Abstract Hoare Logics

Tobias Nipkow

February 23, 2021

Abstract

These therories describe Hoare logics for a number of imperative language constructs, from while-loops to mutually recursive procedures. Both partial and total correctness are treated. In particular a proof system for total correctness of recursive procedures in the presence of unbounded nondeterminism is presented.

Contents

1 Introduction **1**

2 Hoare Logics for While **2**
2.1 The language 2
2.2 Hoare logic for partial correctness 3
2.3 Termination 5
2.4 Hoare logic for total correctness 6

3 Hoare Logics for 1 Procedure **8**
3.1 The language 8
3.2 Hoare logic for partial correctness 11
3.3 Termination 14
3.4 Hoare logic for total correctness 15

4 Hoare Logics for Mutually Recursive Procedure **21**
4.1 The language 21
4.2 Hoare logic for partial correctness 24
4.3 Termination 26
4.4 Hoare logic for total correctness 27

1 Introduction

These are the theories underlying the publications [2, 1]. They should be consulted for explanatory text. The local variable declaration construct in [2] has been generalized; see Section 2.1.

2 Hoare Logics for While

theory *Lang* **imports** *Main* **begin**

2.1 The language

We start by declaring a type of states:

typedecl *state*

Our approach is completely parametric in the state space. We define expressions (*bexp*) as functions from states to the booleans:

type-synonym *bexp* = *state* ⇒ *bool*

Instead of modelling the syntax of boolean expressions, we model their semantics. The (abstract and concrete) syntax of our programming is defined as a recursive datatype:

datatype *com* = *Do* (*state* ⇒ *state set*)
| *Semi* *com com* (-; - [*60*, *60*] *10*)
| *Cond* *bexp com com* (*IF* - *THEN* - *ELSE* - *60*)
| *While* *bexp com* (*WHILE* - *DO* - *60*)
| *Local* (*state* ⇒ *state*) *com* (*state* ⇒ *state* ⇒ *state*)
(*LOCAL* -; -; - [*0*,*0*,*60*] *60*)

Statements in this language are called *commands*. They are modelled as terms of type *com*. *Do f* represents an atomic nondeterministic command that changes the state from *s* to some element of *f s*. Thus the command that does nothing, often called `skip`, can be represented by *Do* (λ*s*. {*s*}). Again we have chosen to model the semantics rather than the syntax, which simplifies matters enormously. Of course it means that we can no longer talk about certain syntactic matters, but that is just fine.

The constructors *Semi*, *Cond* and *While* represent sequential composition, conditional and while-loop. The annotations allow us to write

$$c_1;\ c_2 \qquad \mathit{IF}\ b\ \mathit{THEN}\ c_1\ \mathit{ELSE}\ c_2 \qquad \mathit{WHILE}\ b\ \mathit{DO}\ c$$

instead of *Semi* c_1 c_2, *Cond* *b* c_1 c_2 and *While* *b* *c*.

The command *LOCAL f*; *c*; *g* applies function *f* to the state, executes *c*, and then combines initial and final state via function *g*. More below. The semantics of commands is defined inductively by a so-called big-step semantics.

inductive
exec :: *state* ⇒ *com* ⇒ *state* ⇒ *bool* (-/ −-→/ - [*50*,*0*,*50*] *50*)
where
t ∈ *f s* ⟹ *s* −*Do f*→ *t*

| ⟦ *s0* −*c1*→ *s1*; *s1* −*c2*→ *s2* ⟧ ⟹ *s0* −*c1*;*c2*→ *s2*

$| \llbracket\ b\ s;\ s -c1\rightarrow t \rrbracket \Longrightarrow s -IF\ b\ THEN\ c1\ ELSE\ c2\rightarrow t$
$| \llbracket \neg b\ s;\ s -c2\rightarrow t \rrbracket \Longrightarrow s -IF\ b\ THEN\ c1\ ELSE\ c2\rightarrow t$

$| \neg b\ s \Longrightarrow s -WHILE\ b\ DO\ c\rightarrow s$
$| \llbracket\ b\ s;\ s -c\rightarrow t;\ t -WHILE\ b\ DO\ c\rightarrow u \rrbracket \Longrightarrow s -WHILE\ b\ DO\ c\rightarrow u$

$|\ f\ s -c\rightarrow t \Longrightarrow s -LOCAL\ f;\ c;\ g\rightarrow g\ s\ t$

Assuming that the state is a function from variables to values, the declaration of a new local variable x with inital value a can be modelled as $LOCAL\ (\lambda s.\ s(x := a\ s));\ c;\ (\lambda s\ t.\ t(x := s\ x))$.

lemma *exec-Do-iff*[*iff*]: $(s -Do\ f\rightarrow t) = (t \in f\ s)$
⟨*proof*⟩

lemma [*iff*]: $(s -c;d\rightarrow u) = (\exists t.\ s -c\rightarrow t \wedge t -d\rightarrow u)$
⟨*proof*⟩

lemma [*iff*]: $(s -IF\ b\ THEN\ c\ ELSE\ d\rightarrow t) =$
$(s -if\ b\ s\ then\ c\ else\ d\rightarrow t)$
⟨*proof*⟩

lemma [*iff*]: $(s -LOCAL\ f;\ c;\ g\rightarrow u) = (\exists t.\ f\ s -c\rightarrow t \wedge u = g\ s\ t)$
⟨*proof*⟩

lemma *unfold-while*:
$(s -WHILE\ b\ DO\ c\rightarrow u) =$
$(s -IF\ b\ THEN\ c;WHILE\ b\ DO\ c\ ELSE\ Do(\lambda s.\ \{s\})\rightarrow u)$
⟨*proof*⟩

lemma *while-lemma*[*rule-format*]:
$s -w\rightarrow t \Longrightarrow \forall b\ c.\ w = WHILE\ b\ DO\ c \wedge P\ s \wedge$
$(\forall s\ s'.\ P\ s \wedge b\ s \wedge s -c\rightarrow s' \longrightarrow P\ s') \longrightarrow P\ t \wedge \neg b\ t$
⟨*proof*⟩

lemma *while-rule*:
$\llbracket s -WHILE\ b\ DO\ c\rightarrow t;\ P\ s;\ \forall s\ s'.\ P\ s \wedge b\ s \wedge s -c\rightarrow s' \longrightarrow P\ s' \rrbracket$
$\Longrightarrow P\ t \wedge \neg b\ t$
⟨*proof*⟩

end

theory *Hoare* **imports** *Lang* **begin**

2.2 Hoare logic for partial correctness

We continue our semantic approach by modelling assertions just like boolean expressions, i.e. as functions:

type-synonym *assn* = *state* ⇒ *bool*

Hoare triples are triples of the form $\{P\}\ c\ \{Q\}$, where the assertions P and Q are the so-called pre and postconditions. Such a triple is *valid* (denoted by $\models$) iff every (terminating) execution starting in a state satisfying P ends up in a state satisfying Q:

definition
hoare-valid :: *assn* ⇒ *com* ⇒ *assn* ⇒ *bool* (⊨ {(*1*-)}/ (-)/ {(*1*-)} *50*) **where**
⊨ {*P*}*c*{*Q*} ⟷ (∀ *s t*. *s* −*c*→ *t* ⟶ *P s* ⟶ *Q t*)

This notion of validity is called *partial correctness* because it does not require termination of c.

Provability in Hoare logic is indicated by $\vdash$ and defined inductively:

inductive
hoare :: *assn* ⇒ *com* ⇒ *assn* ⇒ *bool* (⊢ ({(*1*-)}/ (-)/ {(*1*-)}) *50*)
where
⊢ {λ*s*. ∀ *t* ∈ *f s*. *P t*} *Do f* {*P*}

| ⟦ ⊢ {*P*}*c1*{*Q*}; ⊢ {*Q*}*c2*{*R*} ⟧ ⟹ ⊢ {*P*} *c1*;*c2* {*R*}

| ⟦ ⊢ {λ*s*. *P s* ∧ *b s*} *c1* {*Q*}; ⊢ {λ*s*. *P s* ∧ ¬*b s*} *c2* {*Q*} ⟧
⟹ ⊢ {*P*} *IF b THEN c1 ELSE c2* {*Q*}

| ⊢ {λ*s*. *P s* ∧ *b s*} *c* {*P*} ⟹ ⊢ {*P*} *WHILE b DO c* {λ*s*. *P s* ∧ ¬*b s*}

| ⟦ ∀ *s*. *P*′ *s* ⟶ *P s*; ⊢ {*P*}*c*{*Q*}; ∀ *s*. *Q s* ⟶ *Q*′ *s* ⟧ ⟹ ⊢ {*P*′}*c*{*Q*′}

| ⟦ ⋀*s*. *P s* ⟹ *P*′ *s* (*f s*); ∀ *s*. ⊢ {*P*′ *s*} *c* {*Q* ∘ (*g s*)} ⟧ ⟹
⊢ {*P*} *LOCAL f*;*c*;*g* {*Q*}

Soundness is proved by induction on the derivation of $\vdash \{P\}\ c\ \{Q\}$:

theorem *hoare-sound*: ⊢ {*P*}*c*{*Q*} ⟹ ⊨ {*P*}*c*{*Q*}
⟨*proof*⟩

Completeness is not quite as straightforward, but still easy. The proof is best explained in terms of the *weakest precondition*:

definition
wp :: *com* ⇒ *assn* ⇒ *assn* **where**
wp c Q = (λ*s*. ∀ *t*. *s* −*c*→ *t* ⟶ *Q t*)

Dijkstra calls this the weakest *liberal* precondition to emphasize that it corresponds to partial correctness. We use "weakest precondition" all the time and let the context determine if we talk about partial or total correctness — the latter is introduced further below.

The following lemmas about *wp* are easily derived:

lemma [*simp*]: *wp* (*Do f*) *Q* = (λ*s*. ∀ *t* ∈ *f s*. *Q*(*t*))
⟨*proof*⟩

lemma [*simp*]: $wp\ (c_1;c_2)\ R = wp\ c_1\ (wp\ c_2\ R)$
⟨*proof*⟩

lemma [*simp*]:
$wp\ (IF\ b\ THEN\ c_1\ ELSE\ c_2)\ Q = (\lambda s.\ wp\ (if\ b\ s\ then\ c_1\ else\ c_2)\ Q\ s)$
⟨*proof*⟩

lemma *wp-while*:
$wp\ (WHILE\ b\ DO\ c)\ Q =$
$(\lambda s.\ if\ b\ s\ then\ wp\ (c;WHILE\ b\ DO\ c)\ Q\ s\ else\ Q\ s)$
⟨*proof*⟩

lemma [*simp*]:
$wp\ (LOCAL\ f;c;g)\ Q = (\lambda s.\ wp\ c\ (Q\ o\ (g\ s))\ (f\ s))$
⟨*proof*⟩

lemma *strengthen-pre*: $[\![\ \forall s.\ P'\ s \longrightarrow P\ s; \vdash \{P\}c\{Q\}\ ]\!] \Longrightarrow \vdash \{P'\}c\{Q\}$
⟨*proof*⟩

lemma *weaken-post*: $[\![\vdash \{P\}c\{Q\}; \forall s.\ Q\ s \longrightarrow Q'\ s\ ]\!] \Longrightarrow \vdash \{P\}c\{Q'\}$
⟨*proof*⟩

By induction on c one can easily prove

lemma *wp-is-pre*[*rule-format*]: $\vdash \{wp\ c\ Q\}\ c\ \{Q\}$
⟨*proof*⟩

from which completeness follows more or less directly via the rule of consequence:

theorem *hoare-relative-complete*: $\models \{P\}c\{Q\} \implies \vdash \{P\}c\{Q\}$
⟨*proof*⟩

end

theory *Termi* **imports** *Lang* **begin**

2.3 Termination

Although partial correctness appeals because of its simplicity, in many cases one would like the additional assurance that the command is guaranteed to termiate if started in a state that satisfies the precondition. Even to express this we need to define when a command is guaranteed to terminate. We can do this without modifying our existing semantics by merely adding a second inductively defined judgement $c{\downarrow}s$ that expresses guaranteed termination of c started in state s:

inductive
termi :: *com* ⇒ *state* ⇒ *bool* (**infixl** ↓ *50*)
where

$f\ s \neq \{\} \Longrightarrow Do\ f \downarrow s$

| $[\![\ c_1 \downarrow s_0;\ \forall s_1.\ s_0 -c_1\rightarrow s_1 \longrightarrow c_2 \downarrow s_1\ ]\!] \Longrightarrow (c_1;c_2) \downarrow s_0$

| $[\![\ b\ s;\ c_1 \downarrow s\ ]\!] \Longrightarrow IF\ b\ THEN\ c_1\ ELSE\ c_2 \downarrow s$
| $[\![\ \neg b\ s;\ c_2 \downarrow s\ ]\!] \Longrightarrow IF\ b\ THEN\ c_1\ ELSE\ c_2 \downarrow s$

| $\neg b\ s \Longrightarrow WHILE\ b\ DO\ c \downarrow s$
| $[\![\ b\ s;\ c \downarrow s;\ \forall t.\ s -c\rightarrow t \longrightarrow WHILE\ b\ DO\ c \downarrow t\ ]\!] \Longrightarrow WHILE\ b\ DO\ c \downarrow s$

| $c \downarrow f\ s \Longrightarrow LOCAL\ f;c;g \downarrow s$

lemma [*iff*]: $Do\ f \downarrow s = (f\ s \neq \{\})$
⟨*proof*⟩

lemma [*iff*]: $((c_1;c_2) \downarrow s_0) = (c_1 \downarrow s_0 \wedge (\forall s_1.\ s_0 -c_1\rightarrow s_1 \longrightarrow c_2 \downarrow s_1))$
⟨*proof*⟩

lemma [*iff*]: $(IF\ b\ THEN\ c_1\ ELSE\ c_2 \downarrow s) =$
$((if\ b\ s\ then\ c_1\ else\ c_2) \downarrow s)$
⟨*proof*⟩

lemma [*iff*]: $(LOCAL\ f;c;g \downarrow s) = (c \downarrow f\ s)$
⟨*proof*⟩

lemma *termi-while-lemma*[*rule-format*]:
$w{\downarrow}fk \Longrightarrow$
$(\forall k\ b\ c.\ fk = f\ k \wedge w = WHILE\ b\ DO\ c \wedge (\forall i.\ f\ i -c\rightarrow f(Suc\ i))$
$\longrightarrow (\exists i.\ \neg b(f\ i)))$
⟨*proof*⟩

lemma *termi-while*:
$[\![\ (WHILE\ b\ DO\ c) \downarrow f\ k;\ \ \forall i.\ f\ i -c\rightarrow f(Suc\ i)\ ]\!] \Longrightarrow \exists i.\ \neg b(f\ i)$
⟨*proof*⟩

lemma *wf-termi*: $wf\ \{(t,s).\ WHILE\ b\ DO\ c \downarrow s \wedge b\ s \wedge s -c\rightarrow t\}$
⟨*proof*⟩

end

theory *HoareTotal* **imports** *Hoare Termi* **begin**

2.4 Hoare logic for total correctness

Now that we have termination, we can define total validity, $\models_t$, as partial validity and guaranteed termination:

definition

hoare-tvalid :: *assn* ⇒ *com* ⇒ *assn* ⇒ *bool* (⊨$_t$ {(*1-*)}/ (-)/ {(*1-*)} *50*) **where**
⊨$_t$ {*P*}*c*{*Q*} ⟷ ⊨ {*P*}*c*{*Q*} ∧ (∀ *s*. *P s* ⟶ *c*↓*s*)

Proveability of Hoare triples in the proof system for total correctness is written ⊢$_t$ {*P*}*c*{*Q*} and defined inductively. The rules for ⊢$_t$ differ from those for ⊢ only in the one place where nontermination can arise: the *While*-rule.

inductive
thoare :: *assn* ⇒ *com* ⇒ *assn* ⇒ *bool* (⊢$_t$ ({(*1-*)}/ (-)/ {(*1-*)}) *50*)
where
Do: ⊢$_t$ {λ*s*. (∀ *t* ∈ *f s*. *P t*) ∧ *f s* ≠ {}} *Do f* {*P*}
| *Semi*: ⟦ ⊢$_t$ {*P*}*c*{*Q*}; ⊢$_t$ {*Q*}*d*{*R*} ⟧ ⟹ ⊢$_t$ {*P*} *c*;*d* {*R*}
| *If*: ⟦ ⊢$_t$ {λ*s*. *P s* ∧ *b s*}*c*{*Q*}; ⊢$_t$ {λ*s*. *P s* ∧ ~*b s*}*d*{*Q*} ⟧ ⟹
⊢$_t$ {*P*} *IF b THEN c ELSE d* {*Q*}
| *While*:
⟦*wf r*; ∀ *s*′. ⊢$_t$ {λ*s*. *P s* ∧ *b s* ∧ *s*′ = *s*} *c* {λ*s*. *P s* ∧ (*s*,*s*′) ∈ *r*}⟧
⟹ ⊢$_t$ {*P*} *WHILE b DO c* {λ*s*. *P s* ∧ ¬*b s*}
| *Conseq*: ⟦ ∀ *s*. *P*′ *s* ⟶ *P s*; ⊢$_t$ {*P*}*c*{*Q*}; ∀ *s*. *Q s* ⟶ *Q*′ *s* ⟧ ⟹
⊢$_t$ {*P*′}*c*{*Q*′}

| *Local*: (!!*s*. *P s* ⟹ *P*′ *s* (*f s*)) ⟹ ∀ *p*. ⊢$_t$ {*P*′ *p*} *c* {*Q o* (*g p*)} ⟹
⊢$_t$ {*P*} *LOCAL f*;*c*;*g* {*Q*}

The *While*- rule is like the one for partial correctness but it requires additionally that with every execution of the loop body a wellfounded relation (*wf r*) on the state space decreases.

The soundness theorem

theorem ⊢$_t$ {*P*}*c*{*Q*} ⟹ ⊨$_t$ {*P*}*c*{*Q*}
⟨*proof*⟩

In the *While*-case we perform a local proof by wellfounded induction over the given relation *r*.

The completeness proof proceeds along the same lines as the one for partial correctness. First we have to strengthen our notion of weakest precondition to take termination into account:

definition
wpt :: *com* ⇒ *assn* ⇒ *assn* (*wp*$_t$) **where**
wp$_t$ *c Q* = (λ*s*. *wp c Q s* ∧ *c*↓*s*)

lemmas *wp-defs* = *wp-def wpt-def*

lemma [*simp*]: *wp*$_t$ (*Do f*) *Q* = (λ*s*. (∀ *t* ∈ *f s*. *Q t*) ∧ *f s* ≠ {})
⟨*proof*⟩

lemma [*simp*]: *wp*$_t$ (*c*$_1$;*c*$_2$) *R* = *wp*$_t$ *c*$_1$ (*wp*$_t$ *c*$_2$ *R*)
⟨*proof*⟩

lemma [*simp*]:
$wp_t\ (IF\ b\ THEN\ c_1\ ELSE\ c_2)\ Q = (\lambda s.\ wp_t\ (if\ b\ s\ then\ c_1\ else\ c_2)\ Q\ s)$
⟨*proof*⟩

lemma [*simp*]: $wp_t\ (LOCAL\ f;c;g)\ Q = (\lambda s.\ wp_t\ c\ (Q\ o\ (g\ s))\ (f\ s))$
⟨*proof*⟩

lemma *strengthen-pre*: $[\![\ \forall s.\ P'\ s \longrightarrow P\ s; \vdash_t \{P\}c\{Q\}\ ]\!] \Longrightarrow \vdash_t \{P'\}c\{Q\}$
⟨*proof*⟩

lemma *weaken-post*: $[\![\ \vdash_t \{P\}c\{Q\}; \forall s.\ Q\ s \longrightarrow Q'\ s\ ]\!] \Longrightarrow \vdash_t \{P\}c\{Q'\}$
⟨*proof*⟩

inductive-cases [*elim*!]: *WHILE b DO c* ↓ *s*

lemma *wp-is-pre*[*rule-format*]: $\vdash_t \{wp_t\ c\ Q\}\ c\ \{Q\}$
⟨*proof*⟩

The *While*-case is interesting because we now have to furnish a suitable wellfounded relation. Of course the execution of the loop body directly yields the required relation. The actual completeness theorem follows directly, in the same manner as for partial correctness.

theorem $\models_t \{P\}c\{Q\} \Longrightarrow \vdash_t \{P\}c\{Q\}$
⟨*proof*⟩

end

3 Hoare Logics for 1 Procedure

theory *PLang* **imports** *Main* **begin**

3.1 The language

typedecl *state*

type-synonym *bexp* = *state* ⇒ *bool*

datatype *com* = *Do* (*state* ⇒ *state set*)
| *Semi com com* (-; - [*60*, *60*] *10*)
| *Cond bexp com com* (*IF* - *THEN* - *ELSE* - *60*)
| *While bexp com* (*WHILE* - *DO* - *60*)
| *CALL*
| *Local* (*state* ⇒ *state*) *com* (*state* ⇒ *state* ⇒ *state*)
(*LOCAL* -; -; - [*0*,*0*,*60*] *60*)

There is only one parameterless procedure in the program. Hence *CALL* does not even need to mention the procedure name. There is no separate syntax for procedure declarations. Instead we declare a HOL constant that represents the body of the one procedure in the program.

consts *body* :: *com*

As before, command execution is described by transitions $s\ -c\rightarrow\ t$. The only new rule is the one for *CALL* — it requires no comment:

inductive
  exec :: *state* ⇒ *com* ⇒ *state* ⇒ *bool* (-/ −-→/ - [50,0,50] 50)
where
  Do: $t \in f\ s \Longrightarrow s\ -Do\ f\rightarrow\ t$

| *Semi*: ⟦ *s0* −*c1*→ *s1*; *s1* −*c2*→ *s2* ⟧
  ⟹ *s0* −*c1*;*c2*→ *s2*

| *IfTrue*: ⟦ *b s*; *s* −*c1*→ *t* ⟧ ⟹ *s* −*IF b THEN c1 ELSE c2*→ *t*
| *IfFalse*: ⟦ ¬*b s*; *s* −*c2*→ *t* ⟧ ⟹ *s* −*IF b THEN c1 ELSE c2*→ *t*

| *WhileFalse*: ¬*b s* ⟹ *s* −*WHILE b DO c*→ *s*
| *WhileTrue*: ⟦ *b s*; *s* −*c*→ *t*; *t* −*WHILE b DO c*→ *u* ⟧
  ⟹ *s* −*WHILE b DO c*→ *u*

| *s* −*body*→ *t* ⟹ *s* −*CALL*→ *t*

| *Local*: *f s* −*c*→ *t* ⟹ *s* −*LOCAL f*; *c*; *g*→ *g s t*

lemma [*iff*]: (*s* −*Do f*→ *t*) = (*t* ∈ *f s*)
⟨*proof*⟩

lemma [*iff*]: (*s* −*c*;*d*→ *u*) = (∃ *t*. *s* −*c*→ *t* ∧ *t* −*d*→ *u*)
⟨*proof*⟩

lemma [*iff*]: (*s* −*IF b THEN c ELSE d*→ *t*) =
  (*s* −*if b s then c else d*→ *t*)
⟨*proof*⟩

lemma *unfold-while*:
 (*s* −*WHILE b DO c*→ *u*) =
  (*s* −*IF b THEN c*;*WHILE b DO c ELSE Do*(λ*s*. {*s*})→ *u*)
⟨*proof*⟩

lemma [*iff*]: (*s* −*CALL*→ *t*) = (*s* −*body*→ *t*)
⟨*proof*⟩

lemma [*iff*]: (*s* −*LOCAL f*; *c*; *g*→ *u*) = (∃ *t*. *f s* −*c*→ *t* ∧ *u* = *g s t*)
⟨*proof*⟩

lemma [*simp*]: ¬*b s* ⟹ *s* −*WHILE b DO c*→ *s*
⟨*proof*⟩

lemma *WhileI*: ⟦*b s*; *s* −*c*→ *t*; *t* −*WHILE b DO c*→ *u*⟧ ⟹ *s* −*WHILE b DO*

$c\rightarrow\ u$
⟨*proof*⟩

This semantics turns out not to be fine-grained enough. The soundness proof for the Hoare logic below proceeds by induction on the call depth during execution. To make this work we define a second semantics $s\ -c-n\rightarrow\ t$ which expresses that the execution uses at most n nested procedure invocations, where n is a natural number. The rules are straightforward: n is just passed around, except for procedure calls, where it is decremented:

inductive
  execn :: *state* ⇒ *com* ⇒ *nat* ⇒ *state* ⇒ *bool* (-/ -----→/ - [50,0,0,50] 50)
where
  $t \in f\ s \Longrightarrow s\ -Do\ f-n\rightarrow\ t$

| ⟦ *s0* $-c1-n\rightarrow$ *s1*; *s1* $-c2-n\rightarrow$ *s2* ⟧ ⟹ *s0* $-c1;c2-n\rightarrow$ *s2*

| ⟦ *b s*; $s\ -c1-n\rightarrow\ t$ ⟧ ⟹ $s\ -IF\ b\ THEN\ c1\ ELSE\ c2-n\rightarrow\ t$
| ⟦ ¬*b s*; $s\ -c2-n\rightarrow\ t$ ⟧ ⟹ $s\ -IF\ b\ THEN\ c1\ ELSE\ c2-n\rightarrow\ t$

| ¬*b s* ⟹ $s\ -WHILE\ b\ DO\ c-n\rightarrow\ s$
| ⟦*b s*; $s\ -c-n\rightarrow\ t$; $t\ -WHILE\ b\ DO\ c-n\rightarrow\ u$⟧ ⟹ $s\ -WHILE\ b\ DO\ c-n\rightarrow\ u$

| $s\ -body-n\rightarrow\ t \Longrightarrow s\ -CALL-Suc\ n\rightarrow\ t$

| $f\ s\ -c-n\rightarrow\ t \Longrightarrow s\ -LOCAL\ f;\ c;\ g-n\rightarrow\ g\ s\ t$

lemma [*iff*]: $(s\ -Do\ f-n\rightarrow\ t) = (t \in f\ s)$
⟨*proof*⟩

lemma [*iff*]: $(s\ -c1;c2-n\rightarrow\ u) = (\exists\, t.\ s\ -c1-n\rightarrow\ t \wedge t\ -c2-n\rightarrow\ u)$
⟨*proof*⟩

lemma [*iff*]: $(s\ -IF\ b\ THEN\ c\ ELSE\ d-n\rightarrow\ t) =$
$(s\ -if\ b\ s\ then\ c\ else\ d-n\rightarrow\ t)$
⟨*proof*⟩

lemma [*iff*]: $(s\ -CALL-\ 0\rightarrow\ t) = False$
⟨*proof*⟩

lemma [*iff*]: $(s\ -CALL-Suc\ n\rightarrow\ t) = (s\ -body-n\rightarrow\ t)$
⟨*proof*⟩

lemma [*iff*]: $(s\ -LOCAL\ f;\ c;\ g-n\rightarrow\ u) = (\exists\, t.\ f\ s\ -c-n\rightarrow\ t \wedge u = g\ s\ t)$
⟨*proof*⟩

By induction on $s\ -c-m\rightarrow\ t$ we show monotonicity w.r.t. the call depth:

lemma *exec-mono*[*rule-format*]: $s\ -c-m\rightarrow\ t \Longrightarrow \forall\, n.\ m \leq n \longrightarrow s\ -c-n\rightarrow\ t$
⟨*proof*⟩

With the help of this lemma we prove the expected relationship between the two semantics:

lemma *exec-iff-execn*: $(s\ -c\rightarrow\ t) = (\exists\, n.\ s\ -c-n\rightarrow\ t)$
⟨*proof*⟩

lemma *while-lemma*[*rule-format*]:
$s\ -w-n\rightarrow\ t \Longrightarrow \forall\, b\ c.\ w = \mathit{WHILE}\ b\ \mathit{DO}\ c \wedge P\ s \wedge$
$(\forall\, s\ s'.\ P\ s \wedge b\ s \wedge s\ -c-n\rightarrow\ s' \longrightarrow P\ s') \longrightarrow P\ t \wedge \neg b\ t$
⟨*proof*⟩

lemma *while-rule*:
$[\![s\ -\mathit{WHILE}\ b\ \mathit{DO}\ c-n\rightarrow\ t;\ P\ s;\ \bigwedge s\ s'.\ [\![P\ s;\ b\ s;\ s\ -c-n\rightarrow\ s'\,]\!] \Longrightarrow P\ s']\!]$
$\Longrightarrow P\ t \wedge \neg b\ t$
⟨*proof*⟩

end

theory *PHoare* **imports** *PLang* **begin**

3.2 Hoare logic for partial correctness

Taking auxiliary variables seriously means that assertions must now depend on them as well as on the state. Initially we do not fix the type of auxiliary variables but parameterize the type of assertions with a type variable $'a$:

type-synonym $'a\ assn = 'a \Rightarrow state \Rightarrow bool$

The second major change is the need to reason about Hoare triples in a context: proofs about recursive procedures are conducted by induction where we assume that all *CALL*s satisfy the given pre/postconditions and have to show that the body does as well. The assumption is stored in a context, which is a set of Hoare triples:

type-synonym $'a\ cntxt = ('a\ assn \times com \times 'a\ assn)set$

In the presence of only a single procedure the context will always be empty or a singleton set. With multiple procedures, larger sets can arise.

Now that we have contexts, validity becomes more complicated. Ordinary validity (w.r.t. partial correctness) is still what it used to be, except that we have to take auxiliary variables into account as well:

definition
$valid :: 'a\ assn \Rightarrow com \Rightarrow 'a\ assn \Rightarrow bool\ (\models \{(1\text{-})\}/\ (\text{-})/\ \{(1\text{-})\}\ 50)$ **where**
$\models \{P\}c\{Q\} \longleftrightarrow (\forall\, s\ t.\ s\ -c\rightarrow\ t \longrightarrow (\forall\, z.\ P\ z\ s \longrightarrow Q\ z\ t))$

Auxiliary variables are always denoted by z.

Validity of a context and validity of a Hoare triple in a context are defined as follows:

definition
valids :: *'a cntxt* ⇒ *bool* (|⊨ - *50*) **where**
[*simp*]: |⊨ $C \equiv (\forall (P,c,Q) \in C.\ \models \{P\}c\{Q\})$

definition
cvalid :: *'a cntxt* ⇒ *'a assn* ⇒ *com* ⇒ *'a assn* ⇒ *bool* (- ⊨/ {(*1*-)}/ (-)/ {(*1*-)} *50*) **where**
$C \models \{P\}c\{Q\} \longleftrightarrow$ |⊨ $C \longrightarrow \models \{P\}c\{Q\}$

Note that $\{\} \models \{P\}\ c\ \{Q\}$ is equivalent to $\models \{P\}\ c\ \{Q\}$.

Unfortunately, this is not the end of it. As we have two semantics, $-c\rightarrow$ and $-c-n\rightarrow$, we also need a second notion of validity parameterized with the recursion depth n:

definition
nvalid :: *nat* ⇒ *'a assn* ⇒ *com* ⇒ *'a assn* ⇒ *bool* (⊨- {(*1*-)}/ (-)/ {(*1*-)} *50*) **where**
$\models n\ \{P\}c\{Q\} \equiv (\forall s\ t.\ s\ -c-n\rightarrow\ t \longrightarrow (\forall z.\ P\ z\ s \longrightarrow Q\ z\ t))$

definition
nvalids :: *nat* ⇒ *'a cntxt* ⇒ *bool* (|⊨*'*--/ - *50*) **where**
|⊨-*n* $C \equiv (\forall (P,c,Q) \in C.\ \models n\ \{P\}c\{Q\})$

definition
cnvalid :: *'a cntxt* ⇒ *nat* ⇒ *'a assn* ⇒ *com* ⇒ *'a assn* ⇒ *bool* (- ⊨-/ {(*1*-)}/ (-)/ {(*1*-)} *50*) **where**
$C \models n\ \{P\}c\{Q\} \longleftrightarrow$ |⊨-*n* $C \longrightarrow \models n\ \{P\}c\{Q\}$

Finally we come to the proof system for deriving triples in a context:

inductive
hoare :: *'a cntxt* ⇒ *'a assn* ⇒ *com* ⇒ *'a assn* ⇒ *bool* (- ⊢/ ({(*1*-)}/ (-)/ {(*1*-)}) *50*)
where
$C \vdash \{\lambda z\ s.\ \forall t \in f\ s\ .\ P\ z\ t\}\ Do\ f\ \{P\}$

| $[\![\ C \vdash \{P\}c1\{Q\};\ C \vdash \{Q\}c2\{R\}\ ]\!] \Longrightarrow C \vdash \{P\}\ c1;c2\ \{R\}$

| $[\![\ C \vdash \{\lambda z\ s.\ P\ z\ s \wedge b\ s\}c1\{Q\};\ \ C \vdash \{\lambda z\ s.\ P\ z\ s \wedge \neg b\ s\}c2\{Q\}\ \ ]\!]$
$\Longrightarrow C \vdash \{P\}\ IF\ b\ THEN\ c1\ ELSE\ c2\ \{Q\}$

| $C \vdash \{\lambda z\ s.\ P\ z\ s \wedge b\ s\}\ c\ \{P\}$
$\Longrightarrow C \vdash \{P\}\ WHILE\ b\ DO\ c\ \{\lambda z\ s.\ P\ z\ s \wedge \neg b\ s\}$

| $[\![\ C \vdash \{P'\}c\{Q'\};\ \forall s\ t.\ (\forall z.\ P'\ z\ s \longrightarrow Q'\ z\ t) \longrightarrow (\forall z.\ P\ z\ s \longrightarrow Q\ z\ t)\ ]\!]$
$\Longrightarrow C \vdash \{P\}c\{Q\}$

| $\{(P,CALL,Q)\} \vdash \{P\}body\{Q\} \Longrightarrow \{\} \vdash \{P\}\ CALL\ \{Q\}$

$| \{(P,CALL,Q)\} \vdash \{P\}\ CALL\ \{Q\}$
$| \ [\![\forall s'.\ C \vdash \{\lambda z\ s.\ P\ z\ s' \wedge s = f\ s'\}\ c\ \{\lambda z\ t.\ Q\ z\ (g\ s'\ t)\}]\!] \Longrightarrow$
$C \vdash \{P\}\ LOCAL\ f;c;g\ \{Q\}$

abbreviation *hoare1* :: *'a cntxt* ⇒ *'a assn* × *com* × *'a assn* ⇒ *bool* (- ⊢ -) **where**
$C \vdash x \equiv C \vdash \{fst\ x\}fst\ (snd\ x)\{snd\ (snd\ x)\}$

The first four rules are familiar, except for their adaptation to auxiliary variables. The *CALL* rule embodies induction and has already been motivated above. Note that it is only applicable if the context is empty. This shows that we never need nested induction. For the same reason the assumption rule (the last rule) is stated with just a singleton context.

The rule of consequence is explained in the accompanying paper.

lemma *strengthen-pre*:
$[\![\forall z\ s.\ P'\ z\ s \longrightarrow P\ z\ s;\ C \vdash \{P\}c\{Q\}\ ]\!] \Longrightarrow C \vdash \{P'\}c\{Q\}$
⟨*proof*⟩

lemmas *valid-defs* = *valid-def valids-def cvalid-def*
nvalid-def nvalids-def cnvalid-def

theorem *hoare-sound*: $C \vdash \{P\}c\{Q\} \implies C \models \{P\}c\{Q\}$

requires a generalization: $\forall n.\ C \models n\ \{P\}\ c\ \{Q\}$ is proved instead, from which the actual theorem follows directly via lemma *exec-iff-execn* in §**??**. The generalization is proved by induction on *c*. The reason for the generalization is that soundness of the *CALL* rule is proved by induction on the maximal call depth, i.e. *n*.

⟨*proof*⟩

The completeness proof employs the notion of a *most general triple* (or *most general formula*):

definition
MGT :: *com* ⇒ *state assn* × *com* × *state assn* **where**
$MGT\ c = (\lambda z\ s.\ z = s,\ c,\ \lambda z\ t.\ z\ -c\rightarrow\ t)$

declare *MGT-def*[*simp*]

Note that the type of *z* has been identified with *state*. This means that for every state variable there is an auxiliary variable, which is simply there to record the value of the program variables before execution of a command. This is exactly what, for example, VDM offers by allowing you to refer to the pre-value of a variable in a postcondition. The intuition behind $MGT\ c$ is that it completely describes the operational behaviour of *c*. It is easy to see that, in the presence of the new consequence rule, $\{\} \vdash MGT\ c$ implies completeness:

lemma *MGT-implies-complete*:
$\{\} \vdash MGT\ c \implies \{\} \models \{P\}c\{Q\} \implies \{\} \vdash \{P\}c\{Q::state\ assn\}$
⟨*proof*⟩

In order to discharge $\{\} \vdash MGT\ c$ one proves

lemma *MGT-lemma*: $C \vdash MGT\ CALL \implies C \vdash MGT\ c$
⟨*proof*⟩

The proof is by induction on c. In the *While*-case it is easy to show that $\lambda z\ t.\ (z,\ t) \in \{(s,\ t).\ b\ s \wedge s\ -c\rightarrow\ t\}^*$ is invariant. The precondition $\lambda z\ s.\ z{=}s$ establishes the invariant and a reflexive transitive closure induction shows that the invariant conjoined with $\neg\ b\ t$ implies the postcondition $\lambda z.\ exec\ z\ (WHILE\ b\ DO\ c)$. The remaining cases are trivial.

Using the *MGT-lemma* (together with the *CALL* and the assumption rule) one can easily derive

lemma *MGT-CALL*: $\{\} \vdash MGT\ CALL$
⟨*proof*⟩

Using the *MGT-lemma* once more we obtain $\{\} \vdash MGT\ c$ and thus by *MGT-implies-complete* completeness.

theorem $\{\} \models \{P\}c\{Q\} \implies \{\} \vdash \{P\}c\{Q{::}state\ assn\}$
⟨*proof*⟩

end

theory *PTermi* **imports** *PLang* **begin**

3.3 Termination

inductive
  termi :: *com* ⇒ *state* ⇒ *bool* (**infixl** ↓ *50*)
where
  Do[*iff*]: $f\ s \neq \{\} \implies Do\ f \downarrow s$
| *Semi*[*intro*!]: $[\![\ c1 \downarrow s0;\ \bigwedge s1.\ s0\ -c1\rightarrow\ s1 \implies c2 \downarrow s1\ ]\!]$
$\implies (c1;c2) \downarrow s0$

| *IfTrue*[*intro*,*simp*]: $[\![\ b\ s;\ c1 \downarrow s\ ]\!] \implies IF\ b\ THEN\ c1\ ELSE\ c2 \downarrow s$
| *IfFalse*[*intro*,*simp*]: $[\![\ \neg b\ s;\ c2 \downarrow s\ ]\!] \implies IF\ b\ THEN\ c1\ ELSE\ c2 \downarrow s$

| *WhileFalse*: $\neg b\ s \implies WHILE\ b\ DO\ c \downarrow s$

| *WhileTrue*: $[\![\ b\ s;\ c \downarrow s;\ \bigwedge t.\ s\ -c\rightarrow\ t \implies WHILE\ b\ DO\ c \downarrow t\ ]\!]$
$\implies WHILE\ b\ DO\ c \downarrow s$
| *body* ↓ *s* ⟹ *CALL* ↓ *s*

| *Local*: $c \downarrow f\ s \implies LOCAL\ f;c;g \downarrow s$

lemma [*iff*]: $(Do\ f \downarrow s) = (f\ s \neq \{\})$
⟨*proof*⟩

lemma [*iff*]: $((c1;c2) \downarrow s0) = (c1 \downarrow s0 \wedge (\forall s1.\ s0\ -c1\rightarrow\ s1 \longrightarrow c2 \downarrow s1))$
⟨*proof*⟩

lemma [*iff*]: (*IF b THEN c1 ELSE c2* $\downarrow$ *s*) =
((*if b s then c1 else c2*) $\downarrow$ *s*)
⟨*proof*⟩

lemma [*iff*]: (*CALL* $\downarrow$ *s*) = (*body* $\downarrow$ *s*)
⟨*proof*⟩

lemma [*iff*]: (*LOCAL f*;*c*;*g* $\downarrow$ *s*) = (*c* $\downarrow$ *f s*)
⟨*proof*⟩

lemma *termi-while-lemma*[*rule-format*]:
w$\downarrow$*fk* $\Longrightarrow$
($\forall$ *k b c. fk* = *f k* $\wedge$ *w* = *WHILE b DO c* $\wedge$ ($\forall$ *i. f i* $-c\rightarrow$ *f*(*Suc i*))
$\longrightarrow$ ($\exists$ *i.* $\neg$*b*(*f i*)))
⟨*proof*⟩

lemma *termi-while*:
⟦ (*WHILE b DO c*) $\downarrow$ *f k*; $\forall$ *i. f i* $-c\rightarrow$ *f*(*Suc i*) ⟧ $\Longrightarrow$ $\exists$ *i.* $\neg$*b*(*f i*)
⟨*proof*⟩

lemma *wf-termi*: *wf* {(*t*,*s*). *WHILE b DO c* $\downarrow$ *s* $\wedge$ *b s* $\wedge$ *s* $-c\rightarrow$ *t*}
⟨*proof*⟩

end

theory *PHoareTotal* **imports** *PHoare PTermi* **begin**

3.4 Hoare logic for total correctness

Validity is defined as expected:

definition
tvalid :: *'a assn* $\Rightarrow$ *com* $\Rightarrow$ *'a assn* $\Rightarrow$ *bool* ($\models_t$ {(*1-*)}/ (-)/ {(*1-*)} *50*) **where**
$\models_t$ {*P*}*c*{*Q*} $\longleftrightarrow$ $\models$ {*P*}*c*{*Q*} $\wedge$ ($\forall$ *z s. P z s* $\longrightarrow$ *c*$\downarrow$*s*)

definition
ctvalid :: *'a cntxt* $\Rightarrow$ *'a assn* $\Rightarrow$ *com* $\Rightarrow$ *'a assn* $\Rightarrow$ *bool*
((- /$\models_t$ {(*1-*)}/ (-)/ {(*1-*)}) *50*) **where**
C $\models_t$ {*P*}*c*{*Q*} $\longleftrightarrow$ ($\forall$ (*P'*,*c'*,*Q'*) $\in$ *C*. $\models_t$ {*P'*}*c'*{*Q'*}) $\longrightarrow$ $\models_t$ {*P*}*c*{*Q*}

inductive
thoare :: *'a cntxt* $\Rightarrow$ *'a assn* $\Rightarrow$ *com* $\Rightarrow$ *'a assn* $\Rightarrow$ *bool*
((- $\vdash_t$/ ({(*1-*)}/ (-)/ {(*1-*)})) [*50*,*0*,*0*,*0*] *50*)
where
Do: *C* $\vdash_t$ {λ*z s.* ($\forall$ *t* $\in$ *f s* . *P z t*) $\wedge$ *f s* $\neq$ {}} *Do f* {*P*}
| *Semi*: ⟦ *C* $\vdash_t$ {*P*}*c1*{*Q*}; *C* $\vdash_t$ {*Q*}*c2*{*R*} ⟧ $\Longrightarrow$ *C* $\vdash_t$ {*P*} *c1*;*c2* {*R*}
| *If*: ⟦ *C* $\vdash_t$ {λ*z s. P z s* $\wedge$ *b s*}*c*{*Q*}; *C* $\vdash_t$ {λ*z s. P z s* $\wedge$ ~*b s*}*d*{*Q*} ⟧ $\Longrightarrow$

$C \vdash_t \{P\}\ IF\ b\ THEN\ c\ ELSE\ d\ \{Q\}$

| *While*:
$[\![wf\ r;\ \forall s'.\ C \vdash_t \{\lambda z\ s.\ P\ z\ s \wedge b\ s \wedge s' = s\}\ c\ \{\lambda z\ s.\ P\ z\ s \wedge (s,s') \in r\}]\!]$
$\Longrightarrow C \vdash_t \{P\}\ WHILE\ b\ DO\ c\ \{\lambda z\ s.\ P\ z\ s \wedge \neg b\ s\}$

| *Call*:
$[\![wf\ r;\ \forall s'.\ \{(\lambda z\ s.\ P\ z\ s \wedge (s,s') \in r,\ CALL,\ Q)\}$
$\vdash_t \{\lambda z\ s.\ P\ z\ s \wedge s = s'\}\ body\ \{Q\}]\!]$
$\Longrightarrow \{\} \vdash_t \{P\}\ CALL\ \{Q\}$

| *Asm*: $\{(P,CALL,Q)\} \vdash_t \{P\}\ CALL\ \{Q\}$

| *Conseq*:
$[\![\ C \vdash_t \{P'\}c\{Q'\};$
$(\forall s\ t.\ (\forall z.\ P'\ z\ s \longrightarrow Q'\ z\ t) \longrightarrow (\forall z.\ P\ z\ s \longrightarrow Q\ z\ t)) \wedge$
$(\forall s.\ (\exists z.\ P\ z\ s) \longrightarrow (\exists z.\ P'\ z\ s))\ ]\!]$
$\Longrightarrow C \vdash_t \{P\}c\{Q\}$

| *Local*: $[\![\ \forall s'.\ C \vdash_t \{\lambda z\ s.\ P\ z\ s' \wedge s = f\ s'\}\ c\ \{\lambda z\ t.\ Q\ z\ (g\ s'\ t)\}\ ]\!] \Longrightarrow$
$C \vdash_t \{P\}\ LOCAL\ f;c;g\ \{Q\}$

abbreviation *hoare1* :: *'a cntxt* ⇒ *'a assn* × *com* × *'a assn* ⇒ *bool* (- ⊢$_t$ -) **where**
$C \vdash_t x \equiv C \vdash_t \{fst\ x\}fst\ (snd\ x)\{snd\ (snd\ x)\}$

The side condition in our rule of consequence looks quite different from the one by Kleymann, but the two are in fact equivalent:

lemma $((\forall s\ t.\ (\forall z.\ P'\ z\ s \longrightarrow Q'\ z\ t) \longrightarrow (\forall z.\ P\ z\ s \longrightarrow Q\ z\ t)) \wedge$
$(\forall s.\ (\exists z.\ P\ z\ s) \longrightarrow (\exists z.\ P'\ z\ s)))$
$= (\forall z\ s.\ P\ z\ s \longrightarrow (\forall t.\exists z'.\ P'\ z'\ s \wedge (Q'\ z'\ t \longrightarrow Q\ z\ t)))$
⟨*proof*⟩

The key difference to the work by Kleymann (and America and de Boer) is that soundness and completeness are shown for arbitrary, i.e. unbounded nondeterminism. This is a significant extension and appears to have been an open problem. The details are found below and are explained in a separate paper [1].

lemma *strengthen-pre*:
$[\![\ \forall z\ s.\ P'\ z\ s \longrightarrow P\ z\ s;\ C \vdash_t \{P\}c\{Q\}\ \ ]\!] \Longrightarrow C \vdash_t \{P'\}c\{Q\}$
⟨*proof*⟩

lemma *weaken-post*:
$[\![\ C \vdash_t \{P\}c\{Q\};\ \forall z\ s.\ Q\ z\ s \longrightarrow Q'\ z\ s\ ]\!] \Longrightarrow C \vdash_t \{P\}c\{Q'\}$
⟨*proof*⟩

lemmas *tvalid-defs = tvalid-def ctvalid-def valid-defs*

lemma [*iff*]:
$(\models_t \{\lambda z\ s.\ \exists n.\ P\ n\ z\ s\}c\{Q\}) = (\forall n.\ \models_t \{P\ n\}c\{Q\})$

⟨*proof*⟩

lemma [*iff*]:
$(\models_t \{\lambda z\ s.\ P\ z\ s \wedge P'\} c \{Q\}) = (P' \longrightarrow \models_t \{P\} c \{Q\})$
⟨*proof*⟩

lemma [*iff*]: $(\models_t \{P\} CALL \{Q\}) = (\models_t \{P\} body \{Q\})$
⟨*proof*⟩

theorem $C \vdash_t \{P\} c \{Q\} \implies C \models_t \{P\} c \{Q\}$
⟨*proof*⟩

definition $MGT_t :: com \Rightarrow state\ assn \times com \times state\ assn$ **where**
[*simp*]: $MGT_t\ c = (\lambda z\ s.\ z = s \wedge c{\downarrow}s,\ c,\ \lambda z\ t.\ z -c\rightarrow t)$

lemma *MGT-implies-complete*:
$\{\} \vdash_t MGT_t\ c \Longrightarrow \{\} \models_t \{P\} c \{Q\} \Longrightarrow \{\} \vdash_t \{P\} c \{Q :: state\ assn\}$
⟨*proof*⟩

lemma *while-termiE*: $[\![\ WHILE\ b\ DO\ c \downarrow s;\ b\ s\ ]\!] \Longrightarrow c \downarrow s$
⟨*proof*⟩

lemma *while-termiE2*:
$[\![\ WHILE\ b\ DO\ c \downarrow s;\ b\ s;\ s -c\rightarrow t\ ]\!] \Longrightarrow WHILE\ b\ DO\ c \downarrow t$
⟨*proof*⟩

lemma *MGT-lemma*: $C \vdash_t MGT_t\ CALL \Longrightarrow C \vdash_t MGT_t\ c$
⟨*proof*⟩

inductive-set
exec1 :: $((com\ list \times state) \times (com\ list \times state))set$
and *exec1'* :: $(com\ list \times state) \Rightarrow (com\ list \times state) \Rightarrow bool$ (- → - [*81,81*] *100*)
where
cs0 → *cs1* ≡ (*cs0*,*cs1*) : *exec1*

| *Do*[*iff*]: $t \in f\ s \Longrightarrow ((Do\ f)\#cs,s) \rightarrow (cs,t)$

| *Semi*[*iff*]: $((c1;c2)\#cs,s) \rightarrow (c1\#c2\#cs,s)$

| *IfTrue*: $b\ s \Longrightarrow ((IF\ b\ THEN\ c1\ ELSE\ c2)\#cs,s) \rightarrow (c1\#cs,s)$
| *IfFalse*: $\neg b\ s \Longrightarrow ((IF\ b\ THEN\ c1\ ELSE\ c2)\#cs,s) \rightarrow (c2\#cs,s)$

| *WhileFalse*: $\neg b\ s \Longrightarrow ((WHILE\ b\ DO\ c)\#cs,s) \rightarrow (cs,s)$
| *WhileTrue*: $b\ s \Longrightarrow ((WHILE\ b\ DO\ c)\#cs,s) \rightarrow (c\#(WHILE\ b\ DO\ c)\#cs,s)$

| *Call*[*iff*]: $(CALL\#cs,s) \rightarrow (body\#cs,s)$

| *Local*[*iff*]: $((LOCAL\ f;c;g)\#cs,s) \to (c\ \#\ Do(\lambda t.\ \{g\ s\ t\})\#cs,\ f\ s)$

abbreviation
  exectr :: (*com list* × *state*) ⇒ (*com list* × *state*) ⇒ *bool* (- →* - [*81*,*81*] *100*)
  where *cs0* →* *cs1* ≡ (*cs0*,*cs1*) : *exec1*^*

inductive-cases *exec1E*[*elim*!]:
$([],s) \to (cs',s')$
$(Do\ f\#cs,s) \to (cs',s')$
$((c1;c2)\#cs,s) \to (cs',s')$
$((IF\ b\ THEN\ c1\ ELSE\ c2)\#cs,s) \to (cs',s')$
$((WHILE\ b\ DO\ c)\#cs,s) \to (cs',s')$
$(CALL\#cs,s) \to (cs',s')$
$((LOCAL\ f;c;g)\#cs,s) \to (cs',s')$

lemma [*iff*]: ¬ $([],s) \to u$
⟨*proof*⟩

lemma *app-exec*: $(cs,s) \to (cs',s') \Longrightarrow (cs@cs2,s) \to (cs'@cs2,s')$
⟨*proof*⟩

lemma *app-execs*: $(cs,s) \to^* (cs',s') \Longrightarrow (cs@cs2,s) \to^* (cs'@cs2,s')$
⟨*proof*⟩

lemma *exec-impl-execs*[*rule-format*]:
$s -c\to s' \Longrightarrow \forall cs.\ (c\#cs,s) \to^* (cs,s')$
⟨*proof*⟩

inductive
  execs :: *state* ⇒ *com list* ⇒ *state* ⇒ *bool* (-/ =-⇒/ - [*50*,*0*,*50*] *50*)
where
  $s =[]\Rightarrow s$
| $s -c\to t \Longrightarrow t =cs\Rightarrow u \Longrightarrow s =c\#cs\Rightarrow u$

inductive-cases [*elim*!]:
$s =[]\Rightarrow t$
$s =c\#cs\Rightarrow t$

theorem *exec1s-impl-execs*: $(cs,s) \to^* ([],t) \Longrightarrow s =cs\Rightarrow t$
⟨*proof*⟩

theorem *exec1s-impl-exec*: $([c],s) \to^* ([],t) \Longrightarrow s -c\to t$
⟨*proof*⟩

primrec *termis* :: *com list* ⇒ *state* ⇒ *bool* (**infixl** ⇓ *60*) **where**
  $[]\Downarrow s = True$
| $c\#cs \Downarrow s = (c\downarrow s \wedge (\forall t.\ s -c\to t \longrightarrow cs\Downarrow t))$

lemma *exec1-pres-termis*: $(cs,s) \to (cs',s') \Longrightarrow cs{\Downarrow}s \longrightarrow cs'{\Downarrow}s'$
⟨*proof*⟩

lemma *execs-pres-termis*: $(cs,s) \to^* (cs',s') \Longrightarrow cs{\Downarrow}s \longrightarrow cs'{\Downarrow}s'$
⟨*proof*⟩

lemma *execs-pres-termi*: $[\![([c],s) \to^* (c'\#cs',s');\ c{\downarrow}s]\!] \Longrightarrow c'{\downarrow}s'$
⟨*proof*⟩

definition
termi-call-steps :: $(state \times state)set$ **where**
$termi\text{-}call\text{-}steps = \{(t,s).\ body{\downarrow}s \wedge (\exists\, cs.\ ([body],\ s) \to^* (CALL\ \#\ cs,\ t))\}$

lemma *lem*:
$\forall\, y.\ (a,y) \in r^+ \longrightarrow P\ a \longrightarrow P\ y \Longrightarrow ((b,a) \in \{(y,x).\ P\ x \wedge (x,y) : r\}^+) = ((b,a) \in \{(y,x).\ P\ x \wedge (x,y) \in r^+\})$
⟨*proof*⟩

lemma *renumber-aux*:
$[\![\forall\, i.\ (a,f\ i) : r\widehat{\ }* \wedge (f\ i,f(Suc\ i)) : r;\ (a,b) : r\widehat{\ }*\ ]\!] \Longrightarrow b = f\ 0 \longrightarrow (\exists f.\ f\ 0 = a\ \&\ (\forall\, i.\ (f\ i,\ f(Suc\ i)) : r))$
⟨*proof*⟩

lemma *renumber*:
$\forall\, i.\ (a,f\ i) : r\widehat{\ }* \wedge (f\ i,f(Suc\ i)) : r \Longrightarrow \exists f.\ f\ 0 = a\ \&\ (\forall\, i.\ (f\ i,\ f(Suc\ i)) : r)$
⟨*proof*⟩

definition *inf* :: *com list* ⇒ *state* ⇒ *bool* **where**
$inf\ cs\ s \longleftrightarrow (\exists f.\ f\ 0 = (cs,s) \wedge (\forall\, i.\ f\ i \to f(Suc\ i)))$

lemma [*iff*]: $\neg\ inf\ []\ s$
⟨*proof*⟩

lemma [*iff*]: $\neg\ inf\ [Do\ f]\ s$
⟨*proof*⟩

lemma [*iff*]: $inf\ ((c1;c2)\#cs)\ s = inf\ (c1\#c2\#cs)\ s$
⟨*proof*⟩

lemma [*iff*]: $inf\ ((IF\ b\ THEN\ c1\ ELSE\ c2)\#cs)\ s =$
$inf\ ((if\ b\ s\ then\ c1\ else\ c2)\#cs)\ s$
⟨*proof*⟩

lemma [*simp*]:
$inf\ ((WHILE\ b\ DO\ c)\#cs)\ s =$
$(if\ b\ s\ then\ inf\ (c\#(WHILE\ b\ DO\ c)\#cs)\ s\ else\ inf\ cs\ s)$
⟨*proof*⟩

lemma [*iff*]: $inf\ (CALL\#cs)\ s = inf\ (body\#cs)\ s$
⟨*proof*⟩

lemma [*iff*]: $inf\ ((LOCAL\ f;c;g)\#cs)\ s =$
$inf\ (c\#Do(\lambda t.\ \{g\ s\ t\})\#cs)\ (f\ s)$
⟨*proof*⟩

lemma *exec1-only1-aux*: $(ccs,s) \to (cs',t) \Longrightarrow$
$\forall\, c\ cs.\ ccs = c\#cs \longrightarrow (\exists\, cs1.\ cs' = cs1\ @\ cs)$
⟨*proof*⟩

lemma *exec1-only1*: $(c\#cs,s) \to (cs',t) \Longrightarrow \exists\, cs1.\ cs' = cs1\ @\ cs$
⟨*proof*⟩

lemma *exec1-drop-suffix-aux*:
$(cs12,s) \to (cs1'2,s') \Longrightarrow \forall\, cs1\ cs2\ cs1'.$
$cs12 = cs1@cs2\ \&\ cs1'2 = cs1'@cs2\ \&\ cs1 \neq [] \longrightarrow (cs1,s) \to (cs1',s')$
⟨*proof*⟩

lemma *exec1-drop-suffix*:
$(cs1@cs2,s) \to (cs1'@cs2,s') \Longrightarrow cs1 \neq [] \Longrightarrow (cs1,s) \to (cs1',s')$
⟨*proof*⟩

lemma *execs-drop-suffix*[*rule-format*(*no-asm*)]:
$[\![\ f\ 0 = (c\#cs,s); \forall\, i.\ f(i) \to f(Suc\ i)\ ]\!] \Longrightarrow$
$(\forall\, i{<}k.\ p\ i \neq []\ \&\ fst(f\ i) = p\ i@cs) \longrightarrow fst(f\ k) = p\ k@cs$
$\longrightarrow ([c],s) \to^* (p\ k,snd(f\ k))$
⟨*proof*⟩

lemma *execs-drop-suffix0*:
$[\![\ f\ 0 = (c\#cs,s); \forall\, i.\ f(i) \to f(Suc\ i);\ \forall\, i{<}k.\ p\ i \neq []\ \&\ fst(f\ i) = p\ i@cs;$
$fst(f\ k) = cs;\ p\ k = []\ ]\!] \Longrightarrow ([c],s) \to^* ([],snd(f\ k))$
⟨*proof*⟩

lemma *skolemize1*: $\forall\, x.\ P\ x \longrightarrow (\exists\, y.\ Q\ x\ y) \Longrightarrow \exists f.\forall\, x.\ P\ x \longrightarrow Q\ x\ (f\ x)$
⟨*proof*⟩

lemma *least-aux*: $[\![f\ 0 = (c\ \#\ cs,\ s);\ \forall\, i.\ f\ i \to f\ (Suc\ i);$
$fst(f\ k) = cs;\ \forall\, i{<}k.\ fst(f\ i) \neq cs]\!]$
$\Longrightarrow \forall\, i \leq k.\ (\exists\, p.\ (p \neq []) = (i < k)\ \&\ fst(f\ i) = p\ @\ cs)$
⟨*proof*⟩

lemma *least-lem*: $[\![f\ 0 = (c\#cs,s);\ \forall\, i.\ f\ i \to f(Suc\ i);\ \exists\, i.\ fst(f\ i) = cs\ ]\!]$
$\Longrightarrow \exists\, k.\ fst(f\ k) = cs\ \&\ ([c],s) \to^* ([],snd(f\ k))$
⟨*proof*⟩

lemma *skolemize2*: $\forall\, x.\exists\, y.\ P\ x\ y \Longrightarrow \exists f.\forall\, x.\ P\ x\ (f\ x)$
⟨*proof*⟩

lemma *inf-cases*: $inf\ (c\#cs)\ s \Longrightarrow inf\ [c]\ s \vee (\exists\, t.\ s -c\rightarrow t \wedge inf\ cs\ t)$
⟨*proof*⟩

lemma *termi-impl-not-inf*: $c \downarrow s \Longrightarrow \neg\ inf\ [c]\ s$
⟨*proof*⟩

lemma *termi-impl-no-inf-chain*:
$c{\downarrow}s \Longrightarrow \neg(\exists f.\ f\ 0 = ([c],s) \wedge (\forall\, i{::}nat.\ (f\ i,\ f(i{+}1)) : exec1^{\frown}{+}))$
⟨*proof*⟩

primrec *cseq* :: $(nat \Rightarrow state) \Rightarrow nat \Rightarrow com\ list$ **where**
  $cseq\ S\ 0 = []$
| $cseq\ S\ (Suc\ i) = (SOME\ cs.\ ([body],\ S\ i) \rightarrow^*\ (CALL\ \#\ cs,\ S(i{+}1))) @ cseq\ S\ i$

lemma *wf-termi-call-steps*: *wf termi-call-steps*
⟨*proof*⟩

lemma *CALL-lemma*:
$\{(\lambda z\ s.\ (z{=}s \wedge body{\downarrow}s) \wedge (s,t) \in termi\text{-}call\text{-}steps,\ CALL,\ \lambda z\ s.\ z -body\rightarrow s)\} \vdash_t$
$\{\lambda z\ s.\ (z{=}s \wedge body{\downarrow}t) \wedge (\exists\, cs.\ ([body],t) \rightarrow^*\ (c\#cs,s))\}\ c\ \{\lambda z\ s.\ z -c\rightarrow s\}$
⟨*proof*⟩

lemma *CALL-cor*:
$\{(\lambda z\ s.\ (z{=}s \wedge body{\downarrow}s) \wedge (s,t) \in termi\text{-}call\text{-}steps,\ CALL,\ \lambda z\ s.\ z -body\rightarrow s)\} \vdash_t$
$\{\lambda z\ s.\ (z{=}s \wedge body{\downarrow}s) \wedge s = t\}\ body\ \{\lambda z\ s.\ z -body\rightarrow s\}$
⟨*proof*⟩

lemma *MGT-CALL*: $\{\} \vdash_t MGT_t\ CALL$
⟨*proof*⟩

theorem $\{\} \models_t \{P\}c\{Q\} \implies \{\} \vdash_t \{P\}c\{Q{::}state\ assn\}$
⟨*proof*⟩

end

4 Hoare Logics for Mutually Recursive Procedure

theory *PsLang* **imports** *Main* **begin**

4.1 The language

typedecl *state*
typedecl *pname*

type-synonym *bexp* = *state* ⇒ *bool*

datatype

```
com = Do state ⇒ state set
    | Semi  com com          (-; -  [60, 60] 10)
    | Cond  bexp com com     (IF - THEN - ELSE -  60)
    | While bexp com         (WHILE - DO -  60)
    | CALL pname
    | Local (state ⇒ state) com (state ⇒ state ⇒ state)
          (LOCAL -; -; - [0,0,60] 60)
```

consts *body* :: *pname* ⇒ *com*

We generalize from a single procedure to a whole set of procedures following the ideas of von Oheimb [3]. The basic setup is modified only in a few places:

- We introduce a new basic type *pname* of procedure names.
- Constant *body* is now of type *pname* ⇒ *com*.
- The *CALL* command now has an argument of type *pname*, the name of the procedure that is to be called.

```
inductive
  exec :: state ⇒ com ⇒ state ⇒ bool   (-/ -→/ - [50,0,50] 50)
where
    Do:     t ∈ f s ⟹ s -Do f→ t

  | Semi:   ⟦ s0 -c1→ s1; s1 -c2→ s2 ⟧ ⟹ s0 -c1;c2→ s2

  | IfTrue:  ⟦ b s;  s -c1→ t ⟧ ⟹ s -IF b THEN c1 ELSE c2→ t
  | IfFalse: ⟦ ¬b s; s -c2→ t ⟧ ⟹ s -IF b THEN c1 ELSE c2→ t

  | WhileFalse: ¬b s ⟹ s -WHILE b DO c→ s
  | WhileTrue:  ⟦ b s; s -c→ t; t -WHILE b DO c→ u ⟧
              ⟹ s -WHILE b DO c→ u

  | Call: s -body p→ t ⟹ s -CALL p→ t

  | Local: f s -c→ t ⟹ s -LOCAL f; c; g→ g s t
```

lemma [*iff*]: (*s* -*Do f*→ *t*) = (*t* ∈ *f s*)
⟨*proof*⟩

lemma [*iff*]: (*s* -*c*;*d*→ *u*) = (∃ *t*. *s* -*c*→ *t* ∧ *t* -*d*→ *u*)
⟨*proof*⟩

lemma [*iff*]: (*s* -*IF b THEN c ELSE d*→ *t*) =
(*s* -*if b s then c else d*→ *t*)
⟨*proof*⟩

lemma [*iff*]: (*s* -*CALL p*→ *t*) = (*s* -*body p*→ *t*)

⟨*proof*⟩

lemma [*iff*]: $(s\ -LOCAL\ f;\ c;\ g\rightarrow\ u) = (\exists\, t.\ f\ s\ -c\rightarrow\ t \wedge u = g\ s\ t)$
⟨*proof*⟩

inductive
  execn :: *state* ⇒ *com* ⇒ *nat* ⇒ *state* ⇒ *bool* (-/ −-*-→/ - [50,0,0,50] 50)
where
  Do: $t \in f\ s \Longrightarrow s\ -Do\ f-n\rightarrow\ t$

| *Semi*: $[\![\ s0\ -c0-n\rightarrow\ s1;\ s1\ -c1-n\rightarrow\ s2\ ]\!] \Longrightarrow s0\ -c0;c1-n\rightarrow\ s2$

| *IfTrue*: $[\![\ b\ s;\ s\ -c0-n\rightarrow\ t\ ]\!] \Longrightarrow s\ -IF\ b\ THEN\ c0\ ELSE\ c1-n\rightarrow\ t$

| *IfFalse*: $[\![\ \neg b\ s;\ s\ -c1-n\rightarrow\ t\ ]\!] \Longrightarrow s\ -IF\ b\ THEN\ c0\ ELSE\ c1-n\rightarrow\ t$

| *WhileFalse*: $\neg b\ s \Longrightarrow s\ -WHILE\ b\ DO\ c-n\rightarrow\ s$

| *WhileTrue*: $[\![\ b\ s;\ s\ -c-n\rightarrow\ t;\ t\ -WHILE\ b\ DO\ c-n\rightarrow\ u\ ]\!]$
  $\Longrightarrow s\ -WHILE\ b\ DO\ c-n\rightarrow\ u$

| *Call*: $s\ -body\ p-n\rightarrow\ t \Longrightarrow s\ -CALL\ p-Suc\ n\rightarrow\ t$

| *Local*: $f\ s\ -c-n\rightarrow\ t \Longrightarrow s\ -LOCAL\ f;\ c;\ g-n\rightarrow\ g\ s\ t$

lemma [*iff*]: $(s\ -Do\ f-n\rightarrow\ t) = (t \in f\ s)$
⟨*proof*⟩

lemma [*iff*]: $(s\ -c1;c2-n\rightarrow\ u) = (\exists\, t.\ s\ -c1-n\rightarrow\ t \wedge t\ -c2-n\rightarrow\ u)$
⟨*proof*⟩

lemma [*iff*]: $(s\ -IF\ b\ THEN\ c\ ELSE\ d-n\rightarrow\ t) =$
  $(s\ -if\ b\ s\ then\ c\ else\ d-n\rightarrow\ t)$
⟨*proof*⟩

lemma [*iff*]: $(s\ -CALL\ p-\ 0\rightarrow\ t) = False$
⟨*proof*⟩

lemma [*iff*]: $(s\ -CALL\ p-Suc\ n\rightarrow\ t) = (s\ -body\ p-n\rightarrow\ t)$
⟨*proof*⟩

lemma [*iff*]: $(s\ -LOCAL\ f;\ c;\ g-n\rightarrow\ u) = (\exists\, t.\ f\ s\ -c-n\rightarrow\ t \wedge u = g\ s\ t)$
⟨*proof*⟩

lemma *exec-mono*[*rule-format*]: $s\ -c-m\rightarrow\ t \Longrightarrow \forall\, n.\ m \leq n \longrightarrow s\ -c-n\rightarrow\ t$
⟨*proof*⟩

lemma *exec-iff-execn*: $(s\ -c\rightarrow\ t) = (\exists\, n.\ s\ -c-n\rightarrow\ t)$

⟨*proof*⟩

lemma *while-lemma*[*rule-format*]:
$s\ -w-n\rightarrow\ t \Longrightarrow \forall\, b\ c.\ w = \mathit{WHILE}\ b\ \mathit{DO}\ c \wedge P\ s \wedge$
$(\forall\, s\ s'.\ P\ s \wedge b\ s \wedge s\ -c-n\rightarrow\ s' \longrightarrow P\ s') \longrightarrow P\ t \wedge \neg b\ t$
⟨*proof*⟩

lemma *while-rule*:
$[\![s\ -\mathit{WHILE}\ b\ \mathit{DO}\ c-n\rightarrow\ t;\ P\ s;\ \bigwedge s\ s'.\ [\![P\ s;\ b\ s;\ s\ -c-n\rightarrow\ s']\!] \Longrightarrow P\ s']\!]$
$\Longrightarrow P\ t \wedge \neg b\ t$
⟨*proof*⟩

end

theory *PsHoare* **imports** *PsLang* **begin**

4.2 Hoare logic for partial correctness

type-synonym *'a assn* = *'a* ⇒ *state* ⇒ *bool*
type-synonym *'a cntxt* = (*'a assn* × *com* × *'a assn*)*set*

definition
valid :: *'a assn* ⇒ *com* ⇒ *'a assn* ⇒ *bool* (⊨ {(1-)}/ (-)/ {(1-)} 50) **where**
⊨ {*P*}*c*{*Q*} ≡ (∀ *s t z*. *s* −*c*→ *t* ⟶ *P z s* ⟶ *Q z t*)

definition
valids :: *'a cntxt* ⇒ *bool* (|⊨ - 50) **where**
|⊨ *D* ≡ (∀ (*P*,*c*,*Q*) ∈ *D*. ⊨ {*P*}*c*{*Q*})

definition
nvalid :: *nat* ⇒ *'a assn* ⇒ *com* ⇒ *'a assn* ⇒ *bool* (⊨- {(1-)}/ (-)/ {(1-)} 50)
where
⊨*n* {*P*}*c*{*Q*} ≡ (∀ *s t z*. *s* −*c*−*n*→ *t* ⟶ *P z s* ⟶ *Q z t*)

definition
nvalids :: *nat* ⇒ *'a cntxt* ⇒ *bool* (|⊨'--/ - 50) **where**
|⊨-*n* *C* ≡ (∀ (*P*,*c*,*Q*) ∈ *C*. ⊨*n* {*P*}*c*{*Q*})

We now need an additional notion of validity $C \mid\!\models D$ where D is a set as well. The reason is that we can now have mutually recursive procedures whose correctness needs to be established by simultaneous induction. Instead of sets of Hoare triples we may think of conjunctions. We define both $C \mid\!\models D$ and its relativized version:

definition
cvalids :: *'a cntxt* ⇒ *'a cntxt* ⇒ *bool* (- |⊨/ - 50) **where**
C |⊨ *D* ⟷ |⊨ *C* ⟶ |⊨ *D*

definition

cnvalids :: *'a cntxt* ⇒ *nat* ⇒ *'a cntxt* ⇒ *bool* (- |⊨*'*--/ - *50*) **where**
C |⊨-*n D* ⟷ |⊨-*n C* ⟶ |⊨-*n D*

Our Hoare logic now defines judgements of the form $C \Vdash D$ where both C and D are (potentially infinite) sets of Hoare triples; $C \vdash \{P\}c\{Q\}$ is simply an abbreviation for $C \Vdash \{(P,c,Q)\}$.

inductive
hoare :: *'a cntxt* ⇒ *'a cntxt* ⇒ *bool* (- ⊩/ - *50*)
and *hoare3* :: *'a cntxt* ⇒ *'a assn* ⇒ *com* ⇒ *'a assn* ⇒ *bool* (- ⊢/ ({*(1-)*}/ (-)/ {*(1-)*}) *50*)
where
C ⊢ {*P*}*c*{*Q*} ≡ *C* ⊩ {(*P*,*c*,*Q*)}
| *Do*: *C* ⊢ {λ*z s*. ∀ *t* ∈ *f s* . *P z t*} *Do f* {*P*}
| *Semi*: ⟦ *C* ⊢ {*P*}*c*{*Q*}; *C* ⊢ {*Q*}*d*{*R*} ⟧ ⟹ *C* ⊢ {*P*} *c*;*d* {*R*}
| *If*: ⟦ *C* ⊢ {λ*z s*. *P z s* ∧ *b s*}*c*{*Q*}; *C* ⊢ {λ*z s*. *P z s* ∧ ¬*b s*}*d*{*Q*} ⟧ ⟹
C ⊢ {*P*} *IF b THEN c ELSE d* {*Q*}
| *While*: *C* ⊢ {λ*z s*. *P z s* ∧ *b s*} *c* {*P*} ⟹
C ⊢ {*P*} *WHILE b DO c* {λ*z s*. *P z s* ∧ ¬*b s*}

| *Conseq*: ⟦ *C* ⊢ {*P'*}*c*{*Q'*};
∀ *s t*. (∀ *z*. *P' z s* ⟶ *Q' z t*) ⟶ (∀ *z*. *P z s* ⟶ *Q z t*) ⟧ ⟹
C ⊢ {*P*}*c*{*Q*}

| *Call*: ⟦ ∀(*P*,*c*,*Q*) ∈ *C*. ∃ *p*. *c* = *CALL p*;
C ⊩ {(*P*,*b*,*Q*). ∃ *p*. (*P*,*CALL p*,*Q*) ∈ *C* ∧ *b* = *body p*} ⟧
⟹ {} ⊩ *C*

| *Asm*: (*P*,*CALL p*,*Q*) ∈ *C* ⟹ *C* ⊢ {*P*} *CALL p* {*Q*}

| *ConjI*: ∀(*P*,*c*,*Q*) ∈ *D*. *C* ⊢ {*P*}*c*{*Q*} ⟹ *C* ⊩ *D*
| *ConjE*: ⟦ *C* ⊩ *D*; (*P*,*c*,*Q*) ∈ *D* ⟧ ⟹ *C* ⊢ {*P*}*c*{*Q*}

| *Local*: ⟦ ∀ *s'*. *C* ⊢ {λ*z s*. *P z s'* ∧ *s* = *f s'*} *c* {λ*z t*. *Q z* (*g s' t*)} ⟧ ⟹
C ⊢ {*P*} *LOCAL f*;*c*;*g* {*Q*}
monos *split-beta*

lemmas *valid-defs* = *valid-def valids-def cvalids-def*
nvalid-def nvalids-def cnvalids-def

theorem *C* ⊩ *D* ⟹ *C* |⊨ *D*

As before, we prove a generalization of $C \models D$, namely $\forall n.\ C \models_n D$, by induction on $C \Vdash D$, with an induction on n in the *CALL* case.

⟨*proof*⟩

definition *MGT* :: *com* ⇒ *state assn* × *com* × *state assn* **where**
[*simp*]: *MGT c* = (λ*z s*. *z* = *s*, *c*, λ*z t*. *z* −*c*→ *t*)

lemma *strengthen-pre*:
$[\![\forall z\ s.\ P'\ z\ s \longrightarrow P\ z\ s;\ C\vdash \{P\}c\{Q\}\ ]\!] \Longrightarrow C\vdash \{P'\}c\{Q\}$
⟨*proof*⟩

lemma *MGT-implies-complete*:
$\{\} \Vdash \{MGT\ c\} \Longrightarrow \models \{P\}c\{Q\} \Longrightarrow \{\} \vdash \{P\}c\{Q::state\ assn\}$
⟨*proof*⟩

lemma *MGT-lemma*: $\forall p.\ C \Vdash \{MGT(CALL\ p)\} \implies C \Vdash \{MGT\ c\}$
⟨*proof*⟩

lemma *MGT-body*: $(P,\ CALL\ p,\ Q) = MGT\ (CALL\ pa) \Longrightarrow C \Vdash \{MGT\ (body\ p)\} \Longrightarrow C \vdash \{P\}\ body\ p\ \{Q\}$
⟨*proof*⟩

declare *MGT-def*[*simp del*]

lemma *MGT-CALL*: $\{\} \Vdash \{mgt.\ \exists p.\ mgt = MGT(CALL\ p)\}$
⟨*proof*⟩

theorem *Complete*: $\models \{P\}c\{Q\} \implies \{\} \vdash \{P\}c\{Q::state\ assn\}$
⟨*proof*⟩

end

theory *PsTermi* **imports** *PsLang* **begin**

4.3 Termination

inductive
termi :: *com* ⇒ *state* ⇒ *bool* (**infixl** ↓ *50*)
where
Do[*iff*]: $f\ s \neq \{\} \Longrightarrow Do\ f \downarrow s$
| *Semi*[*intro*!]: $[\![c1 \downarrow s0;\ \bigwedge s1.\ s0 -c1\rightarrow s1 \Longrightarrow c2 \downarrow s1]\!] \Longrightarrow (c1;c2) \downarrow s0$

| *IfTrue*[*intro,simp*]: $[\![b\ s;\ c1 \downarrow s]\!] \Longrightarrow IF\ b\ THEN\ c1\ ELSE\ c2 \downarrow s$
| *IfFalse*[*intro,simp*]: $[\![\neg b\ s;\ c2 \downarrow s]\!] \Longrightarrow IF\ b\ THEN\ c1\ ELSE\ c2 \downarrow s$

| *WhileFalse*: $\neg b\ s \Longrightarrow WHILE\ b\ DO\ c \downarrow s$

| *WhileTrue*: $[\![b\ s;\ c \downarrow s;\ \bigwedge t.\ s -c\rightarrow t \Longrightarrow WHILE\ b\ DO\ c \downarrow t]\!] \Longrightarrow WHILE\ b\ DO\ c \downarrow s$

| *body p* ↓ *s* ⟹ *CALL p* ↓ *s*

| *Local*: $c \downarrow f\ s \Longrightarrow LOCAL\ f;c;g \downarrow s$

lemma [*iff*]: $(Do\ f \downarrow s) = (f\ s \neq \{\})$

⟨*proof*⟩

lemma [*iff*]: $((c1;c2) \downarrow s0) = (c1 \downarrow s0 \wedge (\forall\, s1.\ s0 -c1\rightarrow s1 \longrightarrow c2 \downarrow s1))$
⟨*proof*⟩

lemma [*iff*]: $(IF\ b\ THEN\ c1\ ELSE\ c2 \downarrow s) =$
$((if\ b\ s\ then\ c1\ else\ c2) \downarrow s)$
⟨*proof*⟩

lemma [*iff*]: $(CALL\ p \downarrow s) = (body\ p \downarrow s)$
⟨*proof*⟩

lemma [*iff*]: $(LOCAL\ f;c;g \downarrow s) = (c \downarrow f\ s)$
⟨*proof*⟩

lemma *termi-while-lemma*[*rule-format*]:
$w{\downarrow}fk \Longrightarrow$
$(\forall\, k\ b\ c.\ fk = f\ k \wedge w = WHILE\ b\ DO\ c \wedge (\forall\, i.\ f\ i -c\rightarrow f(Suc\ i))$
$\longrightarrow (\exists\, i.\ \neg b(f\ i)))$
⟨*proof*⟩

lemma *termi-while*:
$[\![\ (WHILE\ b\ DO\ c) \downarrow f\ k;\ \ \forall\, i.\ f\ i -c\rightarrow f(Suc\ i)\ ]\!] \Longrightarrow \exists\, i.\ \neg b(f\ i)$
⟨*proof*⟩

lemma *wf-termi*: $wf\ \{(t,s).\ WHILE\ b\ DO\ c \downarrow s \wedge b\ s \wedge s -c\rightarrow t\}$
⟨*proof*⟩

end

theory *PsHoareTotal* **imports** *PsHoare PsTermi* **begin**

4.4 Hoare logic for total correctness

definition
$tvalid :: 'a\ assn \Rightarrow com \Rightarrow 'a\ assn \Rightarrow bool\ (\models_t\ \{(1\text{-})\}/\ (\text{-})/\ \{(1\text{-})\}\ 50)$ **where**
$\models_t\ \{P\}c\{Q\} \longleftrightarrow \models\ \{P\}c\{Q\} \wedge (\forall\, z\ s.\ P\ z\ s \longrightarrow c{\downarrow}s)$

definition
$valids :: 'a\ cntxt \Rightarrow bool\ (|\models_t\ \text{-}\ 50)$ **where**
$|\models_t\ D \longleftrightarrow (\forall\, (P,c,Q) \in D.\ \models_t\ \{P\}c\{Q\})$

definition
$ctvalid :: 'a\ cntxt \Rightarrow 'a\ assn \Rightarrow com \Rightarrow 'a\ assn \Rightarrow bool$
$((\text{-}\ /\models_t\ \{(1\text{-})\}/\ (\text{-})/\ \{(1\text{-})\})\ 50)$ **where**
$C \models_t\ \{P\}c\{Q\} \longleftrightarrow |\models_t\ C \longrightarrow\ \models_t\ \{P\}c\{Q\}$

definition

cvalids :: *'a cntxt* ⇒ *'a cntxt* ⇒ *bool* (- |⊨$_t$/ - *50*) **where**
C |⊨$_t$ *D* ⟷ |⊨$_t$ *C* ⟶ |⊨$_t$ *D*

inductive
thoare :: *'a cntxt* ⇒ *'a cntxt* ⇒ *bool* ((- |⊢$_t$/ -) *50*)
and *thoare'* :: *'a cntxt* ⇒ *'a assn* ⇒ *com* ⇒ *'a assn* ⇒ *bool*
((- ⊢$_t$/ ({(*1*-)}/ (-)/ {(*1*-)})) [*50*,*0*,*0*,*0*] *50*)
where
C ⊢$_t$ {*P*}*c*{*Q*} ≡ *C* |⊢$_t$ {(*P*,*c*,*Q*)}
| *Do*: *C* ⊢$_t$ {λ*z s*. (∀ *t* ∈ *f s* . *P z t*) ∧ *f s* ≠ {}} *Do f* {*P*}
| *Semi*: ⟦ *C* ⊢$_t$ {*P*}*c1*{*Q*}; *C* ⊢$_t$ {*Q*}*c2*{*R*} ⟧ ⟹ *C* ⊢$_t$ {*P*} *c1*;*c2* {*R*}
| *If*: ⟦ *C* ⊢$_t$ {λ*z s*. *P z s* ∧ *b s*}*c*{*Q*}; *C* ⊢$_t$ {λ*z s*. *P z s* ∧ ~*b s*}*d*{*Q*} ⟧ ⟹
C ⊢$_t$ {*P*} *IF b THEN c ELSE d* {*Q*}
| *While*:
⟦*wf r*; ∀ *s'*. *C* ⊢$_t$ {λ*z s*. *P z s* ∧ *b s* ∧ *s'* = *s*} *c* {λ*z s*. *P z s* ∧ (*s*,*s'*) ∈ *r*}⟧
⟹ *C* ⊢$_t$ {*P*} *WHILE b DO c* {λ*z s*. *P z s* ∧ ¬*b s*}

| *Call*:
⟦ *wf r*;
∀ *q pre*.
(⋃*p*. {(λ*z s*. *P p z s* ∧ ((*p*,*s*),(*q*,*pre*)) ∈ *r*,*CALL p*,*Q p*)}
⊢$_t$ {λ*z s*. *P q z s* ∧ *s* = *pre*} *body q* {*Q q*} ⟧
⟹ {} |⊢$_t$ ⋃*p*. {(*P p*, *CALL p*, *Q p*)}

| *Asm*: (*P*,*CALL p*,*Q*) ∈ *C* ⟹ *C* ⊢$_t$ {*P*} *CALL p* {*Q*}

| *Conseq*:
⟦ *C* ⊢$_t$ {*P'*}*c*{*Q'*};
(∀ *s t*. (∀ *z*. *P' z s* ⟶ *Q' z t*) ⟶ (∀ *z*. *P z s* ⟶ *Q z t*)) ∧
(∀ *s*. (∃ *z*. *P z s*) ⟶ (∃ *z*. *P' z s*)) ⟧
⟹ *C* ⊢$_t$ {*P*}*c*{*Q*}

| *ConjI*: ∀ (*P*,*c*,*Q*) ∈ *D*. *C* ⊢$_t$ {*P*}*c*{*Q*} ⟹ *C* |⊢$_t$ *D*
| *ConjE*: ⟦ *C* |⊢$_t$ *D*; (*P*,*c*,*Q*) ∈ *D* ⟧ ⟹ *C* ⊢$_t$ {*P*}*c*{*Q*}

| *Local*: ⟦ ∀ *s'*. *C* ⊢$_t$ {λ*z s*. *P z s'* ∧ *s* = *f s'*} *c* {λ*z t*. *Q z* (*g s' t*)} ⟧ ⟹
C ⊢$_t$ {*P*} *LOCAL f*;*c*;*g* {*Q*}
monos *split-beta*

lemma *strengthen-pre*:
⟦ ∀ *z s*. *P' z s* ⟶ *P z s*; *C* ⊢$_t$ {*P*}*c*{*Q*} ⟧ ⟹ *C* ⊢$_t$ {*P'*}*c*{*Q*}
⟨*proof*⟩

lemma *weaken-post*:
⟦ *C* ⊢$_t$ {*P*}*c*{*Q*}; ∀ *z s*. *Q z s* ⟶ *Q' z s* ⟧ ⟹ *C* ⊢$_t$ {*P*}*c*{*Q'*}
⟨*proof*⟩

lemmas *tvalid-defs* = *tvalid-def ctvalid-def valids-def cvalids-def valid-defs*

lemma [*iff*]:
$(\models_t \{\lambda z\ s.\ \exists\, n.\ P\ n\ z\ s\} c \{Q\}) = (\forall\, n.\ \models_t \{P\ n\} c \{Q\})$
⟨*proof*⟩

lemma [*iff*]:
$(\models_t \{\lambda z\ s.\ P\ z\ s \wedge P'\} c \{Q\}) = (P' \longrightarrow \models_t \{P\} c \{Q\})$
⟨*proof*⟩

lemma [*iff*]: $(\models_t \{P\} CALL\ p \{Q\}) = (\models_t \{P\} body\ p \{Q\})$
⟨*proof*⟩

lemma *unfold-while*:
$(s - WHILE\ b\ DO\ c \rightarrow u) =$
$(s - IF\ b\ THEN\ c; WHILE\ b\ DO\ c\ ELSE\ Do(\lambda s.\ \{s\}) \rightarrow u)$
⟨*proof*⟩

theorem $C \mid\vdash_t D \implies C \mid\models_t D$
⟨*proof*⟩

definition $MGT_t :: com \Rightarrow state\ assn \times com \times state\ assn$ **where**
[*simp*]: $MGT_t\ c = (\lambda z\ s.\ z = s \wedge c{\downarrow}s,\ c,\ \lambda z\ t.\ z - c \rightarrow t)$

lemma *MGT-implies-complete*:
$\{\} \mid\vdash_t \{MGT_t\ c\} \Longrightarrow \{\} \models_t \{P\} c \{Q\} \Longrightarrow \{\} \vdash_t \{P\} c \{Q :: state\ assn\}$
⟨*proof*⟩

lemma *while-termiE*: ⟦ $WHILE\ b\ DO\ c \downarrow s;\ b\ s$ ⟧ ⟹ $c \downarrow s$
⟨*proof*⟩

lemma *while-termiE2*: ⟦ $WHILE\ b\ DO\ c \downarrow s;\ b\ s;\ s - c \rightarrow t$ ⟧ ⟹ $WHILE\ b\ DO\ c \downarrow t$
⟨*proof*⟩

lemma *MGT-lemma*: $\forall\, p.\ \{\} \mid\vdash_t \{MGT_t(CALL\ p)\} \implies \{\} \mid\vdash_t \{MGT_t\ c\}$
⟨*proof*⟩

inductive-set
exec1 :: $((com\ list \times state) \times (com\ list \times state))set$
and *exec1'* :: $(com\ list \times state) \Rightarrow (com\ list \times state) \Rightarrow bool$ (- → - [81,81] 100)
where
cs0 → *cs1* ≡ (*cs0*,*cs1*) : *exec1*

| *Do*[*iff*]: $t \in f\ s \Longrightarrow ((Do\ f)\#cs,s) \rightarrow (cs,t)$

| *Semi*[*iff*]: $((c1;c2)\#cs,s) \rightarrow (c1\#c2\#cs,s)$

| *IfTrue*: $b\ s \Longrightarrow ((IF\ b\ THEN\ c1\ ELSE\ c2)\#cs,s) \rightarrow (c1\#cs,s)$

| *IfFalse*: $\neg b\ s \Longrightarrow ((IF\ b\ THEN\ c1\ ELSE\ c2)\#cs,s) \rightarrow (c2\#cs,s)$

| *WhileFalse*: $\neg b\ s \Longrightarrow ((WHILE\ b\ DO\ c)\#cs,s) \rightarrow (cs,s)$
| *WhileTrue*: $b\ s \Longrightarrow ((WHILE\ b\ DO\ c)\#cs,s) \rightarrow (c\#(WHILE\ b\ DO\ c)\#cs,s)$

| *Call*[*iff*]: $(CALL\ p\#cs,s) \rightarrow (body\ p\#cs,s)$

| *Local*[*iff*]: $((LOCAL\ f;c;g)\#cs,s) \rightarrow (c\ \#\ Do(\lambda t.\ \{g\ s\ t\})\#cs,\ f\ s)$

abbreviation
exectr :: $(com\ list \times state) \Rightarrow (com\ list \times state) \Rightarrow bool$ $(- \rightarrow^* - [81,81]\ 100)$
where $cs0 \rightarrow^*\ cs1 \equiv (cs0,cs1) : exec1\hat{}*$

inductive-cases *exec1E*[*elim*!]:
$([],s) \rightarrow (cs',s')$
$(Do\ f\#cs,s) \rightarrow (cs',s')$
$((c1;c2)\#cs,s) \rightarrow (cs',s')$
$((IF\ b\ THEN\ c1\ ELSE\ c2)\#cs,s) \rightarrow (cs',s')$
$((WHILE\ b\ DO\ c)\#cs,s) \rightarrow (cs',s')$
$(CALL\ p\#cs,s) \rightarrow (cs',s')$
$((LOCAL\ f;c;g)\#cs,s) \rightarrow (cs',s')$

lemma [*iff*]: $\neg\ ([],s) \rightarrow u$
⟨*proof*⟩

lemma *app-exec*: $(cs,s) \rightarrow (cs',s') \Longrightarrow (cs@cs2,s) \rightarrow (cs'@cs2,s')$
⟨*proof*⟩

lemma *app-execs*: $(cs,s) \rightarrow^*\ (cs',s') \Longrightarrow (cs@cs2,s) \rightarrow^*\ (cs'@cs2,s')$
⟨*proof*⟩

lemma *exec-impl-execs*[*rule-format*]:
$s\ -c\rightarrow\ s' \Longrightarrow \forall\ cs.\ (c\#cs,s) \rightarrow^*\ (cs,s')$
⟨*proof*⟩

inductive
execs :: $state \Rightarrow com\ list \Rightarrow state \Rightarrow bool$ $(-/\ =-\Rightarrow/\ - [50,0,50]\ 50)$
where
$s\ =[]\Rightarrow\ s$
| $s\ -c\rightarrow\ t \Longrightarrow t\ =cs\Rightarrow\ u \Longrightarrow s\ =c\#cs\Rightarrow\ u$

inductive-cases [*elim*!]:
$s\ =[]\Rightarrow\ t$
$s\ =c\#cs\Rightarrow\ t$

theorem *exec1s-impl-execs*: $(cs,s) \rightarrow^*\ ([],t) \Longrightarrow s\ =cs\Rightarrow\ t$
⟨*proof*⟩

theorem *exec1s-impl-exec*: $([c],s) \rightarrow^* ([],t) \Longrightarrow s -c\rightarrow t$
⟨*proof*⟩

primrec *termis* :: *com list* ⇒ *state* ⇒ *bool* (**infixl** ⇓ *60*) **where**
  $[]\Downarrow s = True$
| $c\#cs \Downarrow s = (c\downarrow s \wedge (\forall t.\ s -c\rightarrow t \longrightarrow cs\Downarrow t))$

lemma *exec1-pres-termis*: $(cs,s) \rightarrow (cs',s') \Longrightarrow cs\Downarrow s \longrightarrow cs'\Downarrow s'$
⟨*proof*⟩

lemma *execs-pres-termis*: $(cs,s) \rightarrow^* (cs',s') \Longrightarrow cs\Downarrow s \longrightarrow cs'\Downarrow s'$
⟨*proof*⟩

lemma *execs-pres-termi*: $[\![([c],s) \rightarrow^* (c'\#cs',s');\ c\downarrow s]\!] \Longrightarrow c'\downarrow s'$
⟨*proof*⟩

definition *termi-call-steps* :: $((pname \times state) \times (pname \times state))set$ **where**
  termi-call-steps =
    $\{((q,t),(p,s)).\ body\ p\downarrow s \wedge (\exists cs.\ ([body\ p],\ s) \rightarrow^* (CALL\ q\#\ cs,\ t))\}$

lemma *lem*:
  $\forall y.\ (a,y)\in r^+ \longrightarrow P\ a \longrightarrow P\ y \Longrightarrow ((b,a) \in \{(y,x).\ P\ x \wedge (x,y)\in r\}^+) = ((b,a) \in \{(y,x).\ P\ x \wedge (x,y)\in r^+\})$
⟨*proof*⟩

lemma *renumber-aux*:
 $[\![\forall i.\ (a,f\ i) \in r\widehat{}* \wedge (f\ i,f(Suc\ i)) : r;\ (a,b) : r\widehat{}*]\!] \Longrightarrow b = f\ 0 \longrightarrow (\exists f.\ f\ 0 = a\ \&\ (\forall i.\ (f\ i,\ f(Suc\ i)) : r))$
⟨*proof*⟩

lemma *renumber*:
 $\forall i.\ (a,f\ i) : r\widehat{}* \wedge (f\ i,f(Suc\ i)) : r \Longrightarrow \exists f.\ f\ 0 = a\ \&\ (\forall i.\ (f\ i,\ f(Suc\ i)) : r)$
⟨*proof*⟩

definition *inf* :: *com list* ⇒ *state* ⇒ *bool* **where**
  $inf\ cs\ s \longleftrightarrow (\exists f.\ f\ 0 = (cs,s) \wedge (\forall i.\ f\ i \rightarrow f(Suc\ i)))$

lemma [*iff*]: $\neg\ inf\ []\ s$
⟨*proof*⟩

lemma [*iff*]: $\neg\ inf\ [Do\ f]\ s$
⟨*proof*⟩

lemma [*iff*]: $inf\ ((c1;c2)\#cs)\ s = inf\ (c1\#c2\#cs)\ s$
⟨*proof*⟩

lemma [*iff*]: *inf* ((*IF b THEN c1 ELSE c2*)#*cs*) *s* =
inf ((*if b s then c1 else c2*)#*cs*) *s*
⟨*proof*⟩

lemma [*simp*]:
inf ((*WHILE b DO c*)#*cs*) *s* =
(*if b s then inf* (*c*#(*WHILE b DO c*)#*cs*) *s else inf cs s*)
⟨*proof*⟩

lemma [*iff*]: *inf* (*CALL p*#*cs*) *s* = *inf* (*body p*#*cs*) *s*
⟨*proof*⟩

lemma [*iff*]: *inf* ((*LOCAL f*;*c*;*g*)#*cs*) *s* =
inf (*c*#*Do*(λ*t*. {*g s t*})#*cs*) (*f s*)
⟨*proof*⟩

lemma *exec1-only1-aux*: (*ccs*,*s*) → (*cs*′,*t*) ⟹
∀ *c cs*. *ccs* = *c*#*cs* ⟶ (∃ *cs1*. *cs*′ = *cs1* @ *cs*)
⟨*proof*⟩

lemma *exec1-only1*: (*c*#*cs*,*s*) → (*cs*′,*t*) ⟹ ∃ *cs1*. *cs*′ = *cs1* @ *cs*
⟨*proof*⟩

lemma *exec1-drop-suffix-aux*:
(*cs12*,*s*) → (*cs1*′*2*,*s*′) ⟹ ∀ *cs1 cs2 cs1*′.
cs12 = *cs1*@*cs2* & *cs1*′*2* = *cs1*′@*cs2* & *cs1* ≠ [] ⟶ (*cs1*,*s*) → (*cs1*′,*s*′)
⟨*proof*⟩

lemma *exec1-drop-suffix*:
(*cs1*@*cs2*,*s*) → (*cs1*′@*cs2*,*s*′) ⟹ *cs1* ≠ [] ⟹ (*cs1*,*s*) → (*cs1*′,*s*′)
⟨*proof*⟩

lemma *execs-drop-suffix*[*rule-format*(*no-asm*)]:
⟦ *f 0* = (*c*#*cs*,*s*);∀ *i*. *f*(*i*) → *f*(*Suc i*) ⟧ ⟹
(∀ *i*<*k*. *p i* ≠ [] & *fst*(*f i*) = *p i*@*cs*) ⟶ *fst*(*f k*) = *p k*@*cs*
⟶ ([*c*],*s*) →* (*p k*,*snd*(*f k*))
⟨*proof*⟩

lemma *execs-drop-suffix0*:
⟦ *f 0* = (*c*#*cs*,*s*);∀ *i*. *f*(*i*) → *f*(*Suc i*); ∀ *i*<*k*. *p i* ≠ [] & *fst*(*f i*) = *p i*@*cs*;
fst(*f k*) = *cs*; *p k* = [] ⟧ ⟹ ([*c*],*s*) →* ([],*snd*(*f k*))
⟨*proof*⟩

lemma *skolemize1*: ∀ *x*. *P x* ⟶ (∃ *y*. *Q x y*) ⟹ ∃ *f*.∀ *x*. *P x* ⟶ *Q x* (*f x*)
⟨*proof*⟩

lemma *least-aux*: ⟦*f 0* = (*c* # *cs*, *s*); ∀ *i*. *f i* → *f* (*Suc i*);
fst(*f k*) = *cs*; ∀ *i*<*k*. *fst*(*f i*) ≠ *cs*⟧
⟹ ∀ *i* ≤ *k*. (∃ *p*. (*p* ≠ []) = (*i* < *k*) & *fst*(*f i*) = *p* @ *cs*)

⟨*proof*⟩

lemma *least-lem*: ⟦*f 0 = (c#cs,s)*; ∀ *i. f i → f(Suc i)*; ∃ *i. fst(f i) = cs* ⟧
⟹ ∃ *k. fst(f k) = cs* & *([c],s)* →* *([],snd(f k))*
⟨*proof*⟩

lemma *skolemize2*: ∀ *x.*∃ *y. P x y* ⟹ ∃ *f.*∀ *x. P x (f x)*
⟨*proof*⟩

lemma *inf-cases*: *inf (c#cs) s* ⟹ *inf [c] s* ∨ (∃ *t. s* −*c*→ *t* ∧ *inf cs t*)
⟨*proof*⟩

lemma *termi-impl-not-inf*: *c* ↓ *s* ⟹ ¬ *inf [c] s*
⟨*proof*⟩

lemma *termi-impl-no-inf-chain*:
c↓*s* ⟹ ¬(∃ *f. f 0 = ([c],s)* ∧ (∀ *i*::*nat. (f i, f(i+1)) : exec1*^+))
⟨*proof*⟩

primrec *cseq* :: (*nat* ⇒ *pname* × *state*) ⇒ *nat* ⇒ *com list* **where**
cseq S 0 = []
| *cseq S (Suc i) = (SOME cs. ([body(fst(S i))], snd(S i)) →**
(CALL(fst(S(i+1)))# cs, snd(S(i+1)))) @ cseq S i

lemma *wf-termi-call-steps*: *wf termi-call-steps*
⟨*proof*⟩

lemma *CALL-lemma*:
(⋃ *p.* {(λ*z s.* (*z*=*s* ∧ *body p*↓*s*) ∧ ((*p*,*s*),(*q*,*pre*)) ∈ *termi-call-steps*, *CALL p*,
λ*z s. z* −*body p*→ *s*)}) ⊢*t*
{λ*z s.* (*z*=*s* ∧ *body q*↓*pre*) ∧ (∃ *cs.* ([*body q*],*pre*) →* (*c*#*cs*,*s*))} *c*
{λ*z s. z* −*c*→ *s*}
⟨*proof*⟩

lemma *CALL-cor*:
(⋃ *p.* {(λ*z s.* (*z*=*s* ∧ *body p*↓*s*) ∧ ((*p*,*s*),(*q*,*pre*)) ∈ *termi-call-steps*, *CALL p*,
λ*z s. z* −*body p*→ *s*)}) ⊢*t*
{λ*z s.* (*z*=*s* ∧ *body q*↓*s*) ∧ *s* = *pre*} *body q* {λ*z s. z* −*body q*→ *s*}
⟨*proof*⟩

lemma *MGT-CALL*: {} |⊢*t* (⋃ *p.* {*MGT*$_t$(*CALL p*)})
⟨*proof*⟩

lemma *MGT-CALL1*: ∀ *p.* {} |⊢*t* {*MGT*$_t$(*CALL p*)}
⟨*proof*⟩

theorem {} ⊨*t* {*P*}*c*{*Q*} ⟹ {} ⊢*t* {*P*}*c*{*Q*::*state assn*}
⟨*proof*⟩

end

References

[1] T. Nipkow. Hoare logics for recursive procedures and unbounded nondeterminism. In J. Bradfield, editor, *Computer Science Logic (CSL 2002)*, volume 2471, pages 103–119, 2002.

[2] T. Nipkow. Hoare logics in Isabelle/HOL. In H. Schwichtenberg and R. Steinbrüggen, editors, *Proof and System-Reliability*, pages 341–367. Kluwer, 2002.

[3] D. v. Oheimb. Hoare logic for mutual recursion and local variables. In C. P. Rangan, V. Raman, and R. Ramanujam, editors, *Foundations of Software Technology and Theoretical Computer Science (FST&TCS)*, volume 1738, pages 168–180, 1999.