text{Stuck}$ 
assume n:  $n = \text{Suc } m$ 
from execn.Basic [where f=(upd (fst (cl s))) and s=(init s)]
have exec-upd:  $\Gamma \vdash \langle \text{Basic (upd (fst (cl s))), Normal (init s)} \rangle =\text{Suc } m \Rightarrow$ 
   $\text{Normal (((upd (fst (cl s))) \circ \text{init}) s)}$ 
  by auto
from bdy exec-body
have exec-callC:
   $\Gamma \vdash \langle \text{Call (snd (cl s)), Normal ((upd (fst (cl s)) \circ \text{init}) s)} \rangle =\text{Suc } m \Rightarrow \text{Stuck}$ 
  by (rule execn.Call)
from execn.Seq [OF exec-upd [simplified n] exec-callC]
have exec-closure:  $\Gamma \vdash \langle \text{callClosure upd (cl s), Normal (init s)} \rangle =n \Rightarrow \text{Stuck}$ 
  by (simp add: callClosure-def n)
from cinvalidD [OF valid [rule-format] ctxt exec-closure init-P↑] t-notin-F t
have False
  by auto

```

```

thus ?thesis ..
next
fix m
assume no-bdy:  $\Gamma (snd (cl s)) = None$ 
assume t:  $t = Stuck$ 
assume n:  $n = Suc m$ 
from no-bdy
have exec-callC:
   $\Gamma \langle Call (snd (cl s)), Normal ((upd (fst (cl s)) \circ init) s) \rangle = Suc m \Rightarrow Stuck$ 
  by (rule execn.CallUndefined)
from execn.Seq [OF exec-upd [simplified n] exec-callC]
have exec-closure:  $\Gamma \langle callClosure upd (cl s), Normal (init s) \rangle = n \Rightarrow Stuck$ 
  by (simp add: callClosure-def n)
from cinvalidD [OF valid [rule-format] ctxt exec-closure init-P†] t-notin-F t
have False
  by auto
thus ?thesis ..
qed
qed

```

**lemma** dynCallClosure:

```

assumes adapt:  $P \subseteq \{s. \exists P' Q' A'. \Gamma, \Theta \vdash_{/F} P' (callClosure upd (cl s)) Q', A' \wedge$ 
   $init s \in P' \wedge$ 
   $(\forall t \in Q'. return s t \in R s t) \wedge$ 
   $(\forall t \in A'. return s t \in A)\}$ 
assumes res:  $\forall s t. \Gamma, \Theta \vdash_{/F} (R s t) (c s t) Q, A$ 
shows
 $\Gamma, \Theta \vdash_{/F} P (dynCallClosure init upd cl return c) Q, A$ 
apply (rule hoare-complete')
apply (rule allI)
apply (rule dynCallClosure-sound [where R=R])
using adapt
apply (blast intro: hoare-cinvalid)
using res
apply (blast intro: hoare-cinvalid)
done

```

**lemma** in-subsetD:  $\llbracket P \subseteq P'; x \in P \rrbracket \Longrightarrow x \in P'$   
**by** blast

**lemma** dynCallClosureFix:

```

assumes adapt:  $P \subseteq \{s. \exists Z. cl' = cl s \wedge$ 
   $init s \in P' Z \wedge$ 
   $(\forall t \in Q' Z. return s t \in R s t) \wedge$ 
   $(\forall t \in A' Z. return s t \in A)\}$ 
assumes res:  $\forall s t. \Gamma, \Theta \vdash_{/F} (R s t) (c s t) Q, A$ 
assumes spec:  $\forall Z. \Gamma, \Theta \vdash_{/F} (P' Z) (callClosure upd cl') (Q' Z), (A' Z)$ 

```

**shows**  
 $\Gamma, \Theta \vdash_{/F} P$  (*dynCallClosure* *init upd cl return c*)  $Q, A$   
**apply** (*rule dynCallClosure* [*OF - res*])  
**using** *adapt spec*  
**apply** *clarsimp*  
**apply** (*drule* (1) *in-subsetD*)  
**apply** *clarsimp*  
**apply** (*rule-tac*  $x=P' Z$  **in** *exI*)  
**apply** (*rule-tac*  $x=Q' Z$  **in** *exI*)  
**apply** (*rule-tac*  $x=A' Z$  **in** *exI*)  
**apply** *blast*  
**done**

**lemma** *conseq-extract-pre*:  

$$\llbracket \forall s \in P. \Gamma, \Theta \vdash_{/F} (\{s\}) c Q, A \rrbracket$$

$$\implies$$

$$\Gamma, \Theta \vdash_{/F} P c Q, A$$
**apply** (*rule hoarep.Conseq*)  
**apply** *clarify*  
**apply** (*rule-tac*  $x=\{s\}$  **in** *exI*)  
**apply** (*rule-tac*  $x=Q$  **in** *exI*)  
**apply** (*rule-tac*  $x=A$  **in** *exI*)  
**by** *simp*

**lemma** *app-closure-sound*:  
**assumes** *adapt*:  $P \subseteq \{s. \exists P' Q' A'. \forall n. \Gamma, \Theta \models n:_{/F} P' (\text{callClosure upd } (e', p))\}$   
 $Q', A' \wedge$   

$$\text{upd } x s \in P' \wedge Q' \subseteq Q \wedge A' \subseteq A$$
**assumes** *ap*:  $\text{upd } e = \text{upd } e' \circ \text{upd } x$   
**shows**  $\Gamma, \Theta \models n:_{/F} P (\text{callClosure upd } (e, p)) Q, A$   
**proof** (*rule cvalidI*)  
**fix**  $s t$   
**assume** *ctxt*:  $\forall (P, p, Q, A) \in \Theta. \Gamma \models n:_{/F} P \text{ Call } p Q, A$   
**assume** *exec-e*:  $\Gamma \vdash (\text{callClosure upd } (e, p), \text{Normal } s) = n \Rightarrow t$   
**assume**  $P: s \in P$   
**assume**  $t: t \notin \text{Fault } F$   
**from**  $P$  **adapt** **obtain**  $P' Q' A'$   
**where**  
*valid*:  $\forall n. \Gamma, \Theta \models n:_{/F} P' (\text{callClosure upd } (e', p)) Q', A'$  **and**  
*init-P'*:  $\text{upd } x s \in P'$  **and**  
 $Q: Q' \subseteq Q$  **and**  
 $A: A' \subseteq A$   
**by** *auto*  
**from** *exec-e* [*simplified callClosure-def*] **obtain**  $s'$   
**where**

```

  exec-e:  $\Gamma \vdash \langle \text{Basic} (\text{upd} (\text{fst} (e, p))), \text{Normal } s \rangle =n \Rightarrow s'$  and
  exec-p:  $\Gamma \vdash \langle \text{Call} (\text{snd} (e, p)), s' \rangle =n \Rightarrow t$ 
  by cases
  from exec-e [simplified]
  have s':  $s' = \text{Normal} (\text{upd } e \ s)$ 
  by cases simp
  from ap obtain s'' where
  s'':  $\text{upd } x \ s = s''$  and upd-e':  $\text{upd } e' \ s'' = \text{upd } e \ s$ 
  by auto
  from ap s' execn.Basic [where  $f = (\text{upd} (\text{fst} (e', p)))$  and  $s = \text{upd } x \ s$  and  $\Gamma = \Gamma$ ]
  have exec-e':  $\Gamma \vdash \langle \text{Basic} (\text{upd} (\text{fst} (e', p))), \text{Normal} (\text{upd } x \ s) \rangle =n \Rightarrow s'$ 
  by simp
  with exec-p
  have  $\Gamma \vdash \langle \text{callClosure } \text{upd} (e', p), \text{Normal} (\text{upd } x \ s) \rangle =n \Rightarrow t$ 
  by (auto simp add: callClosure-def intro: execn.Seq)
  from cvalidD [OF valid [rule-format] ctxt this init-P∧] t Q A
  show  $t \in \text{Normal } ' Q \cup \text{Abrupt } ' A$ 
  by auto
qed

```

**lemma app-closure:**

```

  assumes adapt:  $P \subseteq \{s. \exists P' Q' A'. \Gamma, \Theta \vdash_{/F} P' (\text{callClosure } \text{upd} (e', p)) Q', A' \wedge$ 
     $\text{upd } x \ s \in P' \wedge Q' \subseteq Q \wedge A' \subseteq A\}$ 
  assumes ap:  $\text{upd } e = \text{upd } e' \circ \text{upd } x$ 
  shows  $\Gamma, \Theta \vdash_{/F} P (\text{callClosure } \text{upd} (e, p)) Q, A$ 
  apply (rule hoare-complete')
  apply (rule allI)
  apply (rule app-closure-sound [where  $x=x$  and  $e'=e'$ , OF - ap])
  using adapt
  apply (blast intro: hoare-cvalid)
done

```

**lemma app-closure-spec:**

```

  assumes adapt:  $P \subseteq \{s. \exists Z. \text{upd } x \ s \in P' \ Z \wedge Q' \ Z \subseteq Q \wedge A' \ Z \subseteq A\}$ 
  assumes ap:  $\text{upd } e = \text{upd } e' \circ \text{upd } x$ 
  assumes spec:  $\forall Z. \Gamma, \Theta \vdash_{/F} (P' \ Z) (\text{callClosure } \text{upd} (e', p)) (Q' \ Z), (A' \ Z)$ 
  shows  $\Gamma, \Theta \vdash_{/F} P (\text{callClosure } \text{upd} (e, p)) Q, A$ 
  apply (rule app-closure [OF - ap])
  apply clarsimp
  using adapt spec
  apply -
  apply (drule (1) in-subsetD)
  apply clarsimp
  apply (rule-tac  $x=P' \ Z$  in exI)
  apply (rule-tac  $x=Q' \ Z$  in exI)
  apply (rule-tac  $x=A' \ Z$  in exI)
  apply blast
done

```

Implementation of closures as association lists.

**definition** *gen-upd var es s = foldl ( $\lambda s (x,i). \text{the } (\text{var } x) \text{ i } s$ ) s es*

**definition** *ap es c  $\equiv$  (es@fst c,snd c)*

**lemma** *gen-upd-app:  $\bigwedge es'. \text{gen-upd var } (es@es') = \text{gen-upd var } es' \circ \text{gen-upd var } es$*

**apply** (*induct es*)  
**apply** (*rule ext*)  
**apply** (*simp add: gen-upd-def*)  
**apply** (*rule ext*)  
**apply** (*simp add: gen-upd-def*)  
**done**

**lemma** *gen-upd-ap:*

*gen-upd var (fst (ap es (es',p))) = gen-upd var es'  $\circ$  gen-upd var es*  
**by** (*simp add: gen-upd-app ap-def*)

**lemma** *ap-closure:*

**assumes** *adapt:  $P \subseteq \{s. \exists P' Q' A'. \Gamma, \Theta \vdash_F P' (\text{callClosure } (\text{gen-upd var}) c) Q', A' \wedge$*

*gen-upd var es s  $\in P' \wedge Q' \subseteq Q \wedge A' \subseteq A\}$*

**shows**  $\Gamma, \Theta \vdash_F P (\text{callClosure } (\text{gen-upd var}) (\text{ap es } c)) Q, A$

**proof** –

**obtain** *es' p where c: c=(es',p)*

**by** (*cases c*)

**have** *gen-upd var (fst (ap es (es',p))) = gen-upd var es'  $\circ$  gen-upd var es*

**by** (*simp add: gen-upd-ap*)

**from** *app-closure [OF adapt [simplified c] this]*

**show** *?thesis*

**by** (*simp add: c ap-def*)

**qed**

**lemma** *ap-closure-spec:*

**assumes** *adapt:  $P \subseteq \{s. \exists Z. \text{gen-upd var es s  $\in P' Z \wedge Q' Z \subseteq Q \wedge A' Z \subseteq A\}$$*

**assumes** *spec:  $\forall Z. \Gamma, \Theta \vdash_F (P' Z) (\text{callClosure } (\text{gen-upd var}) c) (Q' Z), (A' Z)$*

**shows**  $\Gamma, \Theta \vdash_F P (\text{callClosure } (\text{gen-upd var}) (\text{ap es } c)) Q, A$

**proof** –

**obtain** *es' p where c: c=(es',p)*

**by** (*cases c*)

**have** *gen-upd var (fst (ap es (es',p))) = gen-upd var es'  $\circ$  gen-upd var es*

**by** (*simp add: gen-upd-ap*)

**from** *app-closure-spec [OF adapt [simplified c] this spec [simplified c]]*

**show** *?thesis*

**by** (*simp add: c ap-def*)

**qed**

**end**

```

theory ClosureEx
imports ../Vcg ../Simpl-Heap Closure
begin

record globals =
  cnt-' :: ref  $\Rightarrow$  nat
  alloc-' :: ref list
  free-' :: nat
record 'g vars = 'g state +
  p-' :: ref
  r-' :: nat
  n-' :: nat
  m-' :: nat
  c-' :: (string  $\times$  ref) list  $\times$  string
  d-' :: (string  $\times$  ref) list  $\times$  string
  e-' :: (string  $\times$  nat) list  $\times$  string

definition varn = [ 'n''  $\mapsto$  ( $\lambda x.$  n-'-update ( $\lambda.$  x)),
  'm''  $\mapsto$  ( $\lambda x.$  m-'-update ( $\lambda.$  x))]

definition updn = gen-upd varn

lemma updn-ap: updn (fst (ap es (es',p))) = updn es'  $\circ$  updn es
  by (simp add: updn-def gen-upd-ap)

lemma
 $\Gamma \vdash \{ \text{'n} = n_0 \wedge (\forall i j. \Gamma \vdash \{ \text{'n} = i \wedge \text{'m} = j \} \text{ callClosure upd}_n \text{'e } \{ \text{'r} = i + j \}) \}$ 
  'e ::= (ap [ 'n'', 'n ]) 'e
   $\{ \forall j. \Gamma \vdash \{ \text{'m} = j \} \text{ callClosure upd}_n \text{'e } \{ \text{'r} = n_0 + j \} \}$ 
apply vcg-step
apply clarify
apply (rule ap-closure [ where var = varn, folded updn-def ])
apply clarsimp
apply (rename-tac s s')
apply (erule-tac x = n-' s in allE)
apply (erule-tac x = m-' s' in allE)
apply (rule exI)
apply (rule exI)
apply (rule conjI)
apply (assumption)
apply (simp add: updn-def gen-upd-def varn-def)
done

definition var = [ 'p''  $\mapsto$  ( $\lambda x.$  p-'-update ( $\lambda.$  x))]

```



**definition**  $upd = gen-upd\ var$

**procedures**  $Inc(p|r) =$   
 $\quad 'p \rightarrow 'cnt ::= 'p \rightarrow 'cnt + 1;;$   
 $\quad 'r ::= 'p \rightarrow 'cnt$

**lemma** (in  $Inc-impl$ )  
 $\forall i\ p. \Gamma \vdash \{ 'p \rightarrow 'cnt = i \} 'r ::= PROC\ Inc('p) \{ 'r=i+1 \wedge 'p \rightarrow 'cnt = i+1 \}$   
**apply**  $vcg$   
**apply**  $simp$   
**done**

**procedures** (imports  $Inc-signature$ )  $NewCounter(|c) =$   
 $\quad 'p ::= NEW\ 1\ [ 'cnt ::= 0];;$   
 $\quad 'c ::= ([('p'', 'p)], Inc-'proc)$

**locale**  $NewCounter-impl' = NewCounter-impl + Inc-impl$

**lemma** (in  $NewCounter-impl'$ )

**shows**

$\forall alloc. \Gamma \vdash \{ 1 \leq 'free \} 'c ::= PROC\ NewCounter()$   
 $\quad \{ \exists p. p \rightarrow 'cnt = 0 \wedge$   
 $\quad (\forall i. \Gamma \vdash \{ p \rightarrow 'cnt = i \} callClosure\ upd\ 'c \{ 'r=i+1 \wedge p \rightarrow 'cnt = i+1 \}) \}$

**apply**  $vcg$

**apply**  $simp$

**apply** ( $rule-tac\ x=new\ (set\ alloc)\ in\ exI$ )

**apply**  $simp$

**apply** ( $simp\ add: callClosure-def$ )

**apply**  $vcg-step$

**apply**  $vcg-step$

**apply**  $vcg-step$

**apply**  $vcg-step$

**apply** ( $simp\ add: upd-def\ var-def\ gen-upd-def$ )

**done**

**lemma** (in  $NewCounter-impl'$ )

**shows**

$\forall alloc. \Gamma \vdash \{ 1 \leq 'free \} 'c ::= PROC\ NewCounter()$   
 $\quad \{ \exists p. p \rightarrow 'cnt = 0 \wedge$   
 $\quad (\forall i. \Gamma \vdash \{ p \rightarrow 'cnt = i \} callClosure\ upd\ 'c \{ 'r=i+1 \wedge p \rightarrow 'cnt = i+1 \}) \}$

**apply**  $vcg$

**apply**  $simp$

**apply** ( $rule-tac\ x=new\ (set\ alloc)\ in\ exI$ )

**apply**  $simp$

**apply** ( $simp\ add: callClosure-def$ )

**apply**  $vcg-step$

**apply**  $vcg-step$

**apply**  $vcg-step$

```

apply vcg-step
apply (simp add: upd-def var-def gen-upd-def)
done

```

```

lemma (in NewCounter-impl')

```

```

shows NewCounter-spec:

```

$$\forall \text{alloc. } \Gamma \vdash \{1 \leq \text{'free} \wedge \text{'alloc} = \text{alloc}\} \text{'c} ::= \text{PROC NewCounter}()$$

$$\{ \exists p. p \notin \text{set alloc} \wedge p \in \text{set 'alloc} \wedge p \neq \text{Null} \wedge p \rightarrow \text{'cnt} = 0 \wedge$$

$$(\forall i. \Gamma \vdash \{p \rightarrow \text{'cnt} = i\} \text{callClosure upd 'c} \{r = i + 1 \wedge p \rightarrow \text{'cnt} = i + 1\}) \}$$

```

apply vcg
apply clarsimp
apply (rule-tac x=new (set alloc) in exI)
apply simp
apply (simp add: callClosure-def)
apply vcg-step
apply vcg-step
apply vcg-step
apply vcg-step
apply (simp add: upd-def var-def gen-upd-def)
done

```

```

lemma  $\Gamma \vdash \{ \exists p. p \neq \text{Null} \wedge p \rightarrow \text{'cnt} = i \wedge$ 
 $(\forall i. \Gamma \vdash \{p \rightarrow \text{'cnt} = i\} \text{callClosure upd 'c} \{r = i + 1 \wedge p \rightarrow \text{'cnt} = i + 1\}) \}$ 
 $\text{dynCallClosure } (\lambda s. s) \text{ upd } c' (\lambda s t. s(\text{globals} := \text{globals } t))$ 
 $(\lambda s t. \text{Basic } (\lambda u. u(r' := r' t)))$ 
 $\{r = i + 1\}$ 

```

```

apply (rule conseq-extract-pre)
apply clarify
apply (rule dynCallClosureFix)
apply (simp only: Ball-def)
prefer 3
apply (assumption)
prefer 2
apply vcg-step
apply vcg-step
apply (simp only: simp-thms)
apply clarsimp
done

```

```

declare [hoare-trace = 1]

```

```

ML <
val hoare-tacs = #hoare-tacs (Hoare.get-data @ {context});
>

```

```

lemma (in NewCounter-impl')

```

```

shows  $\Gamma \vdash \{1 \leq \text{'free}\}$ 
 $\text{'c} ::= \text{CALL NewCounter } ();;$ 

```

```

      dynCallClosure ( $\lambda s. s$ ) upd c- $'$  ( $\lambda s t. s(\text{globals} := \text{globals } t)$ )
      ( $\lambda s t. \text{Basic } (\lambda u. u(\text{r-}' := \text{r-}' t))$ )
    } r=1 }
  apply vcg-step
  apply (rule dynCallClosure)
  prefer 2
  apply vcg-step
  apply vcg-step
  apply vcg-step
  apply clarsimp
  apply (erule-tac x=0 in allE)
  apply (rule exI)
  apply (rule exI)
  apply (rule conjI)
  apply (assumption)
  apply simp
  done

lemma (in NewCounter-impl')
  shows  $\Gamma \vdash \{1 \leq \text{'free}\}$ 
    'c ::= CALL NewCounter ();;
    dynCallClosure ( $\lambda s. s$ ) upd c- $'$  ( $\lambda s t. s(\text{globals} := \text{globals } t)$ )
    ( $\lambda s t. \text{Basic } (\lambda u. u(\text{r-}' := \text{r-}' t))$ );;
    dynCallClosure ( $\lambda s. s$ ) upd c- $'$  ( $\lambda s t. s(\text{globals} := \text{globals } t)$ )
    ( $\lambda s t. \text{Basic } (\lambda u. u(\text{r-}' := \text{r-}' t))$ )
  } r=2 }
  apply vcg-step
  apply (rule dynCallClosure)
  prefer 2
  apply vcg-step
  apply vcg-step
  apply vcg-step
  apply (rule dynCallClosure)
  apply vcg-step
  apply vcg-step
  apply vcg-step

  apply clarsimp
  apply (subgoal-tac  $\Gamma \vdash \{p \rightarrow \text{'cnt} = 0\}$  callClosure upd (c- $'$  t)  $\{r = \text{Suc } 0 \wedge p \rightarrow \text{'cnt} = \text{Suc } 0\}$ )
  apply (rule exI)
  apply (rule exI)
  apply (rule conjI)
  apply assumption
  apply clarsimp
  apply (erule-tac x=1 in allE)
  apply (rule exI)
  apply (rule exI)

```

```

apply (rule conjI)
apply assumption
apply clarsimp
apply (erule allE)
apply assumption
done

```

```

lemma (in NewCounter-impl')
shows  $\Gamma \vdash \{1 \leq 'free\}$ 
      'c ::= CALL NewCounter ();;
      'd ::= 'c;;
      dynCallClosure ( $\lambda s. s$ ) upd c-' ( $\lambda s t. s(\text{globals} := \text{globals } t)$ )
        ( $\lambda s t. \text{Basic } (\lambda u. u(n-' := r-' t))$ );;
      dynCallClosure ( $\lambda s. s$ ) upd d-' ( $\lambda s t. s(\text{globals} := \text{globals } t)$ )
        ( $\lambda s t. \text{Basic } (\lambda u. u(m-' := r-' t))$ );;
      'r ::= 'n + 'm
      {'r=3'}

```

```

apply vcg-step
apply vcg-step
apply (rule dynCallClosure)
prefer 2
apply vcg-step
apply vcg-step
apply vcg-step
apply (rule dynCallClosure)
apply vcg-step
apply vcg-step
apply vcg-step
apply vcg-step
apply clarsimp
apply (subgoal-tac  $\Gamma \vdash \{p \rightarrow 'cnt = 0\}$  callClosure upd (c-' t) {'r = Suc 0  $\wedge$  p  $\rightarrow$  '
  cnt = Suc 0'})
apply (rule exI)
apply (rule exI)
apply (rule conjI)
apply assumption
apply clarsimp
apply (erule-tac x=1 in allE)
apply (rule exI)
apply (rule exI)
apply (rule conjI)
apply assumption
apply clarsimp
apply (erule allE)
apply assumption
done

```

end

## 25 Experiments on State Composition

**theory** *Compose* **imports** *../HoareTotalProps* **begin**

We develop some theory to support state-space modular development of programs. These experiments aim at the representation of state-spaces with records. If we use *statespaces* instead we get this kind of compositionality for free.

### 25.1 Changing the State-Space

**definition**  $lift_f:: ('S \Rightarrow 's) \Rightarrow ('S \Rightarrow 's \Rightarrow 'S) \Rightarrow ('s \Rightarrow 's) \Rightarrow ('S \Rightarrow 'S)$   
where  $lift_f\ prj\ inject\ f = (\lambda S. inject\ S\ (f\ (prj\ S)))$

**definition**  $lift_s:: ('S \Rightarrow 's) \Rightarrow 's\ set \Rightarrow 'S\ set$   
where  $lift_s\ prj\ A = \{S. prj\ S \in A\}$

**definition**  $lift_r:: ('S \Rightarrow 's) \Rightarrow ('S \Rightarrow 's \Rightarrow 'S) \Rightarrow ('s \times 's)\ set$   
 $\Rightarrow ('S \times 'S)\ set$

**where**

$lift_r\ prj\ inject\ R = \{(S, T). (prj\ S, prj\ T) \in R \wedge T = inject\ S\ (prj\ T)\}$

**primrec**  $lift_c:: ('S \Rightarrow 's) \Rightarrow ('S \Rightarrow 's \Rightarrow 'S) \Rightarrow ('s, 'p, 'f)\ com \Rightarrow ('S, 'p, 'f)\ com$

**where**

$lift_c\ prj\ inject\ Skip = Skip$  |  
 $lift_c\ prj\ inject\ (Basic\ f) = Basic\ (lift_f\ prj\ inject\ f)$  |  
 $lift_c\ prj\ inject\ (Spec\ r) = Spec\ (lift_r\ prj\ inject\ r)$  |  
 $lift_c\ prj\ inject\ (Seq\ c_1\ c_2) =$   
 $(Seq\ (lift_c\ prj\ inject\ c_1)\ (lift_c\ prj\ inject\ c_2))$  |  
 $lift_c\ prj\ inject\ (Cond\ b\ c_1\ c_2) =$   
 $Cond\ (lift_s\ prj\ b)\ (lift_c\ prj\ inject\ c_1)\ (lift_c\ prj\ inject\ c_2)$  |  
 $lift_c\ prj\ inject\ (While\ b\ c) =$   
 $While\ (lift_s\ prj\ b)\ (lift_c\ prj\ inject\ c)$  |  
 $lift_c\ prj\ inject\ (Call\ p) = Call\ p$  |  
 $lift_c\ prj\ inject\ (DynCom\ c) = DynCom\ (\lambda s. lift_c\ prj\ inject\ (c\ (prj\ s)))$  |  
 $lift_c\ prj\ inject\ (Guard\ f\ g\ c) = Guard\ f\ (lift_s\ prj\ g)\ (lift_c\ prj\ inject\ c)$  |  
 $lift_c\ prj\ inject\ Throw = Throw$  |  
 $lift_c\ prj\ inject\ (Catch\ c_1\ c_2) =$   
 $Catch\ (lift_c\ prj\ inject\ c_1)\ (lift_c\ prj\ inject\ c_2)$

**lemma**  $lift_c\text{-}Skip: (lift_c\ prj\ inject\ c = Skip) = (c = Skip)$   
**by**  $(cases\ c)\ auto$

**lemma** *lift<sub>c</sub>-Basic*:

$$(lift_c\ prj\ inject\ c = Basic\ lf) = (\exists f. c = Basic\ f \wedge lf = lift_f\ prj\ inject\ f)$$

**by** (*cases c*) *auto*

**lemma** *lift<sub>c</sub>-Spec*:

$$(lift_c\ prj\ inject\ c = Spec\ lr) = (\exists r. c = Spec\ r \wedge lr = lift_r\ prj\ inject\ r)$$

**by** (*cases c*) *auto*

**lemma** *lift<sub>c</sub>-Seq*:

$$(lift_c\ prj\ inject\ c = Seq\ lc_1\ lc_2) =$$

$$(\exists c_1\ c_2. c = Seq\ c_1\ c_2 \wedge$$

$$lc_1 = lift_c\ prj\ inject\ c_1 \wedge lc_2 = lift_c\ prj\ inject\ c_2)$$

**by** (*cases c*) *auto*

**lemma** *lift<sub>c</sub>-Cond*:

$$(lift_c\ prj\ inject\ c = Cond\ lb\ lc_1\ lc_2) =$$

$$(\exists b\ c_1\ c_2. c = Cond\ b\ c_1\ c_2 \wedge lb = lift_s\ prj\ b \wedge$$

$$lc_1 = lift_c\ prj\ inject\ c_1 \wedge lc_2 = lift_c\ prj\ inject\ c_2)$$

**by** (*cases c*) *auto*

**lemma** *lift<sub>c</sub>-While*:

$$(lift_c\ prj\ inject\ c = While\ lb\ lc') =$$

$$(\exists b\ c'. c = While\ b\ c' \wedge lb = lift_s\ prj\ b \wedge$$

$$lc' = lift_c\ prj\ inject\ c')$$

**by** (*cases c*) *auto*

**lemma** *lift<sub>c</sub>-Call*:

$$(lift_c\ prj\ inject\ c = Call\ p) = (c = Call\ p)$$

**by** (*cases c*) *auto*

**lemma** *lift<sub>c</sub>-DynCom*:

$$(lift_c\ prj\ inject\ c = DynCom\ lc) =$$

$$(\exists C. c = DynCom\ C \wedge lc = (\lambda s. lift_c\ prj\ inject\ (C\ (prj\ s))))$$

**by** (*cases c*) *auto*

**lemma** *lift<sub>c</sub>-Guard*:

$$(lift_c\ prj\ inject\ c = Guard\ f\ lg\ lc') =$$

$$(\exists g\ c'. c = Guard\ f\ g\ c' \wedge lg = lift_s\ prj\ g \wedge$$

$$lc' = lift_c\ prj\ inject\ c')$$

**by** (*cases c*) *auto*

**lemma** *lift<sub>c</sub>-Throw*:

$$(lift_c\ prj\ inject\ c = Throw) = (c = Throw)$$

**by** (*cases c*) *auto*

**lemma** *lift<sub>c</sub>-Catch*:

$$(lift_c\ prj\ inject\ c = Catch\ lc_1\ lc_2) =$$

$$(\exists c_1\ c_2. c = Catch\ c_1\ c_2 \wedge$$

$$lc_1 = lift_c\ prj\ inject\ c_1 \wedge lc_2 = lift_c\ prj\ inject\ c_2)$$

**by** (*cases c*) *auto*

**definition** *xstate-map*:: ('S ⇒ 's) ⇒ ('S,'f) *xstate* ⇒ ('s,'f) *xstate*

**where**

*xstate-map g x* = (case *x* of  
    *Normal s* ⇒ *Normal (g s)*  
    | *Abrupt s* ⇒ *Abrupt (g s)*  
    | *Fault f* ⇒ *Fault f*  
    | *Stuck* ⇒ *Stuck*)

**lemma** *xstate-map-simps* [*simp*]:

*xstate-map g (Normal s)* = *Normal (g s)*

*xstate-map g (Abrupt s)* = *Abrupt (g s)*

*xstate-map g (Fault f)* = (*Fault f*)

*xstate-map g Stuck* = *Stuck*

**by** (*auto simp add: xstate-map-def*)

**lemma** *xstate-map-Normal-conv*:

*xstate-map g S = Normal s* = (∃ *s'*. *S=Normal s' ∧ s = g s'*)

**by** (*cases S*) *auto*

**lemma** *xstate-map-Abrupt-conv*:

*xstate-map g S = Abrupt s* = (∃ *s'*. *S=Abrupt s' ∧ s = g s'*)

**by** (*cases S*) *auto*

**lemma** *xstate-map-Fault-conv*:

*xstate-map g S = Fault f* = (*S=Fault f*)

**by** (*cases S*) *auto*

**lemma** *xstate-map-Stuck-conv*:

*xstate-map g S = Stuck* = (*S=Stuck*)

**by** (*cases S*) *auto*

**lemmas** *xstate-map-convs* = *xstate-map-Normal-conv xstate-map-Abrupt-conv*

*xstate-map-Fault-conv xstate-map-Stuck-conv*

**definition** *state*:: ('s,'f) *xstate* ⇒ 's

**where**

*state x* = (case *x* of  
    *Normal s* ⇒ *s*  
    | *Abrupt s* ⇒ *s*  
    | *Fault g* ⇒ *undefined*  
    | *Stuck* ⇒ *undefined*)

**lemma** *state-simps* [*simp*]:

*state (Normal s)* = *s*

*state (Abrupt s)* = *s*

by (auto simp add: state-def )

**locale** *lift-state-space* =  
**fixes** *project*:: $'S \Rightarrow 's$   
**fixes** *inject*:: $'S \Rightarrow 's \Rightarrow 'S$   
**fixes** *project<sub>x</sub>*:: $('S, 'f) \text{ xstate} \Rightarrow ('s, 'f) \text{ xstate}$   
**fixes** *lift<sub>e</sub>*:: $('s, 'p, 'f) \text{ body} \Rightarrow ('S, 'p, 'f) \text{ body}$   
**fixes** *lift<sub>c</sub>*:: $('s, 'p, 'f) \text{ com} \Rightarrow ('S, 'p, 'f) \text{ com}$   
**fixes** *lift<sub>f</sub>*:: $('s \Rightarrow 's) \Rightarrow ('S \Rightarrow 'S)$   
**fixes** *lift<sub>s</sub>*:: $'s \text{ set} \Rightarrow 'S \text{ set}$   
**fixes** *lift<sub>r</sub>*:: $('s \times 's) \text{ set} \Rightarrow ('S \times 'S) \text{ set}$   
**assumes** *proj-inj-commute*:  $\bigwedge S s. \text{ project } (\text{inject } S s) = s$   
**defines** *lift<sub>c</sub>*  $\equiv \text{Compose.lift}_c \text{ project inject}$   
**defines** *project<sub>x</sub>*  $\equiv \text{xstate-map project}$   
**defines** *lift<sub>e</sub>*  $\equiv (\lambda \Gamma p. \text{map-option lift}_c (\Gamma p))$   
**defines** *lift<sub>f</sub>*  $\equiv \text{Compose.lift}_f \text{ project inject}$   
**defines** *lift<sub>s</sub>*  $\equiv \text{Compose.lift}_s \text{ project}$   
**defines** *lift<sub>r</sub>*  $\equiv \text{Compose.lift}_r \text{ project inject}$

**lemma** (in *lift-state-space*) *lift<sub>f</sub>-simp*:  
*lift<sub>f</sub> f*  $\equiv \lambda S. \text{inject } S (f (\text{project } S))$   
 by (simp add: *lift<sub>f</sub>-def* *Compose.lift<sub>f</sub>-def*)

**lemma** (in *lift-state-space*) *lift<sub>s</sub>-simp*:  
*lift<sub>s</sub> A*  $\equiv \{S. \text{project } S \in A\}$   
 by (simp add: *lift<sub>s</sub>-def* *Compose.lift<sub>s</sub>-def*)

**lemma** (in *lift-state-space*) *lift<sub>r</sub>-simp*:  
*lift<sub>r</sub> R*  $\equiv \{(S, T). (\text{project } S, \text{project } T) \in R \wedge T = \text{inject } S (\text{project } T)\}$   
 by (simp add: *lift<sub>r</sub>-def* *Compose.lift<sub>r</sub>-def*)

**lemma** (in *lift-state-space*) *lift<sub>c</sub>-Skip-simp* [*simp*]:  
*lift<sub>c</sub> Skip* = *Skip*  
 by (simp add: *lift<sub>c</sub>-def*)

**lemma** (in *lift-state-space*) *lift<sub>c</sub>-Basic-simp* [*simp*]:  
*lift<sub>c</sub> (Basic f)* = *Basic (lift<sub>f</sub> f)*  
 by (simp add: *lift<sub>c</sub>-def* *lift<sub>f</sub>-def*)

**lemma** (in *lift-state-space*) *lift<sub>c</sub>-Spec-simp* [*simp*]:  
*lift<sub>c</sub> (Spec r)* = *Spec (lift<sub>r</sub> r)*  
 by (simp add: *lift<sub>c</sub>-def* *lift<sub>r</sub>-def*)

**lemma** (in *lift-state-space*) *lift<sub>c</sub>-Seq-simp* [*simp*]:  
*lift<sub>c</sub> (Seq c<sub>1</sub> c<sub>2</sub>)* =  
*(Seq (lift<sub>c</sub> c<sub>1</sub>) (lift<sub>c</sub> c<sub>2</sub>))*  
 by (simp add: *lift<sub>c</sub>-def*)

**lemma** (in *lift-state-space*) *lift<sub>c</sub>-Cond-simp* [*simp*]:  
*lift<sub>c</sub> (Cond b c<sub>1</sub> c<sub>2</sub>)* =



```

    Cond (lifts b) (liftc c1) (liftc c2)
  by (simp add: liftc-def lifts-def)
lemma (in lift-state-space) liftc-While-simp [simp]:
liftc (While b c) =
  While (lifts b) (liftc c)
  by (simp add: liftc-def lifts-def)
lemma (in lift-state-space) liftc-Call-simp [simp]:
liftc (Call p) = Call p
  by (simp add: liftc-def)
lemma (in lift-state-space) liftc-DynCom-simp [simp]:
liftc (DynCom c) = DynCom (λs. liftc (c (project s)))
  by (simp add: liftc-def)
lemma (in lift-state-space) liftc-Guard-simp [simp]:
liftc (Guard f g c) = Guard f (lifts g) (liftc c)
  by (simp add: liftc-def lifts-def)
lemma (in lift-state-space) liftc-Throw-simp [simp]:
liftc Throw = Throw
  by (simp add: liftc-def)
lemma (in lift-state-space) liftc-Catch-simp [simp]:
liftc (Catch c1 c2) =
  Catch (liftc c1) (liftc c2)
  by (simp add: liftc-def)

```

```

lemma (in lift-state-space) projectx-def':
projectx s ≡ (case s of
  Normal s ⇒ Normal (project s)
| Abrupt s ⇒ Abrupt (project s)
| Fault f ⇒ Fault f
| Stuck ⇒ Stuck)
  by (simp add: xstate-map-def projectx-def)

```

```

lemma (in lift-state-space) lifte-def':
lifte Γ p ≡ (case Γ p of Some bdy ⇒ Some (liftc bdy) | None ⇒ None)
  by (simp add: lifte-def map-option-case)

```

The problem is that  $lift_c project inject \circ \Gamma$  is quite a strong premise. The problem is that  $\Gamma$  is a function here. A map would be better. We only have to lift those procedures in the domain of  $\Gamma$ :  $\Gamma p = Some\ bdy \longrightarrow \Gamma' p = Some\ lift_c\ project\ inject\ bdy$ . We then can com up with theorems that allow us to extend the domains of  $\Gamma$  and preserve validity.

```

lemma (in lift-state-space)
{(S,T). ∃ t. (project S,t) ∈ r ∧ T=inject S t}
⊆ {(S,T). (project S,project T) ∈ r ∧ T=inject S (project T)}
  apply clarsimp
  apply (rename-tac S t)
  apply (simp add: proj-inj-commute)
  done

```

```

lemma (in lift-state-space)

```

```

{(S,T). (project S,project T) ∈ r ∧ T=inject S (project T)}
⊆ {(S,T). ∃ t. (project S,t) ∈ r ∧ T=inject S t}
apply clarsimp
apply (rename-tac S T)
apply (rule-tac x=project T in exI)
apply simp
done

```

```

lemma (in lift-state-space) lift-exec:
assumes exec-lc: (lifte Γ) ⊢ ⟨lc,s⟩ ⇒ t
shows ∧ c. [ liftc c = lc ] ⇒
      Γ ⊢ ⟨c,projectx s⟩ ⇒ projectx t
using exec-lc
proof (induct)
  case Skip thus ?case
    by (auto simp add: projectx-def liftc-Skip liftc-def intro: exec.Skip)
  next
    case Guard thus ?case
      by (auto simp add: projectx-def lifts-def Compose.lifts-def liftc-Guard liftc-def
        intro: exec.Guard)
    next
      case GuardFault thus ?case
        by (auto simp add: projectx-def lifts-def Compose.lifts-def liftc-Guard liftc-def
          intro: exec.GuardFault)
    next
      case FaultProp thus ?case
        by (fastforce simp add: projectx-def)
    next
      case Basic
      thus ?case
        by (fastforce simp add: projectx-def liftc-Basic liftf-def Compose.liftf-def
          liftc-def
          proj-inj-commute
          intro: exec.Basic)
    next
      case Spec
      thus ?case
        by (fastforce simp add: projectx-def liftc-Spec liftf-def Compose.liftf-def
          liftr-def Compose.liftr-def liftc-def
          proj-inj-commute
          intro: exec.Spec)
    next
      case (SpecStuck s r)
      thus ?case
        apply (simp add: projectx-def)
        apply (clarsimp simp add: liftc-Spec liftc-def)
        apply (unfold liftr-def Compose.liftr-def)
        apply (rule exec.SpecStuck)

```

```

    apply (rule allI)
    apply (erule-tac x=inject s t in allE)
    apply clarsimp
    apply (simp add: proj-inj-commute)
  done
next
case Seq
thus ?case
  by (fastforce simp add: project_x-def lift_c-Seq lift_c-def intro: exec.intros)
next
case CondTrue
thus ?case
  by (auto simp add: project_x-def lift_s-def Compose.lift_s-def lift_c-Cond lift_c-def
    intro: exec.CondTrue)
next
case CondFalse
thus ?case
  by (auto simp add: project_x-def lift_s-def Compose.lift_s-def lift_c-Cond lift_c-def
    intro: exec.CondFalse)
next
case WhileTrue
thus ?case
  by (fastforce simp add: project_x-def lift_s-def Compose.lift_s-def
    lift_c-While lift_c-def
    intro: exec.WhileTrue)
next
case WhileFalse
thus ?case
  by (fastforce simp add: project_x-def lift_s-def Compose.lift_s-def
    lift_c-While lift_c-def
    intro: exec.WhileFalse)
next
case Call
thus ?case
  by (fastforce simp add:
    project_x-def lift_c-Call lift_f-def Compose.lift_f-def lift_c-def
    lift_e-def
    intro: exec.Call)
next
case CallUndefined
thus ?case
  by (fastforce simp add:
    project_x-def lift_c-Call lift_f-def Compose.lift_f-def lift_c-def
    lift_e-def
    intro: exec.CallUndefined)
next
case StuckProp thus ?case
  by (fastforce simp add: project_x-def)
next

```

```

case DynCom
thus ?case
  by (fastforce simp add:
    project_x-def lift_c-DynCom lift_f-def Compose.lift_f-def lift_c-def
    intro: exec.DynCom)
next
case Throw thus ?case
  by (fastforce simp add: project_x-def lift_c-Throw lift_c-def intro: exec.Throw)
next
case AbruptProp thus ?case
  by (fastforce simp add: project_x-def)
next
case CatchMatch
thus ?case
  by (fastforce simp add: project_x-def lift_c-Catch lift_c-def intro: exec.CatchMatch)
next
case (CatchMiss c1 s t c2 c)
thus ?case
  by (cases t)
  (fastforce simp add: project_x-def lift_c-Catch lift_c-def intro: exec.CatchMiss)+
qed

```

```

lemma (in lift-state-space) lift-exec':
assumes exec-lc: (lift_e  $\Gamma$ ) $\vdash$  $\langle$ lift_c c,s $\rangle \Rightarrow t$ 
shows  $\Gamma \vdash \langle c, \text{project}_x s \rangle \Rightarrow \text{project}_x t$ 
  using lift-exec [OF exec-lc]
  by simp

```

```

lemma (in lift-state-space) lift-valid:
assumes valid:  $\Gamma \models_F P \ c \ Q, A$ 
shows
  (lift_e  $\Gamma$ ) $\models_F$  (lift_s P) (lift_c c) (lift_s Q), (lift_s A)
proof (rule validI)
  fix s t
  assume lexec:
    (lift_e  $\Gamma$ ) $\vdash$  $\langle$ lift_c c, Normal s $\rangle \Rightarrow t$ 
  assume lP: s  $\in$  lift_s P
  assume noFault: t  $\notin$  Fault ' F
  show t  $\in$  Normal ' lift_s Q  $\cup$  Abrupt ' lift_s A
  proof –
    from lexec
    have  $\Gamma \vdash \langle c, \text{project}_x (\text{Normal } s) \rangle \Rightarrow (\text{project}_x t)$ 
      by (rule lift-exec) (simp-all)
    moreover
    from lP have project s  $\in$  P
      by (simp add: lift_s-def Compose.lift_s-def project_x-def)
    ultimately

```

```

have  $project_x t \in Normal \text{ ' } Q \cup Abrupt \text{ ' } A$ 
  using valid noFault
  apply (clarsimp simp add: valid-def project_x-def)
  apply (cases t)
  apply auto
  done
thus ?thesis
  apply (simp add: lift_s-def Compose.lift_s-def)
  apply (cases t)
  apply (auto simp add: project_x-def)
  done
qed
qed

lemma (in lift-state-space) lift-hoarep:
  assumes  $deriv: \Gamma, \{\} \vdash_F P \text{ c } Q, A$ 
  shows
     $(lift_e \Gamma), \{\} \vdash_F (lift_s P) (lift_c c) (lift_s Q), (lift_s A)$ 
  apply (rule hoare-complete)
  apply (insert hoare-sound [OF deriv])
  apply (rule lift-valid)
  apply (simp add: cvalid-def)
  done

lemma (in lift-state-space) lift-hoarep':
   $\forall Z. \Gamma, \{\} \vdash_F (P Z) \text{ c } (Q Z), (A Z) \implies$ 
   $\forall Z. (lift_e \Gamma), \{\} \vdash_F (lift_s (P Z)) (lift_c c)$ 
   $(lift_s (Q Z)), (lift_s (A Z))$ 
  apply (iprover intro: lift-hoarep)
  done

lemma (in lift-state-space) lift-termination:
  assumes termi:  $\Gamma \vdash c \downarrow s$ 
  shows  $\bigwedge S. project_x S = s \implies$ 
   $lift_e \Gamma \vdash (lift_c c) \downarrow S$ 
  using termi
proof (induct)
  case Skip thus ?case
    by (clarsimp simp add: terminates.Skip project_x-def xstate-map-convs)
  next
  case Basic thus ?case
    by (fastforce simp add: project_x-def xstate-map-convs intro: terminates.intros)
  next
  case Spec thus ?case
    by (fastforce simp add: project_x-def xstate-map-convs intro: terminates.intros)
  next
  case Guard thus ?case

```

```

    by (auto simp add: project_x-def xstate-map-convs intro: terminates.intros)
next
case GuardFault thus ?case
  by (auto simp add: project_x-def xstate-map-convs lift_s-def Compose.lift_s-def
      intro: terminates.intros)
next
case Fault thus ?case by (clarsimp simp add: project_x-def xstate-map-convs)
next
case (Seq c1 s c2)
have project_x S = Normal s by fact
then obtain s' where S: S=Normal s' and s: s = project s'
  by (auto simp add: project_x-def xstate-map-convs)
from Seq have lift_e  $\Gamma \vdash \text{lift}_c c1 \downarrow S$ 
  by simp
moreover
{
  fix w
  assume exec-lc1: lift_e  $\Gamma \vdash \langle \text{lift}_c c1, \text{Normal } s' \rangle \Rightarrow w$ 
  have lift_e  $\Gamma \vdash \text{lift}_c c2 \downarrow w$ 
  proof (cases w)
    case (Normal w')
    with lift-exec [where c=c1, OF exec-lc1] s
    have  $\Gamma \vdash \langle c1, \text{Normal } s \rangle \Rightarrow \text{Normal } (\text{project } w')$ 
      by (simp add: project_x-def)
    from Seq.hyps (3) [rule-format, OF this] Normal
    show lift_e  $\Gamma \vdash \text{lift}_c c2 \downarrow w$ 
      by (auto simp add: project_x-def xstate-map-convs)
  qed (auto)
}
ultimately show ?case
  using S s
  by (auto intro: terminates.intros)
next
case CondTrue thus ?case
  by (fastforce simp add: project_x-def lift_s-def Compose.lift_s-def xstate-map-convs
      intro: terminates.intros)
next
case CondFalse thus ?case
  by (fastforce simp add: project_x-def lift_s-def Compose.lift_s-def xstate-map-convs
      intro: terminates.intros)
next
case (WhileTrue s b c)
have project_x S = Normal s by fact
then obtain s' where S: S=Normal s' and s: s = project s'
  by (auto simp add: project_x-def xstate-map-convs)
from WhileTrue have lift_e  $\Gamma \vdash \text{lift}_c c \downarrow S$ 
  by simp
moreover
{

```

```

fix w
assume exec-lc:  $\text{lift}_e \Gamma \vdash \langle \text{lift}_c c, \text{Normal } s \rangle \Rightarrow w$ 
have  $\text{lift}_e \Gamma \vdash \text{lift}_c (\text{While } b \ c) \downarrow w$ 
proof (cases w)
  case (Normal w')
    with lift-exec [where  $c=c$ , OF exec-lc] s
    have  $\Gamma \vdash \langle c, \text{Normal } s \rangle \Rightarrow \text{Normal } (\text{project } w')$ 
      by (simp add: projectx-def)
    from WhileTrue.hyps (4) [rule-format, OF this] Normal
    show  $\text{lift}_e \Gamma \vdash \text{lift}_c (\text{While } b \ c) \downarrow w$ 
      by (auto simp add: projectx-def xstate-map-convs)
    qed (auto)
  }
ultimately show ?case
  using S s
  by (auto intro: terminates.intros)
next
case WhileFalse thus ?case
  by (fastforce simp add: projectx-def lifts-def Compose.lifts-def xstate-map-convs
    intro: terminates.intros)
next
case Call thus ?case
  by (fastforce simp add: projectx-def xstate-map-convs lifte-def
    intro: terminates.intros)
next
case CallUndefined thus ?case
  by (fastforce simp add: projectx-def xstate-map-convs lifte-def
    intro: terminates.intros)
next
case Stuck thus ?case
  by (fastforce simp add: projectx-def xstate-map-convs)
next
case DynCom thus ?case
  by (fastforce simp add: projectx-def xstate-map-convs
    intro: terminates.intros)
next
case Throw thus ?case
  by (fastforce simp add: projectx-def xstate-map-convs
    intro: terminates.intros)
next
case Abrupt thus ?case
  by (fastforce simp add: projectx-def xstate-map-convs
    intro: terminates.intros)
next
case (Catch c1 s c2)
have  $\text{project}_x S = \text{Normal } s$  by fact
then obtain s' where S:  $S = \text{Normal } s'$  and s:  $s = \text{project } s'$ 
  by (auto simp add: projectx-def xstate-map-convs)
from Catch have  $\text{lift}_e \Gamma \vdash \text{lift}_c c1 \downarrow S$ 

```

```

    by simp
  moreover
  {
    fix w
    assume exec-lc1: lift_e Γ ⊢ ⟨lift_c c1, Normal s⟩ ⇒ Abrupt w
    have lift_e Γ ⊢ lift_c c2 ↓ Normal w
    proof -
      from lift-exec [where c=c1, OF exec-lc1] s
      have Γ ⊢ ⟨c1, Normal s⟩ ⇒ Abrupt (project w)
        by (simp add: project_x-def)
      from Catch.hyps (3) [rule-format, OF this]
      show lift_e Γ ⊢ lift_c c2 ↓ Normal w
        by (auto simp add: project_x-def xstate-map-convs)
    qed
  }
  ultimately show ?case
    using S s
    by (auto intro: terminates.intros)
qed

```

```

lemma (in lift-state-space) lift-termination':
  assumes termi: Γ ⊢ c ↓ project_x S
  shows lift_e Γ ⊢ ⟨lift_c c⟩ ↓ S
    using lift-termination [OF termi]
    by iprover

```

```

lemma (in lift-state-space) lift-validt:
  assumes valid: Γ ⊢_t/F P c Q, A
  shows (lift_e Γ) ⊢_t/F (lift_s P) (lift_c c) (lift_s Q), (lift_s A)
  proof -
    from valid
    have (lift_e Γ) ⊢_F (lift_s P) (lift_c c) (lift_s Q), (lift_s A)
      by (auto intro: lift-valid simp add: validt-def)
    moreover
    {
      fix S
      assume S ∈ lift_s P
      hence project S ∈ P
        by (simp add: lift_s-def Compose.lift_s-def)
      with valid have Γ ⊢ c ↓ project_x (Normal S)
        by (simp add: validt-def project_x-def)
      hence lift_e Γ ⊢ lift_c c ↓ Normal S
        by (rule lift-termination')
    }
    ultimately show ?thesis
      by (simp add: validt-def)
  qed

```



```

lemma (in lift-state-space) lift-hoare:
  assumes deriv:  $\Gamma, \{\} \vdash_{t/F} P \ c \ Q, A$ 
  shows
     $(\text{lift}_e \ \Gamma), \{\} \vdash_{t/F} (\text{lift}_s \ P) \ (\text{lift}_c \ c) \ (\text{lift}_s \ Q), (\text{lift}_s \ A)$ 
apply (rule hoaret-complete)
apply (insert hoaret-sound [OF deriv])
apply (rule lift-validt)
apply (simp add: cvalidt-def)
done

locale lift-state-space-ext = lift-state-space +
  assumes inj-proj-commute:  $\bigwedge S. \text{inject } S \ (\text{project } S) = S$ 
  assumes inject-last:  $\bigwedge S \ s \ t. \text{inject } (\text{inject } S \ s) \ t = \text{inject } S \ t$ 

lemma (in lift-state-space-ext) lift-exec-inject-same:
assumes exec-lc:  $(\text{lift}_e \ \Gamma) \vdash \langle lc, s \rangle \Rightarrow t$ 
shows  $\bigwedge c. \llbracket \text{lift}_c \ c = lc; t \notin (\text{Fault } ' \text{UNIV}) \cup \{\text{Stuck}\} \rrbracket \implies$ 
   $\text{state } t = \text{inject } (\text{state } s) \ (\text{project } (\text{state } t))$ 
using exec-lc
proof (induct)
  case Skip thus ?case
    by (clarsimp simp add: inj-proj-commute)
next
  case Guard thus ?case
    by (clarsimp simp add: lift_c-Guard lift_c-def)
next
  case GuardFault thus ?case
    by simp
next
  case FaultProp thus ?case by simp
next
  case Basic thus ?case
    by (clarsimp simp add: lift_f-def Compose.lift_f-def
      proj-inj-commute lift_c-Basic lift_c-def)
next
  case (Spec r) thus ?case
    by (clarsimp simp add: Compose.lift_r-def lift_c-Spec lift_c-def)
next
  case SpecStuck
  thus ?case by simp
next
  case (Seq lc1 s s' lc2 t c)
  have  $t \notin \text{Fault } ' \text{UNIV} \cup \{\text{Stuck}\}$  by fact
  have  $\text{lift}_c \ c = \text{Seq } lc1 \ lc2$  by fact
  then obtain c1 c2 where
     $c = \text{Seq } c1 \ c2$  and

```

```

    lc1: lc1 = lift_c c1 and
    lc2: lc2 = lift_c c2
    by (auto simp add: lift_c-Seq lift_c-def)
show ?case
proof (cases s')
  case (Normal s'')
  from Seq.hyps (2) [OF lc1 [symmetric]] this
  have s'' = inject s (project s'')
    by auto
  moreover from Seq.hyps (4) [OF lc2 [symmetric]] Normal t
  have state t = inject s'' (project (state t))
    by auto
  ultimately have state t = inject (inject s (project s'')) (project (state t))
    by simp
  then show ?thesis
    by (simp add: inject-last)
next
  case (Abrupt s'')
  from Seq.hyps (2) [OF lc1 [symmetric]] this
  have s'' = inject s (project s'')
    by auto
  moreover from Seq.hyps (4) [OF lc2 [symmetric]] Abrupt t
  have state t = inject s'' (project (state t))
    by auto
  ultimately have state t = inject (inject s (project s'')) (project (state t))
    by simp
  then show ?thesis
    by (simp add: inject-last)
next
  case (Fault f)
  with Seq
  have t = Fault f
    by (auto dest: Fault-end)
  with t have False by simp
  thus ?thesis ..
next
  case (Stuck)
  with Seq
  have t = Stuck
    by (auto dest: Stuck-end)
  with t have False by simp
  thus ?thesis ..
qed
next
  case CondTrue thus ?case
    by (clarsimp simp add: lift_c-Cond lift_c-def)
next
  case CondFalse thus ?case
    by (clarsimp simp add: lift_c-Cond lift_c-def)

```

```

next
  case (WhileTrue s lb lc' s' t c)
  have t:  $t \notin \text{Fault} \cup \text{UNIV} \cup \{\text{Stuck}\}$  by fact
  have lw:  $\text{lift}_c c = \text{While } lb \text{ } lc'$  by fact
  then obtain b c' where
    c:  $c = \text{While } b \text{ } c'$  and
    lb:  $lb = \text{lift}_s b$  and
    lc:  $lc' = \text{lift}_c c'$ 
    by (auto simp add: lift_c-While lift_s-def lift_c-def)
  show ?case
  proof (cases s')
    case (Normal s'')
    from WhileTrue.hyps (3) [OF lc [symmetric]] this
    have  $s'' = \text{inject } s \text{ (project } s'')$ 
      by auto
    moreover from WhileTrue.hyps (5) [OF lw] Normal t
    have  $\text{state } t = \text{inject } s'' \text{ (project (state } t))$ 
      by auto
    ultimately have  $\text{state } t = \text{inject (inject } s \text{ (project } s'')) \text{ (project (state } t))$ 
      by simp
    then show ?thesis
      by (simp add: inject-last)
  next
    case (Abrupt s'')
    from WhileTrue.hyps (3) [OF lc [symmetric]] this
    have  $s'' = \text{inject } s \text{ (project } s'')$ 
      by auto
    moreover from WhileTrue.hyps (5) [OF lw] Abrupt t
    have  $\text{state } t = \text{inject } s'' \text{ (project (state } t))$ 
      by auto
    ultimately have  $\text{state } t = \text{inject (inject } s \text{ (project } s'')) \text{ (project (state } t))$ 
      by simp
    then show ?thesis
      by (simp add: inject-last)
  next
    case (Fault f)
    with WhileTrue
    have  $t = \text{Fault } f$ 
      by (auto dest: Fault-end)
    with t have False by simp
    thus ?thesis ..
  next
    case Stuck
    with WhileTrue
    have  $t = \text{Stuck}$ 
      by (auto dest: Stuck-end)
    with t have False by simp
    thus ?thesis ..
qed

```

```

next
  case WhileFalse thus ?case
    by (clarsimp simp add: liftc-While inj-proj-commute)
next
  case Call thus ?case
    by (clarsimp simp add: inject-last liftc-Call lifte-def liftc-def)
next
  case CallUndefined thus ?case by simp
next
  case StuckProp thus ?case by simp
next
  case DynCom
  thus ?case
    by (clarsimp simp add: liftc-DynCom liftc-def)
next
  case Throw thus ?case
    by (simp add: inj-proj-commute)
next
  case AbruptProp thus ?case by (simp add: inj-proj-commute)
next
  case (CatchMatch lc1 s s' lc2 t c)
  have  $t \notin \text{Fault } \text{'UNIV} \cup \{\text{Stuck}\}$  by fact
  have  $\text{lift}_c c = \text{Catch } lc1 lc2$  by fact
  then obtain c1 c2 where
    c:  $c = \text{Catch } c1 c2$  and
    lc1:  $lc1 = \text{lift}_c c1$  and
    lc2:  $lc2 = \text{lift}_c c2$ 
    by (auto simp add: liftc-Catch liftc-def)
  from CatchMatch.hyps (2) [OF lc1 [symmetric]] this
  have  $s' = \text{inject } s (\text{project } s')$ 
    by auto
  moreover
  from CatchMatch.hyps (4) [OF lc2 [symmetric]] t
  have  $\text{state } t = \text{inject } s' (\text{project } (\text{state } t))$ 
    by auto
  ultimately have  $\text{state } t = \text{inject } (\text{inject } s (\text{project } s')) (\text{project } (\text{state } t))$ 
    by simp
  then show ?case
    by (simp add: inject-last)
next
  case CatchMiss
  thus ?case
    by (clarsimp simp add: liftc-Catch liftc-def)
qed

```

**lemma** (in *lift-state-space-ext*) *valid-inject-project*:  
**assumes** *noFaultStuck*:  
 $\Gamma \vdash \langle c, \text{Normal } (\text{project } \sigma) \rangle \Rightarrow \notin (\text{Fault } \text{'UNIV} \cup \{\text{Stuck}\})$   
**shows**  $\text{lift}_e \Gamma \models_{/F} \{\sigma\} \text{lift}_c c$

$\{t. t = \text{inject } \sigma (\text{project } t)\}, \{t. t = \text{inject } \sigma (\text{project } t)\}$

**proof** (*rule validI*)  
**fix**  $s\ t$   
**assume**  $\text{exec}: \text{lift}_e \Gamma \vdash \langle \text{lift}_c\ c, \text{Normal } s \rangle \Rightarrow t$   
**assume**  $P: s \in \{\sigma\}$   
**assume**  $\text{noFault}: t \notin \text{Fault } 'F$   
**show**  $t \in \text{Normal } ' \{t. t = \text{inject } \sigma (\text{project } t)\} \cup$   
 $\text{Abrupt } ' \{t. t = \text{inject } \sigma (\text{project } t)\}$   
**proof** –  
**from**  $\text{lift-exec } [OF\ \text{exec}]$   
**have**  $\Gamma \vdash \langle c, \text{project}_x (\text{Normal } s) \rangle \Rightarrow \text{project}_x\ t$   
**by**  $\text{simp}$   
**with**  $\text{noFaultStuck } P$  **have**  $t: t \notin \text{Fault } ' \text{UNIV} \cup \{\text{Stuck}\}$   
**by** ( $\text{auto simp add: final-notin-def project}_x\text{-def}$ )  
**from**  $\text{lift-exec-inject-same } [OF\ \text{exec refl this}] P$   
**have**  $\text{state } t = \text{inject } \sigma (\text{project } (\text{state } t))$   
**by**  $\text{simp}$   
**with**  $t$  **show**  $?thesis$   
**by** ( $\text{cases } t$ )  $\text{auto}$   
**qed**  
**qed**

**lemma** (*in lift-state-space-ext*) *lift-exec-inject-same'*:  
**assumes**  $\text{exec-lc}: (\text{lift}_e \Gamma) \vdash \langle \text{lift}_c\ c, S \rangle \Rightarrow T$   
**shows**  $\bigwedge c. \llbracket T \notin (\text{Fault } ' \text{UNIV}) \cup \{\text{Stuck}\} \rrbracket \Longrightarrow$   
 $\text{state } T = \text{inject } (\text{state } S) (\text{project } (\text{state } T))$   
**using**  $\text{lift-exec-inject-same } [OF\ \text{exec-lc}]$   
**by**  $\text{simp}$

**lemma** (*in lift-state-space-ext*) *valid-lift-modifies*:  
**assumes**  $\text{valid}: \forall s. \Gamma \models_{/F} \{s\}\ c\ (\text{Modif } s), (\text{ModifAbr } s)$   
**shows**  $(\text{lift}_e \Gamma) \models_{/F} \{S\}\ (\text{lift}_c\ c)$   
 $\{T. T \in \text{lift}_s (\text{Modif } (\text{project } S)) \wedge T = \text{inject } S (\text{project } T)\},$   
 $\{T. T \in \text{lift}_s (\text{ModifAbr } (\text{project } S)) \wedge T = \text{inject } S (\text{project } T)\}$

**proof** (*rule validI*)  
**fix**  $s\ t$   
**assume**  $\text{exec}: \text{lift}_e \Gamma \vdash \langle \text{lift}_c\ c, \text{Normal } s \rangle \Rightarrow t$   
**assume**  $P: s \in \{S\}$   
**assume**  $\text{noFault}: t \notin \text{Fault } 'F$   
**show**  $t \in \text{Normal } ' \{t \in \text{lift}_s (\text{Modif } (\text{project } S)).$   
 $t = \text{inject } S (\text{project } t)\} \cup$   
 $\text{Abrupt } ' \{t \in \text{lift}_s (\text{ModifAbr } (\text{project } S)).$   
 $t = \text{inject } S (\text{project } t)\}$   
**proof** –  
**from**  $\text{lift-exec } [OF\ \text{exec}]$   
**have**  $\Gamma \vdash \langle c, \text{project}_x (\text{Normal } s) \rangle \Rightarrow \text{project}_x\ t$   
**by**  $\text{auto}$

**moreover**  
**from** *noFault* **have**  $\text{project}_x t \notin \text{Fault} \text{ ' } F$   
**by** (*cases t*) (*auto simp add: project<sub>x</sub>-def*)  
**ultimately**  
**have**  $\text{project}_x t \in$   
 $\text{Normal} \text{ ' } (\text{Modif} (\text{project } s)) \cup \text{Abrupt} \text{ ' } (\text{ModifAbr} (\text{project } s))$   
**using** *valid [rule-format, of (project s)]*  
**by** (*auto simp add: valid-def project<sub>x</sub>-def*)  
**hence**  $t: t \in \text{Normal} \text{ ' } \text{lift}_s (\text{Modif} (\text{project } s)) \cup$   
 $\text{Abrupt} \text{ ' } \text{lift}_s (\text{ModifAbr} (\text{project } s))$   
**by** (*cases t*) (*auto simp add: project<sub>x</sub>-def lift<sub>s</sub>-def Compose.lift<sub>s</sub>-def*)  
**then have**  $t \notin \text{Fault} \text{ ' } \text{UNIV} \cup \{\text{Stuck}\}$   
**by** (*cases t*) *auto*  
**from** *lift-exec-inject-same [OF exec - this]*  
**have**  $\text{state } t = \text{inject} (\text{state} (\text{Normal } s)) (\text{project} (\text{state } t))$   
**by** *simp*  
**with** *t* **show** *?thesis*  
**using** *P* **by** *auto*  
**qed**  
**qed**

**lemma** (*in lift-state-space-ext*) *hoare-lift-modifies*:  
**assumes** *deriv*:  $\forall \sigma. \Gamma, \{\} \vdash_{/F} \{\sigma\} c (\text{Modif } \sigma), (\text{ModifAbr } \sigma)$   
**shows**  $\forall \sigma. (\text{lift}_e \Gamma), \{\} \vdash_{/F} \{\sigma\} (\text{lift}_c c)$   
 $\{T. T \in \text{lift}_s (\text{Modif} (\text{project } \sigma)) \wedge T = \text{inject } \sigma (\text{project } T)\},$   
 $\{T. T \in \text{lift}_s (\text{ModifAbr} (\text{project } \sigma)) \wedge T = \text{inject } \sigma (\text{project } T)\}$   
**apply** (*rule allI*)  
**apply** (*rule hoare-complete*)  
**apply** (*rule valid-lift-modifies*)  
**apply** (*rule allI*)  
**apply** (*insert hoare-sound [OF deriv [rule-format]]*)  
**apply** (*simp add: cvalid-def*)  
**done**

**lemma** (*in lift-state-space-ext*) *hoare-lift-modifies'*:  
**assumes** *deriv*:  $\forall \sigma. \Gamma, \{\} \vdash_{/F} \{\sigma\} c (\text{Modif } \sigma), (\text{ModifAbr } \sigma)$   
**shows**  $\forall \sigma. (\text{lift}_e \Gamma), \{\} \vdash_{/F} \{\sigma\} (\text{lift}_c c)$   
 $\{T. T \in \text{lift}_s (\text{Modif} (\text{project } \sigma)) \wedge$   
 $(\exists T'. T = \text{inject } \sigma T')\},$   
 $\{T. T \in \text{lift}_s (\text{ModifAbr} (\text{project } \sigma)) \wedge$   
 $(\exists T'. T = \text{inject } \sigma T')\}$   
**apply** (*rule allI*)  
**apply** (*rule HoarePartialDef.conseq [OF hoare-lift-modifies [OF deriv]]*)  
**apply** *blast*  
**done**

## 25.2 Renaming Procedures

**primrec** *rename*::  $(p \Rightarrow q) \Rightarrow (s, p, f) \text{ com} \Rightarrow (s, q, f) \text{ com}$

**where**

$rename\ N\ Skip = Skip \mid$   
 $rename\ N\ (Basic\ f) = Basic\ f \mid$   
 $rename\ N\ (Spec\ r) = Spec\ r \mid$   
 $rename\ N\ (Seq\ c_1\ c_2) = (Seq\ (rename\ N\ c_1)\ (rename\ N\ c_2)) \mid$   
 $rename\ N\ (Cond\ b\ c_1\ c_2) = Cond\ b\ (rename\ N\ c_1)\ (rename\ N\ c_2) \mid$   
 $rename\ N\ (While\ b\ c) = While\ b\ (rename\ N\ c) \mid$   
 $rename\ N\ (Call\ p) = Call\ (N\ p) \mid$   
 $rename\ N\ (DynCom\ c) = DynCom\ (\lambda s. rename\ N\ (c\ s)) \mid$   
 $rename\ N\ (Guard\ f\ g\ c) = Guard\ f\ g\ (rename\ N\ c) \mid$   
 $rename\ N\ Throw = Throw \mid$   
 $rename\ N\ (Catch\ c_1\ c_2) = Catch\ (rename\ N\ c_1)\ (rename\ N\ c_2)$

**lemma** *rename-Skip*:  $rename\ h\ c = Skip = (c=Skip)$

**by**  $(cases\ c)\ auto$

**lemma** *rename-Basic*:

$(rename\ h\ c = Basic\ f) = (c=Basic\ f)$

**by**  $(cases\ c)\ auto$

**lemma** *rename-Spec*:

$(rename\ h\ c = Spec\ r) = (c=Spec\ r)$

**by**  $(cases\ c)\ auto$

**lemma** *rename-Seq*:

$(rename\ h\ c = Seq\ rc_1\ rc_2) =$   
 $(\exists\ c_1\ c_2. c = Seq\ c_1\ c_2 \wedge$   
 $rc_1 = rename\ h\ c_1 \wedge rc_2 = rename\ h\ c_2)$

**by**  $(cases\ c)\ auto$

**lemma** *rename-Cond*:

$(rename\ h\ c = Cond\ b\ rc_1\ rc_2) =$   
 $(\exists\ c_1\ c_2. c = Cond\ b\ c_1\ c_2 \wedge rc_1 = rename\ h\ c_1 \wedge rc_2 = rename\ h\ c_2)$

**by**  $(cases\ c)\ auto$

**lemma** *rename-While*:

$(rename\ h\ c = While\ b\ rc') = (\exists\ c'. c = While\ b\ c' \wedge rc' = rename\ h\ c')$

**by**  $(cases\ c)\ auto$

**lemma** *rename-Call*:

$(rename\ h\ c = Call\ q) = (\exists\ p. c = Call\ p \wedge q=h\ p)$

**by**  $(cases\ c)\ auto$

**lemma** *rename-DynCom*:

$(rename\ h\ c = DynCom\ rc) = (\exists\ C. c=DynCom\ C \wedge rc = (\lambda s. rename\ h\ (C\ s)))$

**by**  $(cases\ c)\ auto$

**lemma** *rename-Guard*:

$(rename\ h\ c = Guard\ f\ g\ rc') =$

```

    (∃ c'. c = Guard f g c' ∧ rc' = rename h c')
  by (cases c) auto

lemma rename-Throw:
  (rename h c = Throw) = (c = Throw)
  by (cases c) auto

lemma rename-Catch:
  (rename h c = Catch rc1 rc2) =
    (∃ c1 c2. c = Catch c1 c2 ∧ rc1 = rename h c1 ∧ rc2 = rename h c2)
  by (cases c) auto

lemma exec-rename-to-exec:
  assumes Γ: ∀ p bdy. Γ p = Some bdy → Γ' (h p) = Some (rename h bdy)
  assumes exec: Γ ⊢ ⟨rc, s⟩ ⇒ t
  shows ∧c. rename h c = rc ⇒ ∃ t'. Γ ⊢ ⟨c, s⟩ ⇒ t' ∧ (t' = Stuck ∨ t' = t)
  using exec
  proof (induct)
    case Skip thus ?case by (fastforce intro: exec.intros simp add: rename-Skip)
  next
    case Guard thus ?case by (fastforce intro: exec.intros simp add: rename-Guard)
  next
    case GuardFault thus ?case by (fastforce intro: exec.intros simp add: rename-Guard)
  next
    case FaultProp thus ?case by (fastforce intro: exec.intros)
  next
    case Basic thus ?case by (fastforce intro: exec.intros simp add: rename-Basic)
  next
    case Spec thus ?case by (fastforce intro: exec.intros simp add: rename-Spec)
  next
    case SpecStuck thus ?case by (fastforce intro: exec.intros simp add: rename-Spec)
  next
    case Seq thus ?case by (fastforce intro: exec.intros simp add: rename-Seq)
  next
    case CondTrue thus ?case by (fastforce intro: exec.intros simp add: rename-Cond)
  next
    case CondFalse thus ?case by (fastforce intro: exec.intros simp add: rename-Cond)
  next
    case WhileTrue thus ?case by (fastforce intro: exec.intros simp add: rename-While)
  next
    case WhileFalse thus ?case by (fastforce intro: exec.intros simp add: rename-While)
  next
    case (Call p rbdy s t)
    have rbdy: Γ' p = Some rbdy by fact
    have rename h c = Call p by fact
    then obtain q where c: c = Call q and p: p = h q
    by (auto simp add: rename-Call)
    show ?case
  proof (cases Γ q)

```



```

    case None
    with c show ?thesis by (auto intro: exec.CallUndefined)
next
  case (Some bdy)
  from  $\Gamma$  [rule-format, OF this] p rbdy
  have rename h bdy = rbdy by simp
  with Call.hyps c Some
  show ?thesis
    by (fastforce intro: exec.intros)
qed
next
  case (CallUndefined p s)
  have undef:  $\Gamma' p = \text{None}$  by fact
  have rename h c = Call p by fact
  then obtain q where c:  $c = \text{Call } q$  and p:  $p = h q$ 
    by (auto simp add: rename-Call)
  from undef p  $\Gamma$  have  $\Gamma q = \text{None}$ 
    by (cases  $\Gamma q$ ) auto
  with p c show ?case
    by (auto intro: exec.intros)
next
  case StuckProp thus ?case by (fastforce intro: exec.intros)
next
  case DynCom thus ?case by (fastforce intro: exec.intros simp add: rename-DynCom)
next
  case Throw thus ?case by (fastforce intro: exec.intros simp add: rename-Throw)
next
  case AbruptProp thus ?case by (fastforce intro: exec.intros)
next
  case CatchMatch thus ?case by (fastforce intro: exec.intros simp add: rename-Catch)
next
  case CatchMiss thus ?case by (fastforce intro: exec.intros simp add: rename-Catch)
qed

```

**lemma** *exec-rename-to-exec'*:

```

  assumes  $\Gamma: \forall p \text{ bdy}. \Gamma p = \text{Some } \text{bdy} \longrightarrow \Gamma' (N p) = \text{Some } (\text{rename } N \text{ bdy})$ 
  assumes exec:  $\Gamma \vdash \langle \text{rename } N \text{ c}, s \rangle \Rightarrow t$ 
  shows  $\exists t'. \Gamma \vdash \langle c, s \rangle \Rightarrow t' \wedge (t' = \text{Stuck} \vee t' = t)$ 
  using exec-rename-to-exec [OF  $\Gamma$  exec]
  by auto

```

**lemma** *valid-to-valid-rename*:

```

  assumes  $\Gamma: \forall p \text{ bdy}. \Gamma p = \text{Some } \text{bdy} \longrightarrow \Gamma' (N p) = \text{Some } (\text{rename } N \text{ bdy})$ 
  assumes valid:  $\Gamma \models_{/F} P \text{ c } Q, A$ 
  shows  $\Gamma \models_{/F} P (\text{rename } N \text{ c}) Q, A$ 

```

```

proof (rule validI)
  fix s t
  assume execr:  $\Gamma \vdash \langle \text{rename } N \ c, \text{Normal } s \rangle \Rightarrow t$ 
  assume P:  $s \in P$ 
  assume noFault:  $t \notin \text{Fault } \ ' F$ 
  show  $t \in \text{Normal } \ ' Q \cup \text{Abrupt } \ ' A$ 
  proof -
    from exec-rename-to-exec [OF  $\Gamma$  execr]
    obtain t' where
      exec:  $\Gamma \vdash \langle c, \text{Normal } s \rangle \Rightarrow t'$  and t':  $(t' = \text{Stuck} \vee t' = t)$ 
    by auto
    with valid noFault P show ?thesis
    by (auto simp add: valid-def)
  qed
qed

lemma hoare-to-hoare-rename:
  assumes  $\Gamma: \forall p \ \text{bdy}. \Gamma \ p = \text{Some } \text{bdy} \longrightarrow \Gamma' (N \ p) = \text{Some } (\text{rename } N \ \text{bdy})$ 
  assumes deriv:  $\Gamma, \{\} \vdash_{/F} P \ c \ Q, A$ 
  shows  $\Gamma', \{\} \vdash_{/F} P \ (\text{rename } N \ c) \ Q, A$ 
apply (rule hoare-complete)
apply (insert hoare-sound [OF deriv])
apply (rule valid-to-valid-rename)
apply (rule  $\Gamma$ )
apply (simp add: cvalid-def)
done

lemma hoare-to-hoare-rename':
  assumes  $\Gamma: \forall p \ \text{bdy}. \Gamma \ p = \text{Some } \text{bdy} \longrightarrow \Gamma' (N \ p) = \text{Some } (\text{rename } N \ \text{bdy})$ 
  assumes deriv:  $\forall Z. \Gamma, \{\} \vdash_{/F} (P \ Z) \ c \ (Q \ Z), (A \ Z)$ 
  shows  $\forall Z. \Gamma', \{\} \vdash_{/F} (P \ Z) \ (\text{rename } N \ c) \ (Q \ Z), (A \ Z)$ 
apply rule
apply (rule hoare-to-hoare-rename [OF  $\Gamma$ ])
apply (rule deriv[rule-format])
done

lemma terminates-to-terminates-rename:
  assumes  $\Gamma: \forall p \ \text{bdy}. \Gamma \ p = \text{Some } \text{bdy} \longrightarrow \Gamma' (N \ p) = \text{Some } (\text{rename } N \ \text{bdy})$ 
  assumes termi:  $\Gamma \vdash c \downarrow s$ 
  assumes noStuck:  $\Gamma \vdash \langle c, s \rangle \Rightarrow \notin \{\text{Stuck}\}$ 
  shows  $\Gamma \vdash \text{rename } N \ c \downarrow s$ 
using termi noStuck
proof (induct)
  case Skip thus ?case by (fastforce intro: terminates.intros)
next
  case Basic thus ?case by (fastforce intro: terminates.intros)
next
  case Spec thus ?case by (fastforce intro: terminates.intros)
next

```

```

    case Guard thus ?case by (fastforce intro: terminates.intros
      simp add: final-notin-def exec.intros)
next
  case GuardFault thus ?case by (fastforce intro: terminates.intros)
next
  case Fault thus ?case by (fastforce intro: terminates.intros)
next
  case Seq
  thus ?case
  by (force intro!: terminates.intros exec.intros dest: exec-rename-to-exec [OF  $\Gamma$ ]
    simp add: final-notin-def)
next
  case CondTrue thus ?case by (fastforce intro: terminates.intros
    simp add: final-notin-def exec.intros)
next
  case CondFalse thus ?case by (fastforce intro: terminates.intros
    simp add: final-notin-def exec.intros)
next
  case (WhileTrue s b c)
  have s-in-b:  $s \in b$  by fact
  have noStuck:  $\Gamma \vdash \langle \text{While } b \ c, \text{Normal } s \rangle \Rightarrow \notin \{ \text{Stuck} \}$  by fact
  with s-in-b have  $\Gamma \vdash \langle c, \text{Normal } s \rangle \Rightarrow \notin \{ \text{Stuck} \}$ 
  by (auto simp add: final-notin-def intro: exec.intros)
  with WhileTrue.hyps have  $\Gamma \uparrow \text{rename } N \ c \downarrow \text{Normal } s$ 
  by simp
  moreover
  {
    fix t
    assume exec-rc:  $\Gamma \uparrow \langle \text{rename } N \ c, \text{Normal } s \rangle \Rightarrow t$ 
    have  $\Gamma \uparrow \text{While } b \ (\text{rename } N \ c) \downarrow t$ 
    proof -
      from exec-rename-to-exec [OF  $\Gamma$  exec-rc] obtain t'
      where exec-c:  $\Gamma \vdash \langle c, \text{Normal } s \rangle \Rightarrow t'$  and t':  $(t' = \text{Stuck} \vee t' = t)$ 
      by auto
      with s-in-b noStuck obtain t'=t and  $\Gamma \vdash \langle \text{While } b \ c, t \rangle \Rightarrow \notin \{ \text{Stuck} \}$ 
      by (auto simp add: final-notin-def intro: exec.intros)
      with exec-c WhileTrue.hyps
      show ?thesis
      by auto
    qed
  }
  ultimately show ?case
  using s-in-b
  by (auto intro: terminates.intros)
next
  case WhileFalse thus ?case by (fastforce intro: terminates.intros)
next
  case (Call p bdy s)
  have  $\Gamma \ p = \text{Some } \text{bdy}$  by fact

```

```

from  $\Gamma$  [rule-format, OF this]
have  $bdy'$ :  $\Gamma' (N p) = \text{Some} (\text{rename } N \text{ bdy})$ .
from Call have  $\Gamma \uparrow \text{rename } N \text{ bdy} \downarrow \text{Normal } s$ 
  by (auto simp add: final-notin-def intro: exec.intros)
with  $bdy'$  have  $\Gamma \uparrow \text{Call } (N p) \downarrow \text{Normal } s$ 
  by (auto intro: terminates.intros)
thus ?case by simp
next
  case (CallUndefined p s)
  have  $\Gamma p = \text{None}$   $\Gamma \vdash \langle \text{Call } p, \text{Normal } s \rangle \Rightarrow \notin \{ \text{Stuck} \}$  by fact+
  hence False by (auto simp add: final-notin-def intro: exec.intros)
  thus ?case ..
next
  case Stuck thus ?case by (fastforce intro: terminates.intros)
next
  case DynCom thus ?case by (fastforce intro: terminates.intros
    simp add: final-notin-def exec.intros)
next
  case Throw thus ?case by (fastforce intro: terminates.intros)
next
  case Abrupt thus ?case by (fastforce intro: terminates.intros)
next
  case (Catch c1 s c2)
  have  $\text{noStuck}$ :  $\Gamma \vdash \langle \text{Catch } c1 \ c2, \text{Normal } s \rangle \Rightarrow \notin \{ \text{Stuck} \}$  by fact
  hence  $\Gamma \vdash \langle c1, \text{Normal } s \rangle \Rightarrow \notin \{ \text{Stuck} \}$ 
  by (fastforce simp add: final-notin-def intro: exec.intros)
  with Catch.hyps have  $\Gamma \uparrow \text{rename } N \ c1 \downarrow \text{Normal } s$ 
  by auto
moreover
  {
    fix  $t$ 
    assume  $\text{exec-rc1}$ :  $\Gamma \uparrow \langle \text{rename } N \ c1, \text{Normal } s \rangle \Rightarrow \text{Abrupt } t$ 
    have  $\Gamma \uparrow \text{rename } N \ c2 \downarrow \text{Normal } t$ 
    proof –
      from exec-rename-to-exec [OF  $\Gamma$  exec-rc1] obtain  $t'$ 
        where  $\text{exec-c}$ :  $\Gamma \vdash \langle c1, \text{Normal } s \rangle \Rightarrow t'$  and  $(t' = \text{Stuck} \vee t' = \text{Abrupt } t)$ 
        by auto
      with  $\text{noStuck}$  have  $t': t' = \text{Abrupt } t$ 
        by (fastforce simp add: final-notin-def intro: exec.intros)
      with  $\text{exec-c}$   $\text{noStuck}$  have  $\Gamma \vdash \langle c2, \text{Normal } t \rangle \Rightarrow \notin \{ \text{Stuck} \}$ 
        by (auto simp add: final-notin-def intro: exec.intros)
      with  $\text{exec-c}$   $t'$  Catch.hyps
      show ?thesis
        by auto
    }
  qed
}
ultimately show ?case
  by (auto intro: terminates.intros)
qed

```

**lemma** *validt-to-validt-rename*:  
**assumes**  $\Gamma: \forall p \text{ bdy}. \Gamma \ p = \text{Some bdy} \longrightarrow \Gamma' (N \ p) = \text{Some} (\text{rename } N \ \text{bdy})$   
**assumes** *valid*:  $\Gamma \models_{t/F} P \ c \ Q, A$   
**shows**  $\Gamma' \models_{t/F} P (\text{rename } N \ c) \ Q, A$   
**proof** –  
**from** *valid*  
**have**  $\Gamma' \models_{t/F} P (\text{rename } N \ c) \ Q, A$   
**by** (*auto intro: valid-to-valid-rename [OF  $\Gamma$ ] simp add: validt-def*)  
**moreover**  
{  
**fix**  $s$   
**assume**  $s \in P$   
**with** *valid* **obtain**  $\Gamma \vdash c \downarrow (\text{Normal } s) \ \Gamma \vdash \langle c, \text{Normal } s \rangle \Rightarrow \notin \{\text{Stuck}\}$   
**by** (*auto simp add: validt-def valid-def final-notin-def*)  
**from** *terminates-to-terminates-rename [OF  $\Gamma$  this]*  
**have**  $\Gamma \uparrow \text{rename } N \ c \downarrow \text{Normal } s$   
.  
}  
**ultimately show** *?thesis*  
**by** (*simp add: validt-def*)  
**qed**

**lemma** *hoaret-to-hoaret-rename*:  
**assumes**  $\Gamma: \forall p \text{ bdy}. \Gamma \ p = \text{Some bdy} \longrightarrow \Gamma' (N \ p) = \text{Some} (\text{rename } N \ \text{bdy})$   
**assumes** *deriv*:  $\Gamma, \{\} \vdash_{t/F} P \ c \ Q, A$   
**shows**  $\Gamma', \{\} \vdash_{t/F} P (\text{rename } N \ c) \ Q, A$   
**apply** (*rule hoaret-complete*)  
**apply** (*insert hoaret-sound [OF *deriv*]*)  
**apply** (*rule validt-to-validt-rename*)  
**apply** (*rule  $\Gamma$* )  
**apply** (*simp add: cvalidt-def*)  
**done**

**lemma** *hoaret-to-hoaret-rename'*:  
**assumes**  $\Gamma: \forall p \text{ bdy}. \Gamma \ p = \text{Some bdy} \longrightarrow \Gamma' (N \ p) = \text{Some} (\text{rename } N \ \text{bdy})$   
**assumes** *deriv*:  $\forall Z. \Gamma, \{\} \vdash_{t/F} (P \ Z) \ c \ (Q \ Z), (A \ Z)$   
**shows**  $\forall Z. \Gamma', \{\} \vdash_{t/F} (P \ Z) (\text{rename } N \ c) \ (Q \ Z), (A \ Z)$   
**apply** *rule*  
**apply** (*rule hoaret-to-hoaret-rename [OF  $\Gamma$ ]*)  
**apply** (*rule deriv[rule-format]*)  
**done**

**lemma** *lift<sub>c</sub>-whileAnno [simp]*: *lift<sub>c</sub> prj inject (whileAnno b I V c) = whileAnno (lift<sub>s</sub> prj b) (lift<sub>s</sub> prj I) (lift<sub>r</sub> prj inject V) (lift<sub>c</sub> prj inject c)*  
**by** (*simp add: whileAnno-def*)

**lemma** *lift<sub>c</sub>-block* [*simp*]: *lift<sub>c</sub> prj inject (block init bdy return c) =*  
*block (lift<sub>f</sub> prj inject init) (lift<sub>c</sub> prj inject bdy)*  
*(λs. (lift<sub>f</sub> prj inject (return (prj s))))*  
*(λs t. lift<sub>c</sub> prj inject (c (prj s) (prj t)))*  
**by** (*simp add: block-def block-exn-def*)

**lemma** *lift<sub>c</sub>-call* [*simp*]: *lift<sub>c</sub> prj inject (call init p return c) =*  
*call (lift<sub>f</sub> prj inject init) p*  
*(λs. (lift<sub>f</sub> prj inject (return (prj s))))*  
*(λs t. lift<sub>c</sub> prj inject (c (prj s) (prj t)))*  
**by** (*simp add: call-def lift<sub>c</sub>-block*)

**lemma** *rename-whileAnno* [*simp*]: *rename h (whileAnno b I V c) =*  
*whileAnno b I V (rename h c)*  
**by** (*simp add: whileAnno-def*)

**lemma** *rename-block* [*simp*]: *rename h (block init bdy return c) =*  
*block init (rename h bdy) return (λs t. rename h (c s t))*  
**by** (*simp add: block-def block-exn-def*)

**lemma** *rename-call* [*simp*]: *rename h (call init p return c) =*  
*call init (h p) return (λs t. rename h (c s t))*  
**by** (*simp add: call-def*)

**end**

**theory** *ComposeEx* **imports** *Compose ../Vcg ../HeapList* **begin**

**record** *globals-list* =  
*next-'* :: *ref* ⇒ *ref*

**record** *state-list* = *globals-list state* +  
*p-'* :: *ref*  
*sl-q-'* :: *ref*  
*r-'* :: *ref*

**procedures** *Rev(p|sl-q)* =  
*'sl-q ::= Null;;*  
*WHILE 'p ≠ Null*  
*DO*  
*'r ::= 'p;; { 'p ≠ Null } ⟶ 'p ::= 'p → 'next;;*  
*{ 'r ≠ Null } ⟶ 'r → 'next ::= 'sl-q;; 'sl-q ::= 'r*  
*OD*

**print-theorems**

**lemma** (in *Rev-impl*)

*Rev-modifies:*

$\forall \sigma. \Gamma \vdash /UNIV \{ \sigma \} \text{'sl-q} ::= PROC \text{Rev}(\text{'p}) \{ t. t \text{ may-only-modify-globals } \sigma \text{ in } [next] \}$

**apply** (hoare-rule *HoarePartial.ProcNoRec1*)

**apply** (vcg spec=*modifies*)

**done**

**lemma** (in *Rev-impl*) **shows**

*Rev-spec:*

$\forall Ps. \Gamma \vdash \{ List \text{'p} \text{'next} Ps \} \text{'sl-q} ::= PROC \text{Rev}(\text{'p}) \{ List \text{'sl-q} \text{'next} (rev Ps) \}$

**apply** (hoare-rule *HoarePartial.ProcNoRec1*)

**apply** (hoare-rule *anno* =

$\text{'sl-q} ::= Null;$

$WHILE \text{'p} \neq Null \text{ INV } \{ \exists Ps' Qs'. List \text{'p} \text{'next} Ps' \wedge List \text{'sl-q} \text{'next} Qs' \wedge set Ps' \cap set Qs' = \{ \} \wedge rev Ps' @ Qs' = rev Ps \}$

*DO*

$\text{'r} ::= \text{'p}; \{ \text{'p} \neq Null \} \mapsto \text{'p} ::= \text{'p} \rightarrow \text{'next};$

$\{ \text{'r} \neq Null \} \mapsto \text{'r} \rightarrow \text{'next} ::= \text{'sl-q}; \text{'sl-q} ::= \text{'r}$

*OD* in *HoarePartial.annotate1*)

**apply** *vcg*

**apply** *clarsimp*

**apply** *fastforce*

**apply** *clarsimp*

**done**

**declare** [[*names-unique* = *false*]]

**record** *globals* =

*strnext-'* :: *ref*  $\Rightarrow$  *ref*

*chr-'* :: *ref*  $\Rightarrow$  *char*

*qnext-'* :: *ref*  $\Rightarrow$  *ref*

*cont-'* :: *ref*  $\Rightarrow$  *int*

**record** *state* = *globals state* +

*str-'* :: *ref*

*queue-'* :: *ref*

*q-'* :: *ref*

*r-'* :: *ref*

**definition** *project-globals-str*:: *globals*  $\Rightarrow$  *globals-list*

**where** *project-globals-str g* = (*next-'* = *strnext-'* *g*)

**definition** *project-str*:: *state*  $\Rightarrow$  *state-list*

**where**

*project-str*  $s =$   
 $(\text{globals} = \text{project-globals-str } (\text{globals } s),$   
 $\text{state-list.p}' = \text{str}' s, \text{sl-q}' = \text{q}' s, \text{state-list.r}' = \text{r}' s)$

**definition** *inject-globals-str*:  
 $\text{globals} \Rightarrow \text{globals-list} \Rightarrow \text{globals}$   
**where**

$\text{inject-globals-str } G g =$   
 $G(\text{strnext}' := \text{next}' g)$

**definition** *inject-str*:  $\text{state} \Rightarrow \text{state-list} \Rightarrow \text{state}$  **where**  
 $\text{inject-str } S s = S(\text{globals} := \text{inject-globals-str } (\text{globals } S) (\text{globals } s),$   
 $\text{str}' := \text{state-list.p}' s, \text{q}' := \text{sl-q}' s,$   
 $\text{r}' := \text{state-list.r}' s)$

**lemma** *globals-inject-project-str-commutes*:  
 $\text{inject-globals-str } G (\text{project-globals-str } G) = G$   
**by** (*simp add: inject-globals-str-def project-globals-str-def*)

**lemma** *inject-project-str-commutes*:  $\text{inject-str } S (\text{project-str } S) = S$   
**by** (*simp add: inject-str-def project-str-def globals-inject-project-str-commutes*)

**lemma** *globals-project-inject-str-commutes*:  
 $\text{project-globals-str } (\text{inject-globals-str } G g) = g$   
**by** (*simp add: inject-globals-str-def project-globals-str-def*)

**lemma** *project-inject-str-commutes*:  $\text{project-str } (\text{inject-str } S s) = s$   
**by** (*simp add: inject-str-def project-str-def globals-project-inject-str-commutes*)

**lemma** *globals-inject-str-last*:  
 $\text{inject-globals-str } (\text{inject-globals-str } G g) g' = \text{inject-globals-str } G g'$   
**by** (*simp add: inject-globals-str-def*)

**lemma** *inject-str-last*:  
 $\text{inject-str } (\text{inject-str } S s) s' = \text{inject-str } S s'$   
**by** (*simp add: inject-str-def globals-inject-str-last*)

**definition**  
 $\text{lift}_e = (\lambda \Gamma p. \text{map-option } (\text{lift}_c \text{ project-str inject-str}) (\Gamma p))$

**print-locale** *lift-state-space*

**interpretation** *ex: lift-state-space project-str inject-str*

$\text{xstate-map project-str lift}_e \text{ lift}_c \text{ project-str inject-str}$

$\text{lift}_f \text{ project-str inject-str lift}_s \text{ project-str}$

$\text{lift}_r \text{ project-str inject-str}$

**apply**  $-$

**apply**  $(\text{rule lift-state-space.intro})$

**apply**  $(\text{rule project-inject-str-commutes})$

**apply** *simp*

**apply** *simp*



```

apply (simp add: lifte-def)
apply simp
apply simp
apply simp
done

```

**interpretation** *ex: lift-state-space-ext project-str inject-str*  
*xstate-map project-str lift<sub>e</sub> lift<sub>c</sub> project-str inject-str*  
*lift<sub>f</sub> project-str inject-str lift<sub>s</sub> project-str*  
*lift<sub>r</sub> project-str inject-str*

```

apply –
apply intro-locales [1]
  apply (rule lift-state-space-ext-axioms.intro)
  apply (rule inject-project-str-commutes)
  apply (rule inject-str-last)
apply (simp-all add: lifte-def)
done

```

**lemmas** *Rev-lift-spec = ex.lift-hoarep' [OF Rev-impl.Rev-spec,simplified lift<sub>s</sub>-def*  
*project-str-def project-globals-str-def,simplified, of - "Rev"]*  
**print-theorems**

**definition**  $\mathcal{N} \ p' \ p = (\text{if } p = \text{"Rev"} \text{ then } p' \text{ else ""})$

**procedures** *RevStr(str|q) = rename (N RevStr-'proc)*  
*(lift<sub>c</sub> project-str inject-str (Rev-body.Rev-body))*

**lemmas** *Rev-lift-spec' =*  
*Rev-lift-spec [of ["Rev" ↦ Rev-body.Rev-body] ,*  
*simplified Rev-impl-def Rev-clique-def,simplified]*  
**thm** *Rev-lift-spec'*

**lemma** *Rev-lift-spec'':*  
 $\forall Ps. \text{lift}_e \ [ \text{"Rev"} \mapsto \text{Rev-body.Rev-body} ]$   
 $\vdash \{ \text{List } 'str \ 'strnext \ Ps \} \text{ Call } \text{"Rev"} \{ \text{List } 'q \ 'strnext \ (\text{rev } Ps) \}$   
**by** (rule *Rev-lift-spec'*)

**lemma** (in *RevStr-impl*) *N-ok:*  
 $\forall p \ bdy. (\text{lift}_e \ [ \text{"Rev"} \mapsto \text{Rev-body.Rev-body} ]) \ p = \text{Some } bdy \longrightarrow$   
 $\Gamma \ (\mathcal{N} \ \text{RevStr-'proc } p) = \text{Some } (\text{rename } (\mathcal{N} \ \text{RevStr-'proc}) \ bdy)$

```

apply (insert RevStr-impl)
apply (auto simp add: RevStr-body-def lifte-def N-def)
done

context RevStr-impl
begin
  thm hoare-to-hoare-rename' [OF - Rev-lift-spec'', OF N-ok,
    simplified N-def, simplified ]
end

lemmas (in RevStr-impl) RevStr-spec =
  hoare-to-hoare-rename' [OF - Rev-lift-spec'', OF N-ok,
    simplified N-def, simplified ]

lemma (in RevStr-impl) RevStr-spec':
 $\forall Ps. \Gamma \vdash \{ \text{List } 'str \text{ } 'strnext \ Ps \} 'q ::= PROC \text{ RevStr}('str)$ 
 $\{ \text{List } 'q \text{ } 'strnext (rev Ps) \}$ 
  by (rule RevStr-spec)

lemmas Rev-modifies' =
  Rev-impl.Rev-modifies [of ["Rev"  $\mapsto$  Rev-body.Rev-body], simplified Rev-impl-def,
    simplified]
thm Rev-modifies'

context RevStr-impl
begin
lemmas RevStr-modifies' =
  hoare-to-hoare-rename' [OF - ex.hoare-lift-modifies' [OF Rev-modifies'],
    OF N-ok, of "Rev", simplified N-def Rev-clique-def, simplified]
end

lemma (in RevStr-impl) RevStr-modifies:
 $\forall \sigma. \Gamma \vdash /_{UNIV} \{ \sigma \} 'str ::= PROC \text{ RevStr}('str)$ 
 $\{ t. t \text{ may-only-modify-globals } \sigma \text{ in } [strnext] \}$ 
apply (rule allI)
apply (rule HoarePartialProps.ConseqMGT [OF RevStr-modifies'])
apply (clarsimp simp add:
  lifts-def mex-def meq-def
  project-str-def inject-str-def project-globals-str-def inject-globals-str-def)
apply blast
done

end

```

## 26 User Guide

We introduce the verification environment with a couple of examples that illustrate how to use the different bits and pieces to verify programs.

### 26.1 Basics

First of all we have to decide how to represent the state space. There are currently two implementations. One is based on records the other one on the concept called ‘statespace’ that was introduced with Isabelle 2007 (see `HOL/Statespace`). In contrast to records a ‘statespace’ does not define a new type, but provides a notion of state, based on locales. Logically the state is modelled as a function from (abstract) names to (abstract) values and the statespace infrastructure organises distinctness of names and projection/injection of concrete values into the abstract one. Towards the user the interface of records and statespaces is quite similar. However, statespaces offer more flexibility, inherited from the locale infrastructure, in particular multiple inheritance and renaming of components.

In this user guide we prefer statespaces, but give some comments on the usage of records in Section 26.9.

**hoarestate** *vars* =

```
A :: nat
I :: nat
M :: nat
N :: nat
R :: nat
S :: nat
```

The command **hoarestate** is a simple preprocessor for the command **statespaces** which decorates the state components with the suffix `-'`, to avoid cluttering the namespace. Also note that underscores are printed as hyphens in this documentation. So what you see as `A-'` in this document is actually `A_'`. Every component name becomes a fixed variable in the locale *vars* and can no longer be used for logical variables.

Lookup of a component `A-'` in a state *s* is written as `s.A-'`, and update with a value *term v* as `s⟨A-′ := v⟩`.

To deal with local and global variables in the context of procedures the program state is organised as a record containing the two components *locals* and *globals*. The variables defined in **hoarestate vars** reside in the *locals* part.

Here is a first example.

```
lemma (in vars)  $\Gamma \vdash \{\!| N = 5 \!\} N ::= 2 * N \{\!| N = 10 \!\}$ 
apply vcg
```

1.  $\bigwedge N. N = 5 \implies 2 * N = 10$

**apply** *simp*

*No subgoals!*

**done**

We enable the locale of statespace *vars* by the **in vars** directive. The verification condition generator is invoked via the *vcg* method and leaves us with the expected subgoal that can be proved by simplification.

If we refer to components (variables) of the state-space of the program we always mark these with  $\acute{}$  (in assertions and also in the program itself). It is the acute-symbol and is present on most keyboards. The assertions of the Hoare tuple are ordinary Isabelle sets. As we usually want to refer to the state space in the assertions, we provide special brackets for them. They can be written as  $\{ | \}$  in ASCII or  $\{ \} \}$  with symbols. Internally, marking variables has two effects. First of all we refer to the implicit state and secondary we get rid of the suffix  $\acute{}$ . So the assertion  $\{ \acute{N} = 5 \}$  internally gets expanded to  $\{ s. locals\ s \cdot N\acute{'} = 5 \}$  written in ordinary set comprehension notation of Isabelle. It describes the set of states where the  $N\acute{}$  component is equal to 5. An empty context and an empty postcondition for abrupt termination can be omitted. The lemma above is a shorthand for  $\Gamma, \{ \} \vdash \{ \acute{N} = 5 \} \acute{N} ::= 2 * \acute{N} \{ \acute{N} = 10 \}, \{ \}$ .

We can step through verification condition generation by the method *vcg-step*.

**lemma** (**in vars**)  $\Gamma, \{ \} \vdash \{ \acute{N} = 5 \} \acute{N} ::= 2 * \acute{N} \{ \acute{N} = 10 \}$   
**apply** *vcg-step*

1.  $\{ \acute{N} = 5 \} \subseteq \{ 2 * \acute{N} = 10 \}$

The last step of verification condition generation, transforms the inclusion of state sets to the corresponding predicate on components of the state space.

**apply** *vcg-step*

1.  $\bigwedge N. N = 5 \implies 2 * N = 10$

**by** *simp*

Although our assertions work semantically on the state space, stepping through verification condition generation “feels” like the expected syntactic substitutions of traditional Hoare logic. This is achieved by light simplification on the assertions calculated by the Hoare rules.

**lemma** (**in vars**)  $\Gamma \vdash \{ \acute{N} = 5 \} \acute{N} ::= 2 * \acute{N} \{ \acute{N} = 10 \}$   
**apply** (*rule HoarePartial.Basic*)

1.  $\{ \acute{N} = 5 \} \subseteq \{ s \mid locals\text{-update } (update\ project\text{-Nat-nat } inject\text{-Nat-nat } N\acute{'} (K\text{-statefun } (2 * (locals\ s \cdot N\acute{'})))) \mid s \in \{ \acute{N} = 10 \} \}$

**apply** (*simp only: mem-Collect-eq*)

1.  $\{\!| \text{'N} = 5 \!\!\} \subseteq \{s \mid \text{locals } (\text{locals-update } (\text{update project-Nat-nat inject-Nat-nat } N'\text{-' } (K\text{-statefun } (2 * (\text{locals } s\text{-}N'\text{-'})))) s)\text{-}N'\text{-'} = 10\}$

**apply** (*tactic*

$\langle \text{Hoare.BasicSimpTac } @\{\text{context}\} \text{ Hoare.Function false}$

$\square (K \text{ all-tac } 1 \rangle)$

1.  $\{\!| \text{'N} = 5 \!\!\} \subseteq \{\!| 2 * \text{'N} = 10 \!\!\}$

**by** *simp*

The next example shows how we deal with the while loop. Note the invariant annotation.

**lemma** (*in vars*)

$\Gamma, \{\!| \vdash \{\!| \text{'M} = 0 \wedge \text{'S} = 0 \!\!\}$

$\text{WHILE } \text{'M} \neq a$

$\text{INV } \{\!| \text{'S} = \text{'M} * b \!\!\}$

$\text{DO } \text{'S} ::= \text{'S} + b;; \text{'M} ::= \text{'M} + 1 \text{ OD}$

$\{\!| \text{'S} = a * b \!\!\}$

**apply** *vcg*

1.  $\bigwedge M S. \llbracket M = 0; S = 0 \rrbracket \implies S = M * b$
2.  $\bigwedge M S. \llbracket S = M * b; M \neq a \rrbracket \implies S + b = (M + 1) * b$
3.  $\bigwedge M S. \llbracket S = M * b; \neg M \neq a \rrbracket \implies S = a * b$

The verification condition generator gives us three proof obligations, stemming from the path from the precondition to the invariant, from the invariant together with loop condition through the loop body to the invariant, and finally from the invariant together with the negated loop condition to the postcondition.

**apply** *auto*

**done**

## 26.2 Procedures

### 26.2.1 Declaration

Our first procedure is a simple square procedure. We provide the command **procedures**, to declare and define a procedure.

**procedures**

*Square* ( $N::\text{nat} \mid R::\text{nat}$ )

**where**  $I::\text{nat}$  **in**

$\text{'R} ::= \text{'N} * \text{'N}$

A procedure is given by the signature of the procedure followed by the procedure body. The signature consists of the name of the procedure and a list of parameters together with their types. The parameters in front of the

pipe | are value parameters and behind the pipe are the result parameters. Value parameters model call by value semantics. The value of a result parameter at the end of the procedure is passed back to the caller. Local variables follow the *where*. If there are no local variables the *where ... in* can be omitted. The variable *I* is actually unused in the body, but is used in the examples below.

The `procedures` command provides convenient syntax for procedure calls (that creates the proper *init*, *return* and *result* functions on the fly) and creates locales and statespaces to reason about the procedure. The purpose of locales is to set up logical contexts to support modular reasoning. Locales can be seen as freeze-dried proof contexts that get alive as you setup a new lemma or theorem ([2]). The locale the user deals with is named *Square-impl*. It defines the procedure name (internally *Square-'proc*), the procedure body (named *Square-body*) and the statespaces for parameters and local and global variables. Moreover it contains the assumption  $\Gamma$  *Square-'proc* = *Some Square-body*, which states that the procedure is properly defined in the procedure context.

The purpose of the locale is to give us easy means to setup the context in which we prove programs correct. In this locale the procedure context  $\Gamma$  is fixed. So we always use this letter for the procedure specification. This is crucial, if we prove programs under the assumption of some procedure specifications.

The `procedures` command generates syntax, so that we can either write `CALL Square('I,'R)` or `'I := CALL Square()` for the procedure call. The internal term is the following:

```
call ( $\lambda s. s(\text{locals} := \text{locals } s\langle N\text{'Square-}' := \text{locals } s\cdot I\text{'Square-}'\rangle)$ )
  Square-'proc ( $\lambda s t. s(\text{globals} := \text{globals } t)$ )
  ( $\lambda i t. 'R := \text{locals } t\cdot R\text{'Square-}'$ )
```

Note the additional decoration (with the procedure name) of the parameter and local variable names.

The abstract syntax for the procedure call is `call init p return result`. The *init* function copies the values of the actual parameters to the formal parameters, the *return* function copies the global variables back (in our case there are no global variables), and the *result* function additionally copies the values of the formal result parameters to the actual locations. Actual value parameters can be all kind of expressions, since we only need their value. But result parameters must be proper “lvalues”: variables (including dereferenced pointers) or array locations, since we have to assign values to them.

## 26.2.2 Verification

A procedure specification is an ordinary Hoare tuple. We use the parameterless call for the specification;  $\text{'}R ::= \text{PROC Square}(\text{'}N)$  is syntactic sugar for *Call Square-'proc*. This emphasises that the specification describes the internal behaviour of the procedure, whereas parameter passing corresponds to the procedure call. The following precondition fixes the current value  $\text{'}N$  to the logical variable  $n$ . Universal quantification of  $n$  enables us to adapt the specification to an actual parameter. The specification is used in the rule for procedure call when we come upon a call to *Square*. Thus  $n$  plays the role of the auxiliary variable  $Z$ .

To verify the procedure we need to verify the body. We use a derived variant of the general recursion rule, tailored for non recursive procedures: *HoarePartial.ProcNoRec1*:

$$\llbracket \forall Z. \Gamma, \Theta \vdash_{/F} (P\ Z) \text{ the } (\Gamma\ p)\ (Q\ Z), (A\ Z); p \in \text{dom } \Gamma \rrbracket \implies \forall Z. \Gamma, \Theta \vdash_{/F} (P\ Z)\ \text{Call } p\ (Q\ Z), (A\ Z)$$

The naming convention for the rule is the following: The *1* expresses that we look at one procedure, and *NoRec* that the procedure is non recursive.

**lemma** (in *Square-impl*)

**shows**  $\forall n. \Gamma \vdash \{\text{'}N = n\}\ \text{'}R ::= \text{PROC Square}(\text{'}N)\ \{\text{'}R = n * n\}$

The directive *in* has the effect that the context of the locale *Square-impl* is included to the current lemma, and that the lemma is added as a fact to the locale, after it is proven. The next time locale *Square-impl* is invoked this lemma is immediately available as fact, which the verification condition generator can use.

**apply** (*hoare-rule HoarePartial.ProcNoRec1*)

$$1. \forall n. \Gamma \vdash \{\text{'}N = n\}\ \text{'}R ::= \text{'}N * \text{'}N\ \{\text{'}R = n * n\}$$

The method *hoare-rule*, like *rule* applies a single rule, but additionally does some “obvious” steps: It solves the canonical side-conditions of various Hoare-rules and it automatically expands the procedure body: With *Square-impl*:  $\Gamma\ \text{Square-}'proc = \text{Some Square-body}$  we get the procedure body out of the procedure context  $\Gamma$ ; with *Square-body-def*:  $\text{Square-body} \equiv \text{'}R ::= \text{'}N * \text{'}N$  we can unfold the definition of the body.

The proof is finished by the *vcg* and *simp*.

$$1. \forall n. \Gamma \vdash \{\text{'}N = n\}\ \text{'}R ::= \text{'}N * \text{'}N\ \{\text{'}R = n * n\}$$

**by** *vcg simp*

If the procedure is non recursive and there is no specification given, the verification condition generator automatically expands the body.

**lemma** (in *Square-impl*) *Square-spec*:  
**shows**  $\forall n. \Gamma \vdash \{N = n\} \quad 'R ::= PROC\ Square('N) \{R = n * n\}$   
**by** *vcg simp*

An important naming convention is to name the specification as  $\langle procedure-name \rangle\text{-spec}$ . The verification condition generator refers to this name in order to search for a specification in the theorem database.

### 26.2.3 Usage

Let us see how we can use procedure specifications.

**lemma** (in *Square-impl*)  
**shows**  $\Gamma \vdash \{I = 2\} \quad 'R ::= CALL\ Square('I) \{R = 4\}$

Remember that we have already proven *Square-spec* in the locale *Square-impl*. This is crucial for verification condition generation. When reaching a procedure call, it looks for the specification (by its name) and applies the rule *HoarePartial.ProcSpec* instantiated with the specification (as last premise). Before we apply the verification condition generator, let us take some time to think of what we can expect. Let's look at the specification *Square-spec* again:

$$\forall n. \Gamma \vdash \{N = n\} \quad 'R ::= PROC\ Square('N) \{R = n * n\}$$

The specification talks about the formal parameters  $N$  and  $R$ . The precondition  $\{N = n\}$  just fixes the initial value of  $N$ . The actual parameters are  $I$  and  $R$ . We have to adapt the specification to this calling context.  $\forall n. \Gamma \vdash \{I = n\} \quad 'R ::= CALL\ Square() \{R = n * n\}$ . From the postcondition  $\{R = n * n\}$  we have to derive the actual postcondition  $\{R = 4\}$ . So we gain something like:  $\{n * n = 4\}$ . The precondition is  $\{I = 2\}$  and the specification tells us  $\{I = n\}$  for the pre-state. So the value of  $n$  is the value of  $I$  in the pre-state. So we arrive at  $\{I = 2\} \subseteq \{I * I = 4\}$ .

**apply** *vcg-step*

1.  $\{I = 2\} \subseteq \{\forall t. {}^tR = I * I \longrightarrow I * I = 4\}$

The second set looks slightly more involved:  $\{\forall t. {}^tR = I * I \longrightarrow I * I = 4\}$ , this is an artefact from the procedure call rule. Originally  $I * I = 4$  was  ${}^tR = 4$ . Where  $t$  denotes the final state of the procedure and the superscript notation allows to select a component from a particular state.

**apply** *vcg-step*

1.  $\bigwedge I. I = 2 \implies \forall R. R = I * I \longrightarrow I * I = 4$

**by** *simp*



The adaption of the procedure specification to the actual calling context is done due to the *init*, *return* and *result* functions in the rule *HoarePartial.ProcSpec* (or in the variant *HoarePartial.ProcSpecNoAbrupt* which already incorporates the fact that the postcondition for abrupt termination is the empty set). For the readers interested in the internals, here a version without *vcg*.

**lemma** (in *Square-impl*)

**shows**  $\Gamma \vdash \{\!| T = 2 \!\!\} \text{'R} ::= \text{CALL Square}(\text{'T}) \{\!| \text{'R} = 4 \!\!\}$   
**apply** (rule *HoarePartial.ProcSpecNoAbrupt* [*OF - - Square-spec*])

1.  $\{\!| T = 2 \!\!\}$   
 $\subseteq \{s \mid \exists Z. s(\text{locals} := \text{locals } s \langle N\text{'Square-}' := \text{locals } s \cdot I\text{'Square-}' \rangle)$   
 $\quad \in \{\!| N = Z \!\!\} \wedge$   
 $\quad (\forall t. t \in \{\!| \text{'R} = Z * Z \!\!\} \longrightarrow s(\text{globals} := \text{globals } t) \in ?R \text{ } s \text{ } t)\}$
2.  $\forall s \text{ } t. \Gamma \vdash (?R \text{ } s \text{ } t) \text{'R} ::= \text{locals } t \cdot R\text{'Square-}' \{\!| \text{'R} = 4 \!\!\}$

This is the raw verification condition, It is interesting to see how the auxiliary variable *Z* is actually used. It is unified with *n* of the specification and fixes the state after parameter passing.

**apply** *simp*

1.  $\{\!| T = 2 \!\!\}$   
 $\subseteq \{s \mid \forall t. \text{locals } t \cdot R\text{'Square-}' =$   
 $\quad (\text{locals } s \cdot I\text{'Square-}') * (\text{locals } s \cdot I\text{'Square-}') \longrightarrow$   
 $\quad s(\text{globals} := \text{globals } t) \in ?R \text{ } s \text{ } t\}$
2.  $\forall s \text{ } t. \Gamma \vdash (?R \text{ } s \text{ } t) \text{'R} ::= \text{locals } t \cdot R\text{'Square-}' \{\!| \text{'R} = 4 \!\!\}$

**prefer** 2

**apply** *vcg-step*

1.  $\{\!| T = 2 \!\!\}$   
 $\subseteq \{s \mid \forall t. \text{locals } t \cdot R\text{'Square-}' =$   
 $\quad (\text{locals } s \cdot I\text{'Square-}') * (\text{locals } s \cdot I\text{'Square-}') \longrightarrow$   
 $\quad s(\text{globals} := \text{globals } t) \in \{\!| \text{'R} = 4 \!\!\}\}$

**apply** (*auto intro: ext*)

**done**

## 26.2.4 Recursion

We want to define a procedure for the factorial. We first define a HOL function that calculates it, to specify the procedure later on.

**primrec** *fac*:: *nat*  $\Rightarrow$  *nat*

**where**

$$\begin{aligned} \text{fac } 0 &= 1 \mid \\ \text{fac } (\text{Suc } n) &= (\text{Suc } n) * \text{fac } n \end{aligned}$$

Now we define the procedure.

**procedures**

$$\begin{aligned} & \text{Fac } (N::\text{nat} \mid R::\text{nat}) \\ & \text{IF } 'N = 0 \text{ THEN } 'R ::= 1 \\ & \text{ELSE } 'R ::= \text{CALL Fac}('N - 1); \\ & \quad 'R ::= 'N * 'R \\ & \text{FI} \end{aligned}$$

Now let us prove that our implementation of *Fac* meets its specification.

**lemma** (in *Fac-impl*)

**shows**  $\forall n. \Gamma \vdash \{\!| N = n \!|\} 'R ::= \text{PROC Fac}('N) \{\!| R = \text{fac } n \!|\}$

**apply** (*hoare-rule HoarePartial.ProcRec1*)

$$\begin{aligned} 1. \forall n. \Gamma, (\bigcup_n \{(\{\!| N = n \!|\}, \text{Fac-}'\text{proc}, \{\!| R = \text{fac } n \!|\}, \emptyset)\}) \\ \quad \vdash \{\!| N = n \!|\} \\ \quad \text{IF } 'N = 0 \text{ THEN } 'R ::= 1 \text{ ELSE } 'R ::= \text{CALL Fac}(); 'R ::= 'N \\ * 'R \\ \quad \text{FI} \\ \quad \{\!| R = \text{fac } n \!|\} \end{aligned}$$

**apply** *vcg*

$$\begin{aligned} 1. \bigwedge N. (N = 0 \longrightarrow 1 = \text{fac } N) \wedge \\ (N \neq 0 \longrightarrow (\forall R. R = \text{fac } (N - 1) \longrightarrow N * \text{fac } (N - 1) = \text{fac } N)) \end{aligned}$$

**apply** *simp*

**done**

Since the factorial is implemented recursively, the main ingredient of this proof is, to assume that the specification holds for the recursive call of *Fac* and prove the body correct. The assumption for recursive calls is added to the context by the rule *HoarePartial.ProcRec1* (also derived from the general rule for mutually recursive procedures):

$$\begin{aligned} \llbracket \forall Z. \Gamma, \Theta \cup (\bigcup_Z \{(P \ Z, p, Q \ Z, A \ Z)\}) \vdash_{/F} (P \ Z) \text{ the } (\Gamma \ p) (Q \ Z), (A \ Z); \\ p \in \text{dom } \Gamma \rrbracket \Longrightarrow \forall Z. \Gamma, \Theta \vdash_{/F} (P \ Z) \text{ Call } p (Q \ Z), (A \ Z) \end{aligned}$$

The verification condition generator infers the specification out of the context  $\Theta$  when it encounters a recursive call of the factorial.

### 26.3 Global Variables and Heap

Now we define and verify some procedures on heap-lists. We consider list structures consisting of two fields, a content element *cont* and a reference to the next list element *next*. We model this by the following state space where every field has its own heap.

```
hoarestate globals-heap =
  next :: ref ⇒ ref
  cont :: ref ⇒ nat
```

It is mandatory to start the state name with ‘globals’. This is exploited by the syntax translations to store the components in the *globals* part of the state.

Updates to global components inside a procedure are always propagated to the caller. This is implicitly done by the parameter passing syntax translations.

We first define an append function on lists. It takes two references as parameters. It appends the list referred to by the first parameter with the list referred to by the second parameter. The statespace of the global variables has to be imported.

```
procedures (imports globals-heap)
  append(p :: ref, q::ref | p::ref)
    IF p=Null THEN p ::= q
    ELSE p→next ::= CALL append(p→next,q) FI
```

The difference of a global and a local variable is that global variables are automatically copied back to the procedure caller. We can study this effect on the translation of *p* ::= **CALL** *append*():

```
call
  (λs. s(locals := locals s(p-'append-' := locals s.p-'append-',
    q-'append-' := locals s.q-'append-'))
  append-'proc (λs t. s(globals := globals t))
  (λi t. ip ::= locals t.p-'append-')
```

Below we give two specifications this time. One captures the functional behaviour and focuses on the entities that are potentially modified by the procedure, the second one is a pure frame condition.

The functional specification below introduces two logical variables besides the state space variable  $\sigma$ , namely  $P$ s and  $Q$ s. They are universally quantified and range over both the pre-and the postcondition, so that we are able to properly instantiate the specification during the proofs. The syntax  $\{\sigma$ .

$\dots\}$  is a shorthand to fix the current state:  $\{s. \sigma = s \dots\}$ . Moreover  $\sigma_x$  abbreviates the lookup of variable  $x$  in the state  $\sigma$ .

The approach to specify procedures on lists basically follows [5]. From the pointer structure in the heap we (relationally) abstract to HOL lists of references. Then we can specify further properties on the level of HOL lists, rather than on the heap. The basic abstractions are:

$$\begin{aligned} Path\ x\ h\ y\ [] &= (x = y) \\ Path\ x\ h\ y\ (p \cdot ps) &= (x = p \wedge x \neq Null \wedge Path\ (h\ x)\ h\ y\ ps) \end{aligned}$$

$Path\ (x::ref)\ (h::ref \Rightarrow ref)\ (y::ref)\ (ps::ref\ list)$ :  $ps$  is a list of references that we can obtain out of the heap  $h$  by starting with the reference  $x$ , following the references in  $h$  up to the reference  $y$ .

$$List\ p\ h\ ps = Path\ p\ h\ Null\ ps$$

A list  $List\ p\ h\ ps$  is a path starting in  $p$  and ending up in  $Null$ .

**lemma** (in *append-impl*) *append-spec1*:

**shows**  $\forall \sigma\ Ps\ Qs.$

$$\Gamma \vdash \{\sigma. List\ 'p\ 'next\ Ps \wedge List\ 'q\ 'next\ Qs \wedge set\ Ps \cap set\ Qs = \{\}\}$$

$$'p ::= PROC\ append('p, 'q)$$

$$\{\{List\ 'p\ 'next\ (Ps @ Qs) \wedge (\forall x. x \notin set\ Ps \longrightarrow 'next\ x = \sigma_{next}\ x)\}\}$$

**apply** (*hoare-rule* *HoarePartial.ProcRec1*)

1.  $\forall \sigma\ Ps\ Qs.$

$$\Gamma, (\bigcup_{\sigma} Ps\ Qs$$

$$\{\{\sigma. List\ 'p\ 'next\ Ps \wedge List\ 'q\ 'next\ Qs \wedge set\ Ps \cap set\ Qs = \{\}\},$$

*append-'proc*,

$$\{\{List\ 'p\ 'next\ (Ps @ Qs) \wedge$$

$$(\forall x. x \notin set\ Ps \longrightarrow x \rightarrow 'next = \sigma_{next}\ x)\}\},$$

$\emptyset\}\}$

$$\vdash \{\sigma. List\ 'p\ 'next\ Ps \wedge List\ 'q\ 'next\ Qs \wedge set\ Ps \cap set\ Qs = \{\}\}$$

*IF 'p = Null THEN 'p ::= 'q ELSE 'p → 'next ::= CALL append() FI*

$$\{\{List\ 'p\ 'next\ (Ps @ Qs) \wedge (\forall x. x \notin set\ Ps \longrightarrow x \rightarrow 'next = \sigma_{next}\ x)\}\}$$

Note that *hoare-rule* takes care of multiple auxiliary variables! *HoarePartial.ProcRec1* has only one auxiliary variable, namely  $Z$ . But the type of  $Z$  can be instantiated arbitrarily. So *hoare-rule* instantiates  $Z$  with the tuple  $(\sigma, Ps, Qs)$  and derives a proper variant of the rule. Therefore *hoare-rule* depends on the proper quantification of auxiliary variables!

**apply** *vcg*

1.  $\bigwedge Ps\ Qs\ next\ p\ q.$

$$\llbracket List\ p\ next\ Ps; List\ q\ next\ Qs; set\ Ps \cap set\ Qs = \{\} \rrbracket$$

$$\implies (p = Null \longrightarrow$$

$$\begin{aligned}
& \text{List } q \text{ next } (Ps @ Qs) \wedge (\forall x. x \notin \text{set } Ps \longrightarrow \text{next } x = \text{next } x)) \wedge \\
& (p \neq \text{Null} \longrightarrow \\
& \quad (\exists Psa. \text{List } (\text{next } p) \text{ next } Psa \wedge \\
& \quad \quad (\exists Qsa. \text{List } q \text{ next } Qsa \wedge \\
& \quad \quad \quad \text{set } Psa \cap \text{set } Qsa = \emptyset \wedge \\
& \quad \quad \quad (\forall \text{nexta } pa. \\
& \quad \quad \quad \quad \text{List } pa \text{ nexta } (Psa @ Qsa) \wedge \\
& \quad \quad \quad \quad (\forall x. x \notin \text{set } Psa \longrightarrow \text{nexta } x = \text{next } x) \longrightarrow \\
& \quad \quad \quad \quad \text{List } p (\text{nexta}(p := pa)) (Ps @ Qs) \wedge \\
& \quad \quad \quad \quad (\forall x. x \notin \text{set } Ps \longrightarrow \\
& \quad \quad \quad \quad \quad (\text{nexta}(p := pa)) x = \text{next } x))))))
\end{aligned}$$

For each branch of the *IF* statement we have one conjunct to prove. The *THEN* branch starts with  $p = \text{Null} \longrightarrow \dots$  and the *ELSE* branch with  $p \neq \text{Null} \longrightarrow \dots$ . Let us focus on the *ELSE* branch, where the recursive call to *append* occurs. First of all we have to prove that the precondition for the recursive call is fulfilled. That means we have to provide some witnesses for the lists *Psa* and *Qsa* which are referenced by  $p \rightarrow \text{next}$  (now written as  $\text{next } p$ ) and *q*. Then we have to show that we can derive the overall postcondition from the postcondition of the recursive call. The state components that have changed by the recursive call are the ones with the suffix *a*, like *nexta* and *pa*.

**apply** *fastforce*  
**done**

If the verification condition generator works on a procedure call it checks whether it can find a *modifies* clause in the context. If one is present the procedure call is simplified before the Hoare rule *HoarePartial.ProcSpec* is applied. Simplification of the procedure call means that the “copy back” of the global components is simplified. Only those components that occur in the *modifies* clause are actually copied back. This simplification is justified by the rule *HoarePartial.ProcModifyReturn*. So after this simplification all global components that do not appear in the *modifies* clause are treated as local variables.

We study the effect of the *modifies* clause on the following examples, where we want to prove that (@) does not change the *cont* part of the heap.

**lemma** (*in append-impl*)

**shows**  $\Gamma \vdash \{ \text{cont} = c \} \text{ 'p := CALL append(Null, Null) } \{ \text{cont} = c \}$

**proof** –

**note** *append-spec = append-spec1*

**show** *?thesis*

**apply** *vcg*

1.  $\bigwedge \text{cont next.}$

$\exists Ps. \text{List } \text{Null next } Ps \wedge$

$(\exists Qs. \text{List } \text{Null next } Qs \wedge$

$\text{set } Ps \cap \text{set } Qs = \emptyset \wedge$

$$\begin{aligned}
& (\forall \text{ conta } \text{nexta } p. \\
& \quad \text{List } p \text{ nexta } (Ps @ Qs) \wedge \\
& \quad (\forall x. x \notin \text{set } Ps \longrightarrow \text{nexta } x = \text{next } x) \longrightarrow \\
& \quad \text{conta} = \text{cont})
\end{aligned}$$

Only focus on the very last line: *conta* is the heap component after the procedure call, and *cont* the heap component before the procedure call. Since we have not added the modified clause we do not know that they have to be equal.

**oops**

We now add the frame condition. The list in the modifies clause names all global state components that may be changed by the procedure. Note that we know from the modifies clause that the *cont* parts are not changed. Also a small side note on the syntax. We use ordinary brackets in the postcondition of the modifies clause, and also the state components do not carry the acute, because we explicitly note the state *t* here.

**lemma** (in *append-impl*) *append-modifies*:  
**shows**  $\forall \sigma. \Gamma \vdash /UNIV \{ \sigma \} 'p ::= PROC \text{append}('p, 'q)$   
 $\{ t. t \text{ may-only-modify-globals } \sigma \text{ in } [next] \}$   
**apply** (*hoare-rule HoarePartial.ProcRec1*)  
**apply** (*vcg spec=modifies*)  
**done**

We tell the verification condition generator to use only the modifies clauses and not to search for functional specifications by the parameter *spec=modifies*. It also tries to solve the verification conditions automatically. Again it is crucial to name the lemma with this naming scheme, since the verification condition generator searches for these names.

The modifies clause is equal to a state update specification of the following form.

**lemma** (in *append-impl*) **shows**  $\{ t. t \text{ may-only-modify-globals } Z \text{ in } [next] \}$   
 $=$   
 $\{ t. \exists \text{next}. \text{globals } t = \text{update } id \ id \ \text{next}' \ (K\text{-statefun } \text{next}) \ (\text{globals } Z) \}$   
**apply** (*unfold mex-def meq-def*)  
**apply** *simp*  
**done**

Now that we have proven the frame-condition, it is available within the locale *append-impl* and the *vcg* exploits it.

**lemma** (in *append-impl*)  
**shows**  $\Gamma \vdash \{ 'cont=c \} 'p ::= CALL \text{append}(Null, Null) \{ 'cont=c \}$   
**proof** –  
**note** *append-spec = append-spec1*  
**show** *?thesis*  
**apply** *vcg*

1.  $\bigwedge cont\ next.$ 

$$\begin{aligned} & \exists Ps. List\ Null\ next\ Ps \wedge \\ & (\exists Qs. List\ Null\ next\ Qs \wedge \\ & \quad set\ Ps \cap set\ Qs = \emptyset \wedge \\ & \quad (\forall nexta\ p. \\ & \quad \quad List\ p\ nexta\ (Ps\ @\ Qs) \wedge \\ & \quad \quad (\forall x. x \notin set\ Ps \longrightarrow nexta\ x = next\ x) \longrightarrow \\ & \quad \quad cont = cont)) \end{aligned}$$

With a modifies clause present we know that no change to *cont* has occurred.

by *simp*  
qed

Of course we could add the modifies clause to the functional specification as well. But separating both has the advantage that we split up the verification work. We can make use of the modifies clause before we apply the functional specification in a fully automatic fashion.

To prove that a procedure respects the modifies clause, we only need the modifies clauses of the procedures called in the body. We do not need the functional specifications. So we can always prove the modifies clause without functional specifications, but we may need the modifies clause to prove the functional specifications. So usually the modifies clause is proved before the proof of the functional specification, so that it can already be used by the verification condition generator.

## 26.4 Total Correctness

When proving total correctness the additional proof burden to the user is to come up with a well-founded relation and to prove that certain states get smaller according to this relation. Proving that a relation is well-founded can be quite hard. But fortunately there are ways to construct and stick together relations so that they are well-founded by construction. This infrastructure is already present in Isabelle/HOL. For example, *measure f* is always well-founded; the lexicographic product of two well-founded relations is again well-founded and the inverse image construction *inv-image* of a well-founded relation is again well-founded. The constructions are best explained by some equations:

$$\begin{aligned} ((x, y) \in measure\ f) &= (f\ x < f\ y) \\ (((a, b), x, y) \in r\ <*\!lex*\!>\ s) &= ((a, x) \in r \vee a = x \wedge (b, y) \in s) \\ ((x, y) \in inv\ image\ r\ f) &= ((f\ x, f\ y) \in r) \end{aligned}$$

Another useful construction is *<\*\!mlex\*\!>* which is a combination of a measure and a lexicographic product:

$$((x, y) \in f\ <*\!mlex*\!>\ r) = (f\ x < f\ y \vee f\ x = f\ y \wedge (x, y) \in r)$$

In contrast to the lexicographic product it does not construct a product

type. The state may either decrease according to the measure function  $f$  or the measure stays the same and the state decreases because of the relation  $r$ .

Lets look at a loop:

**lemma** (in *vars*)

$$\begin{array}{l} \Gamma \vdash_t \{ \text{'M} = 0 \wedge \text{'S} = 0 \} \\ \text{WHILE } \text{'M} \neq a \\ \text{INV } \{ \text{'S} = \text{'M} * b \wedge \text{'M} \leq a \} \\ \text{VAR MEASURE } a - \text{'M} \\ \text{DO } \text{'S} ::= \text{'S} + b;; \text{'M} ::= \text{'M} + 1 \text{ OD} \\ \{ \text{'S} = a * b \} \end{array}$$

**apply** *vcg*

1.  $\bigwedge M S. \llbracket M = 0; S = 0 \rrbracket \implies S = M * b \wedge M \leq a$
2.  $\bigwedge M S. \llbracket S = M * b; M \leq a; M \neq a \rrbracket$   
 $\implies a - (M + 1) < a - M \wedge S + b = (M + 1) * b \wedge M + 1 \leq a$
3.  $\bigwedge M S. \llbracket S = M * b; M \leq a; \neg M \neq a \rrbracket \implies S = a * b$

The first conjunct of the second subgoal is the proof obligation that the variant decreases in the loop body.

**by** *auto*

The variant annotation is preceded by *VAR*. The capital *MEASURE* is a shorthand for *measure* ( $\lambda s. a - {}^sM$ ). Analogous there is a capital  $\langle *MLEX* \rangle$ .

**lemma** (in *Fac-impl*) *Fac-spec'*:

**shows**  $\forall \sigma. \Gamma \vdash_t \{ \sigma \} \text{'R} ::= \text{PROC Fac}(\text{'N}) \{ \text{'R} = \text{fac } \sigma \text{'N} \}$   
**apply** (*hoare-rule HoareTotal.ProcRec1* [**where**  $r = \text{measure } (\lambda(s,p). {}^sN)$ ])

In case of the factorial the parameter  $N$  decreases in every call. This is easily expressed by the measure function. Note that the well-founded relation for recursive procedures is formally defined on tuples containing the state space and the procedure name.

1.  $\forall \sigma \sigma'.$   
 $\Gamma, (\bigcup_{\sigma'} \{ (\{ \sigma' \} \cap \{ \text{'N} < \sigma \text{'N} \}, \text{Fac-}'proc, \{ \text{'R} = \text{fac } \sigma' \text{'N} \}, \emptyset) \})$   
 $\vdash_t (\{ \sigma \} \cap \{ \sigma' \})$   
 $\text{IF } \text{'N} = 0 \text{ THEN } \text{'R} ::= 1 \text{ ELSE } \text{'R} ::= \text{CALL Fac}();; \text{'R} ::= \text{'N}$   
 $* \text{'R}$   
 $\text{FI}$   
 $\{ \text{'R} = \text{fac } \sigma' \text{'N} \}$

The initial call to the factorial is in state  $\sigma$ . Note that in the precondition  $\{ \sigma \} \cap \{ \sigma' \}$ ,  $\sigma'$  stems from the lemma we want to prove and  $\sigma$  stems from the recursion rule for total correctness. Both are synonym for the initial state. To use the assumption in the Hoare context we have to show that the call to the factorial is invoked on a smaller  $N$  compared to the initial  $\sigma N$ .

**apply** *vcg*



1.  $\bigwedge N. (N = 0 \longrightarrow 1 = \text{fac } N) \wedge$   
 $(N \neq 0 \longrightarrow$   
 $N - 1 < N \wedge (\forall R. R = \text{fac } (N - 1) \longrightarrow N * \text{fac } (N - 1) = \text{fac } N))$

The tribute to termination is that we have to show  $N - 1 < N$  in case of the recursive call.

by *simp*

**lemma** (in *append-impl*) *append-spec2*:

**shows**  $\forall \sigma \ Ps \ Qs. \Gamma \vdash_t$

$\{\{\sigma. \text{List } 'p \ 'next \ Ps \wedge \text{List } 'q \ 'next \ Qs \wedge \text{set } Ps \cap \text{set } Qs = \{\}\}$   
 $'p := \text{PROC } \text{append}('p, 'q)$

$\{\{\text{List } 'p \ 'next \ (Ps @ Qs) \wedge (\forall x. x \notin \text{set } Ps \longrightarrow 'next \ x = \sigma \text{next } x)\}\}$

**apply** (*hoare-rule HoareTotal.ProcRec1*

**[where**  $r = \text{measure } (\lambda(s, p). \text{length } (\text{list } ^s p \ ^s \text{next}))$ **])**

In case of the *append* function the length of the list referenced by *p* decreases in every recursive call.

1.  $\forall \sigma \ \sigma' \ Ps \ Qs.$

$\Gamma, (\bigcup_{\sigma'} \sigma' \ Ps \ Qs$

$\{\{\{\sigma'. \text{List } 'p \ 'next \ Ps \wedge \text{List } 'q \ 'next \ Qs \wedge \text{set } Ps \cap \text{set } Qs = \emptyset\} \cap$

$\{\{| \text{list } 'p \ 'next | < | \text{list } \sigma p \ \sigma \text{next} |\},$

*append-'proc*,

$\{\{\text{List } 'p \ 'next \ (Ps @ Qs) \wedge$

$(\forall x. x \notin \text{set } Ps \longrightarrow x \rightarrow 'next = \sigma' \text{next } x)\},$

$\emptyset\}\}$

$\vdash_t (\{\sigma\} \cap$

$\{\{\sigma'. \text{List } 'p \ 'next \ Ps \wedge \text{List } 'q \ 'next \ Qs \wedge \text{set } Ps \cap \text{set } Qs = \emptyset\})$

*IF*  $'p = \text{Null}$  *THEN*  $'p := 'q$  *ELSE*  $'p \rightarrow 'next := \text{CALL } \text{append}()$

*FI*

$\{\{\text{List } 'p \ 'next \ (Ps @ Qs) \wedge (\forall x. x \notin \text{set } Ps \longrightarrow x \rightarrow 'next = \sigma' \text{next } x)\}\}$

**apply** *vcg*

**apply** (*fastforce simp add: List-list*)

**done**

In case of the lists above, we have used a relational list abstraction *List* to construct the HOL lists *Ps* and *Qs* for the pre- and postcondition. To supply a proper measure function we use a functional abstraction *list*. The functional abstraction can be defined by means of the relational list abstraction, since the lists are already uniquely determined by the relational abstraction:

$\text{islist } p \ h = (\exists ps. \text{List } p \ h \ ps)$

$\text{list } p \ h = (\text{THE } ps. \text{List } p \ h \ ps)$

**lemma**  $\text{List } p \ h \ ps = (\text{islist } p \ h \wedge ps = \text{list } p \ h)$

The next contrived example is taken from [3], to illustrate a more complex termination criterion for mutually recursive procedures. The procedures do not calculate anything useful.

**procedures**

```

pedal(N::nat,M::nat)
  IF 0 < 'N THEN
    IF 0 < 'M THEN
      CALL coast('N - 1, 'M - 1) FI;;
      CALL pedal('N - 1, 'M)
    FI
  and

```

```

coast(N::nat,M::nat)
  CALL pedal('N, 'M);;
  IF 0 < 'M THEN CALL coast('N, 'M - 1) FI

```

In the recursive calls in procedure *pedal* the first argument always decreases. In the body of *coast* in the recursive call of *coast* the second argument decreases, but in the call to *pedal* no argument decreases. Therefore an relation only on the state space is insufficient. We have to take the procedure names into account, too. We consider the procedure *coast* to be “bigger” than *pedal* when we construct a well-founded relation on the product of state space and procedure names.

**ML**  $\langle ML\text{-Thms.bind-thm } (HoareTotal\text{-ProcRec2}, Hoare.gen\text{-proc-rec } @\{context\} Hoare.Total\ 2) \rangle$

We provide the ML function `gen_proc_rec` to automatically derive a convenient rule for recursion for a given number of mutually recursive procedures.

**lemma** (in *pedal-coast-clique*)

**shows**  $(\forall \sigma. \Gamma \vdash_t \{\sigma\} PROC\ pedal('N, 'M) UNIV) \wedge$   
 $(\forall \sigma. \Gamma \vdash_t \{\sigma\} PROC\ coast('N, 'M) UNIV)$

**apply** (*hoare-rule HoareTotal-ProcRec2*

[**where**  $r = ((\lambda(s,p). {}^sN) < *mlex* >$   
 $(\lambda(s,p). {}^sM) < *mlex* >$   
 $measure (\lambda(s,p). \text{if } p = \text{coast-}'proc \text{ then } 1 \text{ else } 0))$ ])

We can directly express the termination condition described above with the  $< *mlex* >$  construction. Either state component *N* decreases, or it stays the same and *M* decreases or this also stays the same, but then the procedure name has to decrease.

1.  $\forall \sigma \sigma'. \Gamma, (\bigcup_{\sigma'} \{ \{\sigma'\} \cap$   
 $\{ N < \sigma N \vee$   
 $N = \sigma N \wedge$   
 $(M < \sigma M \vee$   
 $M = \sigma M \wedge$   
 $(\text{if } pedal\text{-}'proc = coast\text{-}'proc \text{ then } 1 \text{ else } 0)$   
 $< (\text{if } pedal\text{-}'proc = coast\text{-}'proc \text{ then } 1 \text{ else } 0) \} \},$

$$\begin{aligned}
& \text{pedal-'proc, UNIV, } \emptyset)) \cup \\
& (\bigcup_{\sigma'} \{ \{ \sigma' \} \cap \\
& \quad \{ \{ N < \sigma N \vee \\
& \quad \quad N = \sigma N \wedge \\
& \quad \quad (M < \sigma M \vee \\
& \quad \quad \quad M = \sigma M \wedge \\
& \quad \quad \quad (\text{if coast-'proc} = \text{coast-'proc then } 1 \text{ else } 0) \\
& \quad \quad \quad < (\text{if pedal-'proc} = \text{coast-'proc then } 1 \text{ else } 0) \} \}, \\
& \quad \text{coast-'proc, UNIV, } \emptyset) \} \\
& \vdash_t (\{ \sigma \} \cap \{ \sigma' \}) \\
& \quad \text{IF } 0 < 'N \text{ THEN IF } 0 < 'M \text{ THEN CALL coast() FI;; CALL} \\
\text{pedal() FI} \\
& \quad \text{UNIV} \\
& 2. \forall \sigma \sigma'. \Gamma, (\bigcup_{\sigma'} \{ \{ \sigma' \} \cap \\
& \quad \{ \{ N < \sigma N \vee \\
& \quad \quad N = \sigma N \wedge \\
& \quad \quad (M < \sigma M \vee \\
& \quad \quad \quad M = \sigma M \wedge \\
& \quad \quad \quad (\text{if pedal-'proc} = \text{coast-'proc then } 1 \text{ else } 0) \\
& \quad \quad \quad < (\text{if coast-'proc} = \text{coast-'proc then } 1 \text{ else } 0) \} \}, \\
& \quad \text{pedal-'proc, UNIV, } \emptyset) \} \cup \\
& (\bigcup_{\sigma'} \{ \{ \sigma' \} \cap \\
& \quad \{ \{ N < \sigma N \vee \\
& \quad \quad N = \sigma N \wedge \\
& \quad \quad (M < \sigma M \vee \\
& \quad \quad \quad M = \sigma M \wedge \\
& \quad \quad \quad (\text{if coast-'proc} = \text{coast-'proc then } 1 \text{ else } 0) \\
& \quad \quad \quad < (\text{if coast-'proc} = \text{coast-'proc then } 1 \text{ else } 0) \} \}, \\
& \quad \text{coast-'proc, UNIV, } \emptyset) \} \\
& \vdash_t (\{ \sigma \} \cap \{ \sigma' \}) \text{ CALL pedal();; IF } 0 < 'M \text{ THEN CALL coast() FI} \\
& \text{UNIV}
\end{aligned}$$

**apply simp-all**

$$\begin{aligned}
& 1. \forall \sigma \sigma'. \\
& \quad \Gamma, (\bigcup_x \{ \{ \{ x \} \cap \{ \{ N < \sigma N \vee N = \sigma N \wedge M < \sigma M \} \}, \text{pedal-'proc, UNIV,} \\
& \quad \quad \emptyset) \} \cup \\
& \quad (\bigcup_x \{ \{ \{ x \} \cap \{ \{ N < \sigma N \vee N = \sigma N \wedge M < \sigma M \} \}, \text{coast-'proc, UNIV, } \emptyset) \} \\
& \quad \vdash_t (\{ \sigma \} \cap \{ \sigma' \}) \\
& \quad \quad \text{IF } 0 < 'N \text{ THEN IF } 0 < 'M \text{ THEN CALL coast() FI;; CALL pedal()} \\
& \quad \quad \text{FI} \\
& \quad \quad \text{UNIV} \\
& 2. \forall \sigma \sigma'. \\
& \quad \Gamma, (\bigcup_x \{ \{ \{ x \} \cap \{ \{ N < \sigma N \vee N = \sigma N \wedge (M < \sigma M \vee M = \sigma M) \} \}, \\
& \quad \quad \text{pedal-'proc, UNIV, } \emptyset) \} \cup \\
& \quad (\bigcup_x \{ \{ \{ x \} \cap \{ \{ N < \sigma N \vee N = \sigma N \wedge M < \sigma M \} \}, \text{coast-'proc, UNIV, } \emptyset) \} \\
& \quad \vdash_t (\{ \sigma \} \cap \{ \sigma' \}) \text{ CALL pedal();; IF } 0 < 'M \text{ THEN CALL coast() FI} \\
& \quad \quad \text{UNIV}
\end{aligned}$$

by  $(vcg, simp)+$

We can achieve the same effect without  $\langle *mlex* \rangle$  by using the ordinary lexicographic product  $\langle *lex* \rangle$ ,  $inv\text{-}image$  and  $measure$

**lemma** (in  $pedal\text{-}coast\text{-}clique$ )

**shows**  $(\forall \sigma. \Gamma \vdash_t \{\sigma\} PROC\ pedal('N, 'M) UNIV) \wedge$   
 $(\forall \sigma. \Gamma \vdash_t \{\sigma\} PROC\ coast('N, 'M) UNIV)$

**apply**  $(hoare\text{-}rule\ HoareTotal\text{-}ProcRec2$

**[where**  $r = inv\text{-}image\ (measure\ (\lambda m. m) \langle *lex* \rangle$   
 $measure\ (\lambda m. m) \langle *lex* \rangle$   
 $measure\ (\lambda p. if\ p = coast\text{-}'proc\ then\ 1\ else\ 0))$   
 $(\lambda (s, p). (^sN, ^sM, p)))]$

With the lexicographic product we construct a well-founded relation on triples of type  $nat \times nat \times string$ . With  $inv\text{-}image$  we project the components out of the state-space and the procedure names to this triple.

1.  $\forall \sigma \sigma'$ .

$\Gamma, (\bigcup_{\sigma'} \{(\{\sigma'\} \cap$   
 $\{\} 'N < \sigma N \vee$   
 $'N = \sigma N \wedge$   
 $('M < \sigma M \vee$   
 $'M = \sigma M \wedge$   
 $(if\ pedal\text{-}'proc = coast\text{-}'proc\ then\ 1\ else\ 0)$   
 $< (if\ pedal\text{-}'proc = coast\text{-}'proc\ then\ 1\ else\ 0))\}\},$   
 $pedal\text{-}'proc, UNIV, \emptyset)\} \cup$   
 $(\bigcup_{\sigma'} \{(\{\sigma'\} \cap$   
 $\{\} 'N < \sigma N \vee$   
 $'N = \sigma N \wedge$   
 $('M < \sigma M \vee$   
 $'M = \sigma M \wedge$   
 $(if\ coast\text{-}'proc = coast\text{-}'proc\ then\ 1\ else\ 0)$   
 $< (if\ pedal\text{-}'proc = coast\text{-}'proc\ then\ 1\ else\ 0))\}\},$   
 $coast\text{-}'proc, UNIV, \emptyset)\}$   
 $\vdash_t (\{\sigma\} \cap \{\sigma'\})$   
 $IF\ 0 < 'N\ THEN\ IF\ 0 < 'M\ THEN\ CALL\ coast()\ FI;;\ CALL\ pedal()$   
 $FI$   
 $UNIV$

2.  $\forall \sigma \sigma'$ .

$\Gamma, (\bigcup_{\sigma'} \{(\{\sigma'\} \cap$   
 $\{\} 'N < \sigma N \vee$   
 $'N = \sigma N \wedge$   
 $('M < \sigma M \vee$   
 $'M = \sigma M \wedge$   
 $(if\ pedal\text{-}'proc = coast\text{-}'proc\ then\ 1\ else\ 0)$   
 $< (if\ coast\text{-}'proc = coast\text{-}'proc\ then\ 1\ else\ 0))\}\},$   
 $pedal\text{-}'proc, UNIV, \emptyset)\} \cup$   
 $(\bigcup_{\sigma'} \{(\{\sigma'\} \cap$   
 $\{\} 'N < \sigma N \vee$

$$\begin{aligned}
& 'N = \sigma N \wedge \\
& ('M < \sigma M \vee \\
& \quad 'M = \sigma M \wedge \\
& \quad \text{(if } \textit{coast-'proc} = \textit{coast-'proc} \text{ then } 1 \text{ else } 0) \\
& \quad < \text{(if } \textit{coast-'proc} = \textit{coast-'proc} \text{ then } 1 \text{ else } 0)\text{)}, \\
& \textit{coast-'proc}, \textit{UNIV}, \emptyset\}) \\
\vdash_t (\{\sigma\} \cap \{\sigma'\}) \textit{CALL } \textit{pedal}();; \textit{IF } 0 < 'M \textit{ THEN CALL } \textit{coast}() \textit{ FI} \\
& \textit{UNIV}
\end{aligned}$$

**apply** *simp-all*  
**by** (*vcg, simp*)<sup>+</sup>

By doing some arithmetic we can express the termination condition with a single measure function.

**lemma** (*in pedal-coast-clique*)

**shows**  $(\forall \sigma. \Gamma \vdash_t \{\sigma\} \textit{PROC } \textit{pedal}('N, 'M) \textit{UNIV}) \wedge$   
 $(\forall \sigma. \Gamma \vdash_t \{\sigma\} \textit{PROC } \textit{coast}('N, 'M) \textit{UNIV})$

**apply** (*hoare-rule HoareTotal-ProcRec2*)

[**where**  $r = \textit{measure } (\lambda(s,p). {}^sN + {}^sM + (\textit{if } p = \textit{coast-'proc} \textit{ then } 1 \textit{ else } 0))$ )]

**apply** *simp-all*

1.  $\forall \sigma \sigma'$ .

$$\begin{aligned}
& \Gamma, (\bigcup_x \{(\{x\} \cap \{\{N + 'M < \sigma N + \sigma M\}, \textit{pedal-'proc}, \textit{UNIV}, \emptyset)\}) \cup \\
& \quad (\bigcup_x \{(\{x\} \cap \{\textit{Suc}('N + 'M) < \sigma N + \sigma M\}, \textit{coast-'proc}, \textit{UNIV}, \emptyset)\}) \\
& \quad \vdash_t (\{\sigma\} \cap \{\sigma'\}) \\
& \quad \textit{IF } 0 < 'N \textit{ THEN IF } 0 < 'M \textit{ THEN CALL } \textit{coast}() \textit{ FI};; \textit{CALL } \textit{pedal}() \\
& \quad \textit{FI} \\
& \quad \textit{UNIV}
\end{aligned}$$

2.  $\forall \sigma \sigma'$ .

$$\begin{aligned}
& \Gamma, (\bigcup_x \{(\{x\} \cap \{\textit{Suc}(\sigma N + \sigma M)\}, \textit{pedal-'proc}, \textit{UNIV}, \emptyset)\}) \cup \\
& \quad (\bigcup_x \{(\{x\} \cap \{\textit{Suc}('N + 'M) < \sigma N + \sigma M\}, \textit{coast-'proc}, \textit{UNIV}, \emptyset)\}) \\
& \quad \vdash_t (\{\sigma\} \cap \{\sigma'\}) \textit{CALL } \textit{pedal}();; \textit{IF } 0 < 'M \textit{ THEN CALL } \textit{coast}() \textit{ FI} \\
& \quad \textit{UNIV}
\end{aligned}$$

**by** (*vcg, simp, arith?*)<sup>+</sup>

## 26.5 Guards

The purpose of a guard is to guard the **(sub-) expressions** of a statement against runtime faults. Typical runtime faults are array bound violations, dereferencing null pointers or arithmetical overflow. Guards make the potential runtime faults explicit, since the expressions themselves never “fail” because they are ordinary HOL expressions. To relieve the user from typing in lots of standard guards for every subexpression, we supply some input syntax for the common language constructs that automatically generate the guards. For example the guarded assignment  $'M ::=_g ('M + 1) \textit{div} 'N$

gets expanded to guarded command  $(False, \{in-range ('M + 1) \wedge 'N \neq 0 \wedge in-range (('M + 1) div 'N)\}) \mapsto 'M := ('M + 1) div 'N$ . Here *in-range* is uninterpreted by now.

**lemma** (in vars)  $\Gamma \vdash \{True\} 'M :=_g ('M + 1) div 'N \{True\}$   
**apply** *vcg*

1.  $\bigwedge M N. True \implies in-range (M + 1) \wedge N \neq 0 \wedge in-range ((M + 1) div N)$

**oops**

The user can supply on (overloaded) definition of *in-range* to fit to his needs. Currently guards are generated for:

- overflow and underflow of numbers (*in-range*). For subtraction of natural numbers  $a - b$  the guard  $b \leq a$  is generated instead of *in-range* to guard against underflows.
- division by 0
- dereferencing of *Null* pointers
- array bound violations

Following (input) variants of guarded statements are available:

- Assignment:  $\dots :=_g \dots$
- If:  $IF_g \dots$
- While:  $WHILE_g \dots$
- Call:  $CALL_g \dots$  or  $\dots := CALL_g \dots$

## 26.6 Miscellaneous Techniques

### 26.6.1 Modifies Clause

We look at some issues regarding the modifies clause with the example of insertion sort for heap lists.

**primrec** *sorted*::  $('a \Rightarrow 'a \Rightarrow bool) \Rightarrow 'a\ list \Rightarrow bool$

**where**

*sorted le [] = True |*

*sorted le (x#xs) = (( $\forall y \in set\ xs. le\ x\ y$ )  $\wedge$  *sorted le xs*)*

**procedures** (**imports** *globals-heap*)

*insert*( $r::ref, p::ref \mid p::ref$ )

*IF 'r=Null THEN SKIP*

```

ELSE IF 'p=Null THEN 'p ::= 'r;; 'p→'next ::= Null
ELSE IF 'r→'cont ≤ 'p→'cont
  THEN 'r→'next ::= 'p;; 'p::='r
  ELSE 'p→'next ::= CALL insert('r,'p→'next)
FI
FI
FI

```

**lemma** (in *insert-impl*) *insert-modifies*:

```

∀σ. Γ ⊢ /UNIV {σ} 'p ::= PROC insert('r,'p)
  {t. t may-only-modify-globals σ in [next]}
by (hoare-rule HoarePartial.ProcRec1) (vcg spec=modifies)

```

**lemma** (in *insert-impl*) *insert-spec*:

```

∀σ Ps . Γ ⊢
  {σ. List 'p 'next Ps ∧ sorted (≤) (map 'cont Ps) ∧
   'r ≠ Null ∧ 'r ∉ set Ps}
  'p ::= PROC insert('r,'p)
  {∃ Qs. List 'p 'next Qs ∧ sorted (≤) (map σcont Qs) ∧
   set Qs = insert σr (set Ps) ∧
   (∀ x. x ∉ set Qs → 'next x = σnext x)}

```

In the postcondition of the functional specification there is a small but important subtlety. Whenever we talk about the *cont* part we refer to the one of the pre-state. The reason is that we have separated out the information that *cont* is not modified by the procedure, to the *modifies* clause. So whenever we talk about unmodified parts in the postcondition we have to use the pre-state part, or explicitly state an equality in the postcondition. The reason is simple. If the postcondition would talk about *'cont* instead of  $\sigma cont$ , we get a new instance of *cont* during verification and the postcondition would only state something about this new instance. But as the verification condition generator uses the *modifies* clause the caller of *insert* instead still has the old *cont* after the call. That's the sense of the *modifies* clause. So the caller and the specification simply talk about two different things, without being able to relate them (unless an explicit equality is added to the specification).

## 26.6.2 Annotations

Annotations (like loop invariants) are mere syntactic sugar of statements that are used by the *vcg*. Logically a statement with an annotation is equal to the statement without it. Hence annotations can be introduced by the user while building a proof:

$$\text{HoarePartial.annotateI: } \frac{\Gamma, \Theta \vdash /F P \text{ anno } Q, A \quad c = \text{anno}}{\Gamma, \Theta \vdash /F P \ c \ Q, A}$$

When introducing annotations it can easily happen that these mess around with the nesting of sequential composition. Then after stripping the annotations the resulting statement is no longer syntactically identical to original one, only equivalent modulo associativity of sequential composition. The following rule also deals with this case:

$$\text{HoarePartial.annotate-normI: } \frac{\Gamma, \Theta \vdash /_F P \text{ anno } Q, A \quad \text{Language.normalize } c = \text{Language.normalize anno}}{\Gamma, \Theta \vdash /_F P \ c \ Q, A}$$

## Loop Annotations

```

procedures (imports globals-heap)
  insertSort(p::ref | p::ref)
  where r::ref q::ref in
    r ::= Null;;
    WHILE (p ≠ Null) DO
      q ::= p;;
      p ::= p → next;;
      r ::= CALL insert(q, r)
    OD;;
    p ::= r

```

**lemma** (**in** *insertSort-impl*) *insertSort-modifies*:

**shows**

$\forall \sigma. \Gamma \vdash /_{UNIV} \{ \sigma \} \ i \ p \ ::= \ \text{PROC } \textit{insertSort}(i \ p)$   
 $\{ t. \ t \ \text{may-only-modify-globals } \sigma \ \text{in } [\textit{next}] \}$

**apply** (*hoare-rule* *HoarePartial.ProcRec1*)

**apply** (*vcg spec=modifies*)

**done**

Insertion sort is not implemented recursively here, but with a loop. Note that the while loop is not annotated with an invariant in the procedure definition. The invariant only comes into play during verification. Therefore we annotate the loop first, before we run the *vcg*.

**lemma** (**in** *insertSort-impl*) *insertSort-spec*:

**shows**  $\forall \sigma \ Ps.$

$\Gamma \vdash \{ \sigma. \ \text{List } i \ p \ \textit{next} \ Ps \}$   
 $i \ p \ ::= \ \text{PROC } \textit{insertSort}(i \ p)$   
 $\{ \exists Qs. \ \text{List } i \ p \ \textit{next} \ Qs \wedge \text{sorted } (\leq) \ (\text{map } \sigma \text{cont } Qs) \wedge$   
 $\text{set } Qs = \text{set } Ps \}$

**apply** (*hoare-rule* *HoarePartial.ProcRec1*)

**apply** (*hoare-rule anno=*

$i \ r \ ::= \ \text{Null};;$   
 $\text{WHILE } i \ p \ \neq \ \text{Null}$   
 $\text{INV } \{ \exists Qs \ Rs. \ \text{List } i \ p \ \textit{next} \ Qs \wedge \text{List } i \ r \ \textit{next} \ Rs \wedge$   
 $\text{set } Qs \cap \text{set } Rs = \{ \} \wedge$   
 $\text{sorted } (\leq) \ (\text{map } i \ \text{cont } Rs) \wedge \text{set } Qs \cup \text{set } Rs = \text{set } Ps \wedge$



$$\begin{aligned} & \text{'cont} = \sigma_{\text{cont}} \} \\ & \text{DO 'q} ::= \text{'p};; \text{'p} ::= \text{'p} \rightarrow \text{'next};; \text{'r} ::= \text{CALL insert('q, 'r) OD}; \\ & \text{'p} ::= \text{'r in HoarePartial.annotateI)} \\ & \text{apply vcg} \\ & \dots \end{aligned}$$

The method *hoare-rule* automatically solves the side-condition that the annotated program is the same as the original one after stripping the annotations.

## Specification Annotations

When verifying a larger block of program text, it might be useful to split up the block and to prove the parts in isolation. This is especially useful to isolate loops. On the level of the Hoare calculus the parts can then be combined with the consequence rule. To automate this process we introduce the derived command *specAnno*, which allows to introduce a Hoare tuple (inclusive auxiliary variables) in the program text:

$$\text{specAnno } P \ c \ Q \ A = c \ \text{undefined}$$

The whole annotation reduces to the body *c undefined*. The type of the assertions *P*, *Q* and *A* is '*a*  $\Rightarrow$  '*s* set and the type of command *c* is '*a*  $\Rightarrow$  ('*s*, '*p*, '*f*) com. All entities formally depend on an auxiliary (logical) variable of type '*a*. The body *c* formally also depends on this variable, since a nested annotation or loop invariant may also depend on this logical variable. But the raw body without annotations does not depend on the logical variable. The logical variable is only used by the verification condition generator. We express this by defining the whole *specAnno* to be equivalent with the body applied to an arbitrary variable.

The Hoare rule for *specAnno* is mainly an instance of the consequence rule:

$$\llbracket P \subseteq \{s \mid \exists Z. s \in P' Z \wedge Q' Z \subseteq Q \wedge A' Z \subseteq A\}; \forall Z. \Gamma, \Theta \vdash_{/F} (P' Z) \ c \ Z \ (Q' Z), (A' Z); \forall Z. c \ Z = c \ \text{undefined} \rrbracket \Longrightarrow \Gamma, \Theta \vdash_{/F} P \ \text{specAnno } P' \ c \ Q' \ A' \ Q, A$$

The side-condition  $\forall Z. c \ Z = c \ \text{undefined}$  expresses the intention of body *c* explained above: The raw body is independent of the auxiliary variable. This side-condition is solved automatically by the *vcg*. The concrete syntax for this specification annotation is shown in the following example:

**lemma** (in vars)  $\Gamma \vdash \{\sigma\}$

$$\begin{aligned} & \text{'I} ::= \text{'M};; \\ & \text{ANNO } \tau. \{ \tau. \text{'I} = \sigma \text{M} \} \\ & \quad \text{'M} ::= \text{'N};; \text{'N} ::= \text{'I} \\ & \quad \{ \text{'M} = \tau \text{N} \wedge \text{'N} = \tau \text{I} \} \\ & \{ \text{'M} = \sigma \text{N} \wedge \text{'N} = \sigma \text{M} \} \end{aligned}$$

With the annotation we can name an intermediate state  $\tau$ . Since the postcondition refers to  $\sigma$  we have to link the information about the equivalence of  ${}^\tau I$  and  ${}^\sigma M$  in the specification in order to be able to derive the postcondition.

**apply** *vcg-step*  
**apply** *vcg-step*

1.  $\bigwedge \tau. \Gamma \vdash \{\tau. {}^\tau I = {}^\sigma M\} \ 'M ::= \ 'N;; \ 'N ::= \ 'I \ \{\{ \ 'M = {}^\tau N \wedge \ 'N = {}^\tau I \}\}$
2.  $\Gamma \vdash \{\sigma\} \ 'I ::= \ 'M \ \{\{ \ 'I = {}^\sigma M \wedge (\forall t. \ 'tM = \ 'N \wedge \ 'tN = \ 'I \longrightarrow \ 'N = {}^\sigma N \wedge \ 'I = {}^\sigma M) \}\}$

The first subgoal is the isolated Hoare tuple. The second one is the side-condition of the consequence rule that allows us to derive the outermost pre/post condition from our inserted specification.  ${}^\tau I = {}^\sigma M$  is the precondition of the specification, The second conjunct is a simplified version of  $\forall t. \ 'tM = \ 'N \wedge \ 'tN = \ 'I \longrightarrow \ 'tM = {}^\sigma N \wedge \ 'tN = {}^\sigma M$  expressing that the postcondition of the specification implies the outermost postcondition.

**apply** *vcg*

1.  $\bigwedge M \ I \ N. \ I = M \implies N = N \wedge I = I$
2.  $\Gamma \vdash \{\sigma\} \ 'I ::= \ 'M \ \{\{ \ 'I = {}^\sigma M \wedge (\forall t. \ 'tM = \ 'N \wedge \ 'tN = \ 'I \longrightarrow \ 'N = {}^\sigma N \wedge \ 'I = {}^\sigma M) \}\}$

**apply** *simp*  
**apply** *vcg*

1.  $\bigwedge M \ N. \ M = M \wedge (\forall Ma \ Na. \ Ma = N \wedge Na = M \longrightarrow N = N \wedge M = M)$

**by** *simp*

**lemma** (in *vars*)

$$\begin{array}{l} \Gamma \vdash \{\sigma\} \\ \ 'I ::= \ 'M;; \\ ANNO \ \tau. \ \{\tau. \ 'I = {}^\sigma M\} \\ \ 'M ::= \ 'N;; \ 'N ::= \ 'I \\ \ \{\{ \ 'M = {}^\tau N \wedge \ 'N = {}^\tau I \}\} \\ \ \{\{ \ 'M = {}^\sigma N \wedge \ 'N = {}^\sigma M \}\} \end{array}$$

**apply** *vcg*

1.  $\bigwedge M \ N. \ M = M \wedge (\forall Ma \ Na. \ Ma = N \wedge Na = M \longrightarrow N = N \wedge M = M)$
2.  $\bigwedge M \ I \ N. \ I = M \implies N = N \wedge I = I$

**by** *simp-all*

Note that *vcg-step* changes the order of sequential composition, to allow the user to decompose sequences by repeated calls to *vcg-step*, whereas *vcg* preserves the order.

The above example illustrates how we can introduce a new logical state variable  $\tau$ . You can introduce multiple variables by using a tuple:

**lemma (in vars)**

```

 $\Gamma \vdash \{\sigma\}$ 
 $\tau := \tau;$ 
 $ANNO (n,i,m). \{\tau = \sigma M \wedge \tau N = n \wedge \tau i = i \wedge \tau M = m\}$ 
 $\tau M := \tau N;; \tau N := \tau i$ 
 $\{\tau M = n \wedge \tau N = i\}$ 
 $\{\tau M = \sigma N \wedge \tau N = \sigma M\}$ 

```

**apply** *vcg*

1.  $\bigwedge M N. M = M \wedge (\forall Ma Na. Ma = N \wedge Na = M \longrightarrow N = N \wedge M = M)$
2.  $\bigwedge M I N. I = M \implies N = N \wedge I = I$

**by** *simp-all*

## Lemma Annotations

The specification annotations described before split the verification into several Hoare triples which result in several subgoals. If we instead want to proof the Hoare triples independently as separate lemmas we can use the *LEMMA* annotation to plug together the lemmas. It inserts the lemma in the same fashion as the specification annotation.

**lemma (in vars) foo-lemma:**

```

 $\forall n m. \Gamma \vdash \{\tau N = n \wedge \tau M = m\} \tau N := \tau N + 1;; \tau M := \tau M + 1$ 
 $\{\tau N = n + 1 \wedge \tau M = m + 1\}$ 

```

**apply** *vcg*

**apply** *simp*

**done**

**lemma (in vars)**

```

 $\Gamma \vdash \{\tau N = n \wedge \tau M = m\}$ 
 $LEMMA \text{ foo-lemma}$ 
 $\tau N := \tau N + 1;; \tau M := \tau M + 1$ 
 $END;;$ 
 $\tau N := \tau N + 1$ 
 $\{\tau N = n + 2 \wedge \tau M = m + 1\}$ 

```

**apply** *vcg*

**apply** *simp*

**done**

**lemma (in vars)**

```

 $\Gamma \vdash \{ 'N = n \wedge 'M = m \}$ 
  LEMMA foo-lemma
    'N ::= 'N + 1;; 'M ::= 'M + 1
  END;;
  LEMMA foo-lemma
    'N ::= 'N + 1;; 'M ::= 'M + 1
  END
  { 'N = n + 2  $\wedge$  'M = m + 2 }
apply vcg
apply simp
done

```

```

lemma (in vars)
 $\Gamma \vdash \{ 'N = n \wedge 'M = m \}$ 
  'N ::= 'N + 1;; 'M ::= 'M + 1;;
  'N ::= 'N + 1;; 'M ::= 'M + 1
  { 'N = n + 2  $\wedge$  'M = m + 2 }
apply (hoare-rule anno=
  LEMMA foo-lemma
    'N ::= 'N + 1;; 'M ::= 'M + 1
  END;;
  LEMMA foo-lemma
    'N ::= 'N + 1;; 'M ::= 'M + 1
  END
  in HoarePartial.annotate-normI)
apply vcg
apply simp
done

```

### 26.6.3 Total Correctness of Nested Loops

When proving termination of nested loops it is sometimes necessary to express that the loop variable of the outer loop is not modified in the inner loop. To express this one has to fix the value of the outer loop variable before the inner loop and use this value in the invariant of the inner loop. This can be achieved by surrounding the inner while loop with an *ANNO* specification as explained previously. However, this leads to repeating the invariant of the inner loop three times: in the invariant itself and in the the pre- and postcondition of the *ANNO* specification. Moreover one has to deal with the additional subgoal introduced by *ANNO* that expresses how the pre- and postcondition is connected to the invariant. To avoid this extra specification and verification work, we introduce an variant of the annotated while-loop, where one can introduce logical variables by *FIX*. As for the *ANNO* specification multiple logical variables can be introduced via a tuple (*FIX* (*a,b,c*)).

The Hoare logic rule for the augmented while-loop is a mixture of the invariant rule for loops and the consequence rule for *ANNO*:

$$\begin{aligned} & \llbracket P \subseteq \{s \mid \exists Z. s \in IZ \wedge (\forall t. t \in IZ \cap \neg b \longrightarrow t \in Q)\}; \forall Z \sigma. \Gamma, \Theta \vdash_{t/F} \\ & (\{\sigma\} \cap IZ \cap b) \ c \ Z \ (\{t \mid (t, \sigma) \in VZ\} \cap IZ), A; \forall Z. c \ Z = c \ \text{undefined}; \\ & \forall Z. wf \ (VZ) \rrbracket \Longrightarrow \Gamma, \Theta \vdash_{t/F} P \ \text{whileAnnoFix} \ b \ I \ V \ c \ Q, A \end{aligned}$$

The first premise expresses that the precondition implies the invariant and that the invariant together with the negated loop condition implies the postcondition. Since both implications may depend on the choice of the auxiliary variable  $Z$  these two implications are expressed in a single premise and not in two of them as for the usual while rule. The second premise is the preservation of the invariant by the loop body. And the third premise is the side-condition that the computational part of the body does not depend on the auxiliary variable. Finally the last premise is the well-foundedness of the variant. The last two premises are usually discharged automatically by the verification condition generator. Hence usually two subgoals remain for the user, stemming from the first two premises.

The following example illustrates the usage of this rule. The outer loop increments the loop variable  $M$  while the inner loop increments  $N$ . To discharge the proof obligation for the termination of the outer loop, we need to know that the inner loop does not mess around with  $M$ . This is expressed by introducing the logical variable  $m$  and fixing the value of  $M$  to it.

**lemma** (in vars)

$$\begin{aligned} & \Gamma \vdash_t \{ \{ 'M=0 \wedge 'N=0 \} \\ & \quad \text{WHILE } ('M < i) \\ & \quad \text{INV } \{ 'M \leq i \wedge ('M \neq 0 \longrightarrow 'N = j) \wedge 'N \leq j \} \\ & \quad \text{VAR MEASURE } (i - 'M) \\ & \quad \text{DO} \\ & \quad \quad 'N ::= 0;; \\ & \quad \quad \text{WHILE } ('N < j) \\ & \quad \quad \text{FIX } m. \\ & \quad \quad \text{INV } \{ 'M=m \wedge 'N \leq j \} \\ & \quad \quad \text{VAR MEASURE } (j - 'N) \\ & \quad \quad \text{DO} \\ & \quad \quad \quad 'N ::= 'N + 1 \\ & \quad \quad \text{OD};; \\ & \quad \quad 'M ::= 'M + 1 \\ & \quad \quad \text{OD} \\ & \quad \{ 'M=i \wedge ('M \neq 0 \longrightarrow 'N=j) \} \end{aligned}$$

**apply** *vcg*

$$\begin{aligned} & 1. \bigwedge M N. \llbracket M = 0; N = 0 \rrbracket \Longrightarrow M \leq i \wedge (M \neq 0 \longrightarrow N = j) \wedge N \leq j \\ & 2. \bigwedge M N. \llbracket M \leq i; M \neq 0 \longrightarrow N = j; N \leq j; M < i \rrbracket \\ & \quad \Longrightarrow 0 \leq j \wedge \\ & \quad (\forall M a N. \\ & \quad \quad Ma = M \wedge N \leq j \wedge \neg N < j \longrightarrow \\ & \quad \quad i - (M + 1) < i - M \wedge M + 1 \leq i \wedge (M + 1 \neq 0 \longrightarrow N \\ & \quad = j)) \end{aligned}$$

3.  $\bigwedge M N. \llbracket N \leq j; N < j \rrbracket \implies j - (N + 1) < j - N \wedge M = M \wedge N + 1 \leq j$
4.  $\bigwedge M N. \llbracket M \leq i; M \neq 0 \longrightarrow N = j; N \leq j; \neg M < i \rrbracket$   
 $\implies M = i \wedge (M \neq 0 \longrightarrow N = j)$

The first subgoal is from the precondition to the invariant of the outer loop. The fourth subgoal is from the invariant together with the negated loop condition of the outer loop to the postcondition. The subgoals two and three are from the body of the outer while loop which is mainly the inner while loop. Because we introduce the logical variable  $m$  here, the while Rule described above is used instead of the ordinary while Rule. That is why we end up with two subgoals for the inner loop. Subgoal two is from the invariant and the loop condition of the outer loop to the invariant of the inner loop. And at the same time from the invariant of the inner loop to the invariant of the outer loop (together with the proof obligation that the measure of the outer loop decreases). The universal quantified variables  $M$  and  $N$  are the “fresh” state variables introduced for the final state of the inner loop. The equality  $M = M$  is the result of the equality  $M = m$  in the inner invariant. Subgoal three is the preservation of the invariant by the inner loop body (together with the proof obligation that the measure of the inner loop decreases).

## 26.7 Functional Correctness, Termination and Runtime Faults

Total correctness of a program with guards conceptually leads to three verification tasks.

- functional (partial) correctness
- absence of runtime faults
- termination

In case of a modifies specification the functional correctness part can be solved automatically. But the absence of runtime faults and termination may be non trivial. Fortunately the modifies clause is usually just a helpful companion of another specification that expresses the “real” functional behaviour. Therefore the task to prove the absence of runtime faults and termination can be dealt with during the proof of this functional specification. In most cases the absence of runtime faults and termination heavily build on the functional specification parts. So after all there is no reason why we should again prove the absence of runtime faults and termination for the modifies clause. Therefore it suffices to have partial correctness of the modifies clause for a program were all guards are ignored. This leads to the following pattern:

**procedures**  $foo(N::nat|M::nat)$

$\text{'}M ::= \text{'}$   $M$

— think of body with guards instead

$foo-spec: \forall \sigma. \Gamma \vdash_t (P \sigma) \text{'}M ::= PROC\ foo(\text{'}N) (Q \sigma)$

*foo-modifies*:  $\forall \sigma. \Gamma \vdash_{UNIV} \{ \sigma \} 'M ::= PROC\ foo('N)$   
 $\{ t. t\ may\ only\ modify\ globals\ \sigma\ in\ [] \}$

The verification condition generator can solve those modifies clauses automatically and can use them to simplify calls to *foo* even in the context of total correctness.

## 26.8 Procedures and Locales

Verification of a larger program is organised on the granularity of procedures. We proof the procedures in a bottom up fashion. Of course you can also always use Isabelle’s dummy proof *sorry* to prototype your formalisation. So you can write the theory in a bottom up fashion but actually prove the lemmas in any other order.

Here are some explanations of handling of locales. In the examples below, consider *proc<sub>1</sub>* and *proc<sub>2</sub>* to be “leaf” procedures, which do not call any other procedure. Procedure *proc* directly calls *proc<sub>1</sub>* and *proc<sub>2</sub>*.

**lemma** (**in** *proc<sub>1</sub>-impl*) *proc<sub>1</sub>-modifies*:  
**shows** ...

After the proof of *proc<sub>1</sub>-modifies*, the **in** directive stores the lemma in the locale *proc<sub>1</sub>-impl*. When we later on include *proc<sub>1</sub>-impl* or prove another theorem in locale *proc<sub>1</sub>-impl* the lemma *proc<sub>1</sub>-modifies* will already be available as fact.

**lemma** (**in** *proc<sub>1</sub>-impl*) *proc<sub>1</sub>-spec*:  
**shows** ...

**lemma** (**in** *proc<sub>2</sub>-impl*) *proc<sub>2</sub>-modifies*:  
**shows** ...

**lemma** (**in** *proc<sub>2</sub>-impl*) *proc<sub>2</sub>-spec*:  
**shows** ...

**lemma** (**in** *proc-impl*) *proc-modifies*:  
**shows** ...

Note that we do not explicitly include anything about *proc<sub>1</sub>* or *proc<sub>2</sub>* here. This is handled automatically. When defining an *impl*-locale it imports all *impl*-locales of procedures that are called in the body. In case of *proc-impl* this means, that *proc<sub>1</sub>-impl* and *proc<sub>2</sub>-impl* are imported. This has the neat effect that all theorems that are proven in *proc<sub>1</sub>-impl* and *proc<sub>2</sub>-impl* are also present in *proc-impl*.

**lemma** (**in** *proc-impl*) *proc-spec*:  
**shows** ...

As we have seen in this example you only have to prove a procedure in its own *impl* locale. You do not have to include any other locale.

## 26.9 Records

Before *statespaces* were introduced the state was represented as a *record*. This is still supported. Compared to the flexibility of *statespaces* there are some drawbacks in particular with respect to modularity. Even names of local variables and parameters are globally visible and records can only be extended in a linear fashion, whereas *statespaces* also allow multiple inheritance. The usage of records is quite similar to the usage of *statespaces*. We repeat the example of an append function for heap lists. First we define the global components. Again the appearance of the prefix ‘globals’ is mandatory. This is the way the syntax layer distinguishes local and global variables.

```
record globals-list =
  next-' :: ref ⇒ ref
  cont-' :: ref ⇒ nat
```

The local variables also have to be defined as a record before the actual definition of the procedure. The parent record *state* defines a generic *globals* field as a place-holder for the record of global components. In contrast to the *statespace* approach there is no single *locals* slot. The local components are just added to the record.

```
record 'g list-vars = 'g state +
  p-'   :: ref
  q-'   :: ref
  r-'   :: ref
  root-' :: ref
  tmp-' :: ref
```

Since the parameters and local variables are determined by the record, there are no type annotations or definitions of local variables while defining a procedure.

### procedures

```
append'(p,q|p) =
  IF 'p=Null THEN 'p ::= 'q
  ELSE 'p → 'next ::= CALL append'('p→'next, 'q) FI
```

As in the *statespace* approach, a locale called *append'-impl* is created. Note that we do not give any explicit information which global or local state-record to use. Since the records are already defined we rely on Isabelle’s type inference. Dealing with the locale is analogous to the case with *statespaces*.

**lemma** (in *append'-impl*) *append'-modifies*:

**shows**

```
∀ σ. Γ ⊢ {σ} 'p ::= PROC append'('p, 'q)
  {t. t may-only-modify-globals σ in [next]}
```

**apply** (*hoare-rule HoarePartial.ProcRec1*)

**apply** (*vcg spec=modifies*)



**done**

**lemma** (in *append'-impl*) *append'-spec*:

**shows**  $\forall \sigma \ Ps \ Qs. \Gamma \vdash$

$\{\sigma. \text{List } 'p \ 'next \ Ps \wedge \text{List } 'q \ 'next \ Qs \wedge \text{set } Ps \cap \text{set } Qs = \{\}\}$

$'p ::= \text{PROC } \text{append}'('p, 'q)$

$\{\text{List } 'p \ 'next \ (Ps @ Qs) \wedge (\forall x. x \notin \text{set } Ps \longrightarrow 'next \ x = \sigma_{next} \ x)\}$

**apply** (*hoare-rule HoarePartial.ProcRec1*)

**apply** *vcg*

**apply** *fastforce*

**done**

However, in some corner cases the inferred state type in a procedure definition can be too general which raises problems when attempting to proof a suitable specifications in the locale. Consider for example the simple procedure body  $'p ::= \text{NULL}$  for a procedure *init*.

**procedures** *init* ( $|p$ ) =

$'p ::= \text{Null}$

Here Isabelle can only infer the local variable record. Since no reference to any global variable is made the type fixed for the global variables (in the locale *init'-impl*) is a type variable say *'g* and not a *globals-list* record. Any specification mentioning *next* or *cont* restricts the state type and cannot be added to the locale *init'-impl*. Hence we have to restrict the body  $'p ::= \text{NULL}$  in the first place by adding a typing annotation:

**procedures** *init'* ( $|p$ ) =

$'p ::= \text{Null} :: (('a \ \text{globals-list-scheme}, 'b) \ \text{list-vars-scheme}, \text{char } \text{list}, 'c) \ \text{com}$

### 26.9.1 Extending State Spaces

The records in Isabelle are extensible [7, 6]. In principle this can be exploited during verification. The state space can be extended while we add procedures. But there is one major drawback:

- records can only be extended in a linear fashion (there is no multiple inheritance)

You can extend both the main state record as well as the record for the global variables.

### 26.9.2 Mapping Variables to Record Fields

Generally the state space (global and local variables) is flat and all components are accessible from everywhere. Locality or globality of variables is achieved by the proper *init* and *return/result* functions in procedure calls. What is the best way to map programming language variables to the state

records? One way is to disambiguate all names, by using the procedure names as prefix or the structure names for heap components. This leads to long names and lots of record components. But for local variables this is not necessary, since variable  $i$  of procedure  $A$  and variable  $i$  of procedure  $B$  can be mapped to the same record component, without any harm, provided they have the same logical type. Therefore for local variables it is preferable to map them per type. You only have to distinguish a variable with the same name if they have a different type. Note that all pointers just have logical type  $ref$ . So you even do not have to distinguish between a pointer  $p$  to an integer and a pointer  $p$  to a list. For global components (global variables and heap structures) you have to disambiguate the name. But hopefully the field names of structures have different names anyway. Also note that there is no notion of hiding of a global component by a local one in the logic. You have to disambiguate global and local names! As the names of the components show up in the specifications and the proof obligations, names are even more important as for programming. Try to find meaningful and short names, to avoid cluttering up your reasoning.

## References

- [1] E. Alkassar, N. Schirmer, and A. Starostin. Formal pervasive verification of a paging mechanism. In *14th intl Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS08)*, to appear, LNCS. Springer, 2008.
- [2] C. Ballarin. Locales and locale expressions in Isabelle/Isar. In S. Berardi, M. Coppo, and F. Damiani, editors, *Types for Proofs and Programs: International Workshop, TYPES 2003, Torino, Italy, April 30–May 4, 2003, Selected Papers*, number 3085 in LNCS, pages 34–50. Springer, 2004.
- [3] P. V. Homeier. *Trustworthy Tools for Trustworthy Programs: A Mechanically Verified Verification Condition Generator for the Total Correctness of Procedures*. PhD thesis, Department of Computer Science, University of California, Los Angeles, 1995.
- [4] D. Leinenbach and E. Petrova. Pervasive compiler verification – from verified programs to verified systems. In *3rd intl Workshop on Systems Software Verification (SSV08)*, to appear. Elsevier Science B. V., 2008.
- [5] F. Mehta and T. Nipkow. Proving pointer programs in higher-order logic. In F. Baader, editor, *Automated Deduction — CADE-19*, volume 2741 of LNCS, pages 121–135. Springer, 2003.

- [6] W. Naraschewski and M. Wenzel. Object-oriented verification based on record subtyping in higher-order logic. In *Theorem Proving in Higher Order Logics: 11th International Conference, TPHOLs'98*, volume 1479 of *LNCS*. Springer, 1998.
- [7] T. Nipkow, L. Paulson, and M. Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002. <http://www.in.tum.de/~nipkow/LNCS2283/>.
- [8] V. Ortner and N. Schirmer. Verification of BDD normalization. In J. Hurd and T. Melham, editors, *Theorem Proving in Higher Order Logics, 18th International Conference, TPHOLs 2005, Oxford, UK, August 2005*, volume 3603 of *LNCS*, pages 261–277. Springer, 2005.
- [9] N. Schirmer. *Verification of Sequential Imperative Programs in Isabelle/HOL*. PhD thesis, Technische Universität München, 2006. Available from <http://mediatum2.ub.tum.de/doc/601799/601799.pdf>.
- [10] H. Tuch, G. Klein, and M. Norrish. Types, bytes, and separation logic. In *POPL '07: Proceedings of the 34th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 97–108, New York, NY, USA, 2007. ACM Press.