

pGCL for Isabelle

David Cock

September 13, 2023

Contents

1	Overview	1
2	Introduction to pGCL	3
2.1	Language Primitives	3
2.1.1	The Basics	3
2.1.2	Assertion and Annotation	4
2.1.3	Probability	4
2.1.4	Nondeterminism	5
2.1.5	Properties of Expectations	5
2.2	Loops	6
2.2.1	Guaranteed Termination	6
2.2.2	Probabilistic Termination	8
2.3	The Monty Hall Problem	9
2.3.1	The State Space	9
2.3.2	The Game	10
2.3.3	A Brute Force Solution	11
2.3.4	A Modular Approach	11
3	Semantic Structures	17
3.1	Expectations	17
3.1.1	Bounded Functions	18
3.1.2	Non-Negative Functions.	20
3.1.3	Sound Expectations	21
3.1.4	Unitary expectations	24
3.1.5	Standard Expectations	25
3.1.6	Entailment	27
3.1.7	Expectation Conjunction	28
3.1.8	Rules Involving Conjunction.	29
3.1.9	Rules Involving Entailment and Conjunction Together	31
3.2	Expectation Transformers	33
3.2.1	Comparing Transformers	35
3.2.2	Healthy Transformers	39
3.2.3	Sublinearity	46

3.2.4	Determinism	56
3.2.5	Modular Reasoning	59
3.2.6	Transforming Standard Expectations	60
3.3	Induction	62
3.3.1	The Lattice of Expectations	62
3.3.2	The Lattice of Transformers	66
3.3.3	Tail Recursion	71
4	The pGCL Language	75
4.1	A Shallow Embedding of pGCL in HOL	75
4.1.1	Core Primitives and Syntax	75
4.1.2	Unfolding rules for non-recursive primitives	79
4.2	Healthiness	81
4.2.1	The Healthiness of the Embedding	81
4.2.2	Healthiness for Loops	94
4.3	Continuity	104
4.3.1	Continuity of Primitives	105
4.3.2	Continuity of a Single Loop Step	122
4.4	Continuity and Induction for Loops	125
4.4.1	The Limit of Iterates	133
4.5	Sublinearity	142
4.5.1	Nonrecursive Primitives	142
4.5.2	Sublinearity for Loops	148
4.6	Determinism	152
4.6.1	Additivity	152
4.6.2	Maximality	153
4.6.3	Determinism	155
4.7	Well-Defined Programs.	156
4.7.1	Strict Implies Liberal	156
4.7.2	Sub-Distributivity of Conjunction	160
4.7.3	The Well-Defined Predicate.	170
4.8	The Loop Rules	173
4.8.1	Liberal and Strict Invariants.	173
4.8.2	Partial Correctness	174
4.8.3	Total Correctness	174
4.8.4	Unfolding	175
4.9	The Algebra of pGCL	176
4.9.1	Program Refinement	176
4.9.2	Simple Identities	177
4.9.3	Deterministic Programs are Maximal	182
4.9.4	The Algebraic Structure of Refinement	184
4.9.5	Data Refinement	187
4.9.6	The Algebra of Data Refinement	191
4.9.7	Structural Rules for Correspondence	198

4.9.8	Structural Rules for Data Refinement	199
4.10	Structured Reasoning	201
4.10.1	Syntactic Decomposition	201
4.10.2	Algebraic Decomposition	206
4.10.3	Hoare triples	206
4.11	Loop Termination	207
4.11.1	Trivial Termination	207
4.11.2	Classical Termination	208
4.11.3	Probabilistic Termination	211
4.12	Automated Reasoning	214
Additional Material		217
4.13	Miscellaneous Mathematics	217
4.13.1	Truncated Subtraction	223

Chapter 1

Overview

pGCL is both a programming language and a specification language that incorporates both probabilistic and nondeterministic choice, in a unified manner. Program verification is by *refinement* or *annotation* (or both), using either Hoare triples, or weakest-precondition entailment, in the style of GCL [[Dijkstra, 1975](#)].

This document is divided into three parts: [Chapter 2](#) gives a tutorial-style introduction to pGCL, and demonstrates the tools provided by the package; [Chapter 3](#) covers the development of the semantic interpretation: *expectation transformers*; and [Chapter 4](#) covers the formalisation of the language primitives, the associated *healthiness* results, and the tools for structured and automated reasoning. This second part follows the technical development of the pGCL theory package, in detail. It is not a great place to start learning pGCL. For that, see either the tutorial or [McIver and Morgan \[2004\]](#).

This formalisation was first presented (as an overview) in [Cock \[2012\]](#). The language has previously been formalised in HOL4 by [Hurd et al. \[2005\]](#). Two substantial results using this package were presented in [Cock \[2013\]](#), [Cock \[2014a\]](#) and [Cock \[2014b\]](#).

Chapter 2

Introduction to pGCL

2.1 Language Primitives

theory *Primitives* **imports** *../pGCL* **begin**

Programs in pGCL are probabilistic automata. They can do anything a traditional program can, plus, they may make truly probabilistic choices.

2.1.1 The Basics

Imagine flipping a pair of fair coins: a and b . Using a record type for the state allows a number of syntactic niceties, which we describe shortly:

datatype *coin* = *Heads* | *Tails*

record *coins* =

$a :: \textit{coin}$

$b :: \textit{coin}$

The primitive state operation is *Apply*, which takes a state transformer as an argument, constructs the pGCL equivalent. Thus *Apply* ($a\text{-update}$ ($\lambda\cdot$. *Heads*)) sets the value of coin a to *Heads*. As records are so common as state types, we introduce syntax to make these update neater: The same program may be defined more simply as *Apply* ($a\text{-update}$ ($\lambda\cdot$. *Heads*)) (note that the syntax translation involved does not apply to Latex output, and thus this lemma appears trivial):

lemma

$\textit{Apply} (\lambda s. s \langle a := \textit{Heads} \rangle) = (a := (\lambda s. \textit{Heads}))$

by(*simp*)

We can treat the record's fields as the names of *variables*. Note that the right-hand side of an assignment is always a function of the current state. Thus we may use a record accessor directly, for example *Apply* ($\lambda s. s \langle a := b \ s \rangle$), which updates a with the current value of b . If we wish to formally establish that the previous statement

is correct i.e. that in the final state, a really will have whatever value b had in the initial state, we must first introduce the assertion language.

2.1.2 Assertion and Annotation

Assertions in pGCL are real-valued functions of the state, which are often interpreted as a probability distribution over possible outcomes. These functions are termed *expectations*, for reasons which shortly be clear. Initially, however, we need only consider *standard* expectations: those derived from a binary predicate. A predicate $P::'s \Rightarrow bool$ is embedded as $\ll P \gg::'s \Rightarrow real$, such that $P s \longrightarrow \ll P \gg s = 1 \wedge \neg P s \longrightarrow \ll P \gg s = 0$.

An annotation consists of an assertion on the initial state and one on the final state, which for standard expectations may be interpreted as ‘if P holds in the initial state, then Q will hold in the final state’. These are in weakest-precondition form: we assert that the precondition implies the *weakest precondition*: the weakest assertion on the initial state, which implies that the postcondition must hold on the final state. So far, this is identical to the standard approach. Remember, however, that we are working with *real-valued* assertions. For standard expectations, the logic is nevertheless identical, if the implication $\forall s. P s \longrightarrow Q s$ is substituted with the equivalent expectation entailment $\ll P \gg \Vdash \ll Q \gg$, $\ll \ll ?P \gg \Vdash \ll ?Q \gg; ?P ?s \rrbracket \Longrightarrow ?Q ?s$. Thus a valid specification of *Apply* ($\lambda s. s(a := b s)$) is:

lemma

$$\bigwedge x. \ll \lambda s. b s = x \gg \Vdash wp (a := b) \ll \lambda s. a s = x \gg$$

by(pvcg, simp add:o-def)

Any ordinary computation and its associated annotation can be expressed in this form.

2.1.3 Probability

Next, we introduce the syntax $x ;; y$ for the sequential composition of x and y , and also demonstrate that one can operate directly on a real-valued (and thus infinite) state space:

lemma

$$\ll \lambda s::real. s \neq 0 \gg \Vdash wp (Apply ((* 2) ;; Apply (\lambda s. s / s)) \ll \lambda s. s = 1 \gg$$

by(pvcg, simp add:o-def)

So far, we haven’t done anything that required probabilities, or expectations other than 0 and 1. As an example of both, we show that a single coin toss is fair. We introduce the syntax $x \oplus_p y$ for a probabilistic choice between x and y . This program behaves as x with probability p , and as y with probability $(1::'a) - p$. The probability may depend on the state, and is therefore of type $'s \Rightarrow real$. The following annotation states that the probability of heads is exactly 1/2:

definition

flip-a :: *real* \Rightarrow *coins prog*
where
flip-a *p* = *a* := (λ -. *Heads*) (λ s. *p*) \oplus *a* := (λ -. *Tails*)

lemma
 $(\lambda$ s. $1/2$) = *wp* (*flip-a* ($1/2$)) \ll λ s. *a* *s* = *Heads* \gg
unfolding *flip-a-def*

Sufficiently small problems can be handled by the simplifier, by symbolic evaluation.

by(*simp add:wp-eval o-def*)

2.1.4 Nondeterminism

We can also under-specify a program, using the *nondeterministic choice* operator, $x \sqcap y$. This is interpreted demonically, giving the pointwise *minimum* of the pre-expectations for *x* and *y*: the chance of seeing heads, if your opponent is allowed choose between a pair of coins, one biased $2/3$ heads and one $2/3$ tails, and then flips it, is *at least* $1/3$, but we can make no stronger statement:

lemma
 λ s. $1/3 \Vdash$ *wp* (*flip-a* ($2/3$) \sqcap *flip-a* ($1/3$)) \ll λ s. *a* *s* = *Heads* \gg
unfolding *flip-a-def*
by *pvcg*

2.1.5 Properties of Expectations

The probabilities of independent events combine as usual, by multiplying: The chance of getting heads on two separate coins is $(1::'a) / (4::'a)$.

definition
flip-b :: *real* \Rightarrow *coins prog*
where
flip-b *p* = *b* := (λ -. *Heads*) (λ s. *p*) \oplus *b* := (λ -. *Tails*)

lemma
 $(\lambda$ s. $1/4$) = *wp* (*flip-a* ($1/2$) ;; *flip-b* ($1/2$))
 \ll λ s. *a* *s* = *Heads* \wedge *b* *s* = *Heads* \gg
unfolding *flip-a-def flip-b-def*
by(*simp add:wp-eval o-def*)

If, rather than two coins, we use two dice, we can make some slightly more involved calculations. We see that the weakest pre-expectation of the value on the face of the die after rolling is its *expected value* in the initial state, which justifies the use of the term expectation.

record *dice* =
red :: *nat*
blue :: *nat*

definition *Puniform* :: 'a set \Rightarrow ('a \Rightarrow real)
where *Puniform* $S = (\lambda x. \text{if } x \in S \text{ then } 1 / \text{card } S \text{ else } 0)$

lemma *Puniform-in*:
 $x \in S \Longrightarrow \text{Puniform } S \ x = 1 / \text{card } S$
by(*simp add:Puniform-def*)

lemma *Puniform-out*:
 $x \notin S \Longrightarrow \text{Puniform } S \ x = 0$
by(*simp add:Puniform-def*)

lemma *supp-Puniform*:
 $\text{finite } S \Longrightarrow \text{supp } (\text{Puniform } S) = S$
by(*auto simp:Puniform-def supp-def*)

The expected value of a roll of a six-sided die is $(7::'a) / (2::'a)$:

lemma
 $(\lambda s. 7/2) = \text{wp } (\text{bind } v \text{ at } (\lambda s. \text{Puniform } \{1..6\} \ v) \text{ in } \text{red} := (\lambda s. v)) \ \text{red}$
by(*simp add:wp-eval supp-Puniform sum.atLeast-Suc-atMost Puniform-in*)

The expectations of independent variables add:

lemma
 $(\lambda s. 7) = \text{wp } ((\text{bind } v \text{ at } (\lambda s. \text{Puniform } \{1..6\} \ v) \text{ in } \text{red} := (\lambda s. v)) \ ; \ ;$
 $(\text{bind } v \text{ at } (\lambda s. \text{Puniform } \{1..6\} \ v) \text{ in } \text{blue} := (\lambda s. v)))$
 $(\lambda s. \text{red } s + \text{blue } s)$
by(*simp add:wp-eval supp-Puniform sum.atLeast-Suc-atMost Puniform-in*)

end

2.2 Loops

theory *LoopExamples* **imports** *../pGCL* **begin**

Reasoning about loops in pGCL is mostly familiar, in particular in the use of invariants. Proving termination for truly probabilistic loops is slightly different: We appeal to a 0–1 law to show that the loop terminates *with probability 1*. In our semantic model, terminating with certainty and with probability 1 are exactly equivalent.

2.2.1 Guaranteed Termination

We start with a completely classical loop, to show that standard techniques apply. Here, we have a program that simply decrements a counter until it hits zero:

definition *countdown* :: *int prog*
where
 $\text{countdown} = \text{do } (\lambda x. 0 < x) \longrightarrow \text{Apply } (\lambda s. s - 1) \ \text{od}$

Clearly, this loop will only terminate from a state where $(0::'a) \leq x$. This is, in fact, also a loop invariant.

definition *inv-count* :: *int* \Rightarrow *bool*

where

inv-count = $(\lambda x. 0 \leq x)$

Read *wp-inv G body I* as: *I* is an invariant of the loop $\mu x. \text{body} ;; x \ll G \gg \oplus \text{Skip}$, or $\ll G \gg \&\& I \Vdash \text{wp body } I$.

lemma *wp-inv-count*:

wp-inv $(\lambda x. 0 < x)$ *(Apply* $(\lambda s. s - 1)$ *)* $\ll \text{inv-count} \gg$

unfolding *wp-inv-def inv-count-def wp-eval o-def*

proof(*clarify, cases*)

fix *x::int*

assume $0 \leq x$

then show $\ll \lambda x. 0 < x \gg x * \ll \lambda x. 0 \leq x \gg x \leq \ll \lambda x. 0 \leq x \gg (x - 1)$

by(*simp add:embed-bool-def*)

next

fix *x::int*

assume $\neg 0 \leq x$

then show $\ll \lambda x. 0 < x \gg x * \ll \lambda x. 0 \leq x \gg x \leq \ll \lambda x. 0 \leq x \gg (x - 1)$

by(*simp add:embed-bool-def*)

qed

This example is contrived to give us an obvious variant, or measure function: the counter itself.

lemma *term-countdown*:

$\ll \text{inv-count} \gg \Vdash \text{wp countdown } (\lambda s. 1)$

unfolding *countdown-def*

proof(*intro loop-term-nat-measure*[**where** $m = \lambda x. \text{nat } (\max x 0)$]) *wp-inv-count*)

let $?p = \text{Apply } (\lambda x. x - 1::\text{int})$

As usual, well-definedness is trivial.

show *well-def ?p*

by(*rule wd-intros*)

A measure of 0 implies termination.

show $\bigwedge x. \text{nat } (\max x 0) = 0 \longrightarrow \neg 0 < x$

by(*auto*)

This is the meat of the proof: that the measure must decrease, whenever the invariant holds. Note that the invariant is essential here, as if $x \leq (0::'a)$, the measure will *not* decrease.

This is the kind of proof that the VCG is good at. It leaves a purely logical goal, which we can solve with *auto*.

show $\bigwedge n. \ll \lambda x. \text{nat } (\max x 0) = \text{Suc } n \gg \&\& \ll \text{inv-count} \gg \Vdash$

wp ?p $\ll \lambda x. \text{nat } (\max x 0) = n \gg$

unfolding *inv-count-def*

by(*pvcg,*

auto simp: o-def exp-conj-std-split[symmetric]
intro: implies-entails)

qed

2.2.2 Probabilistic Termination

Loops need not terminate deterministically: it is sufficient to terminate with probability 1. Here we show the intuitively obvious result that by flipping a coin repeatedly, you will eventually see heads.

type-synonym *coin = bool*

definition *Heads = True*

definition *Tails = False*

definition

flip :: coin prog

where

flip = Apply ($\lambda\cdot$. Heads) ($\lambda s. 1/2$) \oplus Apply ($\lambda\cdot$. Tails)

We can't define a measure here, as we did previously, as neither of the two possible states guarantee termination.

definition

wait-for-heads :: coin prog

where

wait-for-heads = do ((\neq) Heads) \longrightarrow flip od

Nonetheless, we can show termination .

lemma *wait-for-heads-term:*

$\lambda s. 1 \Vdash wp$ wait-for-heads ($\lambda s. 1$)

unfolding *wait-for-heads-def*

We use one of the zero-one laws for termination, specifically that if, from every state there is a nonzero probability of satisfying the guard (and thus terminating) in a single step, then the loop terminates from *any* state, with probability 1.

proof(*rule termination-0-1*)

show *well-def flip*

unfolding *flip-def*

by(*auto intro:wd-intros*)

We must show that the loop body is deterministic, meaning that it cannot diverge by itself.

show *maximal (wp flip)*

unfolding *flip-def by(auto intro:max-intros)*

The verification condition for the loop body is one-step-termination, here shown to hold with probability 1/2. As usual, this result falls to the VCG.

show *$\lambda s. 1/2 \Vdash wp$ flip $\ll \mathcal{N} ((\neq) Heads) \gg$*

unfolding *flip-def*

by(*pvcg, simp add:o-def Heads-def Tails-def*)

Finally, the one-step escape probability is non-zero.

```
show (0::real) < 1/2 by(simp)
qed

end
```

2.3 The Monty Hall Problem

```
theory Monty imports ../pGCL begin
```

We now tackle a more substantial example, allowing us to demonstrate the tools for compositional reasoning and the use of invariants in non-recursive programs. Our example is the well-known Monty Hall puzzle in statistical inference [Selvin, 1975].

The setting is a game show: There is a prize hidden behind one of three doors, and the contestant is invited to choose one. Once the guess is made, the host then opens one of the remaining two doors, revealing a goat and showing that the prize is elsewhere. The contestant is then given the choice of switching their guess to the other unopened door, or sticking to their first guess.

The puzzle is whether the contestant is better off switching or staying put; or indeed whether it makes a difference at all. Most people's intuition suggests that it make no difference, whereas in fact, switching raises the chance of success from $1/3$ to $2/3$.

2.3.1 The State Space

The game state consists of the prize location, the guess, and the clue (the door the host opens). These are not constrained a priori to the range $\{1, 2, 3\}$, but are simply natural numbers: We instead show that this is in fact an invariant.

```
record game =
  prize :: nat
  guess :: nat
  clue  :: nat
```

The victory condition: The player wins if they have guessed the correct door, when the game ends.

```
definition player-wins :: game  $\Rightarrow$  bool
where player-wins g  $\equiv$  guess g = prize g
```

Invariants

We prove explicitly that only valid doors are ever chosen.

```
definition inv-prize :: game  $\Rightarrow$  bool
where inv-prize g  $\equiv$  prize g  $\in$  {1,2,3}
```

definition $inv-clue :: game \Rightarrow bool$
where $inv-clue\ g \equiv clue\ g \in \{1,2,3\}$

definition $inv-guess :: game \Rightarrow bool$
where $inv-guess\ g \equiv guess\ g \in \{1,2,3\}$

2.3.2 The Game

Hide the prize behind door D .

definition $hide-behind :: nat \Rightarrow game\ prog$
where $hide-behind\ D \equiv Apply\ (prize-update\ (\lambda x. D))$

Choose door D .

definition $guess-behind :: nat \Rightarrow game\ prog$
where $guess-behind\ D \equiv Apply\ (guess-update\ (\lambda x. D))$

Open door D and reveal what's behind.

definition $open-door :: nat \Rightarrow game\ prog$
where $open-door\ D \equiv Apply\ (clue-update\ (\lambda x. D))$

Hide the prize behind door 1, 2 or 3, demonically i.e. according to any probability distribution (or none).

definition $hide-prize :: game\ prog$
where $hide-prize \equiv hide-behind\ 1 \sqcap hide-behind\ 2 \sqcap hide-behind\ 3$

Guess uniformly at random.

definition $make-guess :: game\ prog$
where $make-guess \equiv guess-behind\ 1\ (\lambda s. 1/3) \oplus$
 $guess-behind\ 2\ (\lambda s. 1/2) \oplus guess-behind\ 3$

Open one of the two doors that *doesn't* hide the prize.

definition $reveal :: game\ prog$
where $reveal \equiv \sqcap d \in (\lambda s. \{1,2,3\} - \{prize\ s, guess\ s\}). open-door\ d$

Switch your guess to the other unopened door.

definition $switch-guess :: game\ prog$
where $switch-guess \equiv \sqcap d \in (\lambda s. \{1,2,3\} - \{clue\ s, guess\ s\}). guess-behind\ d$

The complete game, either with or without switching guesses.

definition $monty :: bool \Rightarrow game\ prog$
where
 $monty\ switch \equiv hide-prize\ ;;$
 $make-guess\ ;;$
 $reveal\ ;;$
 $(if\ switch\ then\ switch-guess\ else\ Skip)$

2.3.3 A Brute Force Solution

For sufficiently simple programs, we can calculate the exact weakest pre-expectation by unfolding.

lemma *eval-win*[simp]:

$$p = g \implies \langle \text{player-wins} \rangle (s \langle \text{prize} := p, \text{guess} := g, \text{clue} := c \rangle) = 1$$

by(simp add:embed-bool-def player-wins-def)

lemma *eval-loss*[simp]:

$$p \neq g \implies \langle \text{player-wins} \rangle (s \langle \text{prize} := p, \text{guess} := g, \text{clue} := c \rangle) = 0$$

by(simp add:embed-bool-def player-wins-def)

If they stick to their guns, the player wins with $p = 1/3$.

lemma *wp-monty-noswitch*:

$$(\lambda s. 1/3) = \text{wp} (\text{monty False}) \langle \text{player-wins} \rangle$$

unfolding *monty-def hide-prize-def make-guess-def reveal-def*
hide-behind-def guess-behind-def open-door-def
switch-guess-def

by(simp add:wp-eval insert-Diff-if o-def)

lemma *swap-upd*:

$$s \langle \text{prize} := p, \text{clue} := c, \text{guess} := g \rangle =$$

$$s \langle \text{prize} := p, \text{guess} := g, \text{clue} := c \rangle$$

by(simp)

If they switch, they win with $p = 2/3$. Brute force here takes longer, but is still feasible. On larger programs, this will rapidly become impossible, as the size of the terms (generally) grows exponentially with the length of the program.

lemma *wp-monty-switch-bruteforce*:

$$(\lambda s. 2/3) = \text{wp} (\text{monty True}) \langle \text{player-wins} \rangle$$

unfolding *monty-def hide-prize-def make-guess-def reveal-def*
hide-behind-def guess-behind-def open-door-def
switch-guess-def

— Note that this is getting slow

by (simp add: wp-eval insert-Diff-if swap-upd o-def cong del: INF-cong-simp)

2.3.4 A Modular Approach

We can solve the problem more efficiently, at the cost of a little more user effort, by breaking up the problem and annotating each step of the game separately. While this is not strictly necessary for this program, it will scale to larger examples, as the work in annotation only increases linearly with the length of the program.

Healthiness

We first establish healthiness for each step. This follows straightforwardly by applying the supplied rulesets.

lemma *wd-hide-prize*:
well-def hide-prize
unfolding *hide-prize-def hide-behind-def*
by(*simp add:wd-intros*)

lemma *wd-make-guess*:
well-def make-guess
unfolding *make-guess-def guess-behind-def*
by(*simp add:wd-intros*)

lemma *wd-reveal*:
well-def reveal
proof –

Here, we do need a subsidiary lemma: that there is always a ‘fresh’ door available. The rest of the healthiness proof follows as usual.

have $\bigwedge s. \{1, 2, 3\} - \{\text{prize } s, \text{guess } s\} \neq \{\}$
by(*auto simp:insert-Diff-if*)
thus *?thesis*
unfolding *reveal-def open-door-def*
by(*intro wd-intros, auto*)
qed

lemma *wd-switch-guess*:
well-def switch-guess
proof –
have $\bigwedge s. \{1, 2, 3\} - \{\text{clue } s, \text{guess } s\} \neq \{\}$
by(*auto simp:insert-Diff-if*)
thus *?thesis*
unfolding *switch-guess-def guess-behind-def*
by(*intro wd-intros, auto*)
qed

lemmas *monty-healthy* =
wd-switch-guess wd-reveal wd-make-guess wd-hide-prize

Annotations

We now annotate each step individually, and then combine them to produce an annotation for the entire program.

hide-prize chooses a valid door.

lemma *wp-hide-prize*:
 $(\lambda s. I) \Vdash \text{wp } \text{hide-prize} \llbracket \text{inv-prize} \rrbracket$
unfolding *hide-prize-def hide-behind-def wp-eval o-def*
by(*simp add:embed-bool-def inv-prize-def*)

Given the prize invariant, *make-guess* chooses a valid door, and guesses incorrectly with probability at least $2/3$.

lemma *wp-make-guess*:

$(\lambda s. 2/3 * \langle \lambda g. \text{inv-prize } g \rangle s) \Vdash$
 $\text{wp make-guess } \langle \lambda g. \text{guess } g \neq \text{prize } g \wedge \text{inv-prize } g \wedge \text{inv-guess } g \rangle$
unfolding *make-guess-def guess-behind-def wp-eval o-def*
by(*auto simp:embed-bool-def inv-prize-def inv-guess-def*)

lemma *last-one*:

assumes $a \neq b$ **and** $a \in \{1::\text{nat}, 2, 3\}$ **and** $b \in \{1, 2, 3\}$
shows $\exists! c. \{1, 2, 3\} - \{b, a\} = \{c\}$
apply(*simp add:insert-Diff-iff*)
using *assms* **by**(*auto intro:assms*)

Given the composed invariants, and an incorrect guess, *reveal* will give a clue that is neither the prize, nor the guess.

lemma *wp-reveal*:

$\langle \lambda g. \text{guess } g \neq \text{prize } g \wedge \text{inv-prize } g \wedge \text{inv-guess } g \rangle \Vdash$
 $\text{wp reveal } \langle \lambda g. \text{guess } g \neq \text{prize } g \wedge$
 $\text{clue } g \neq \text{prize } g \wedge$
 $\text{clue } g \neq \text{guess } g \wedge$
 $\text{inv-prize } g \wedge \text{inv-guess } g \wedge \text{inv-clue } g \rangle$
(is ?X \Vdash wp reveal ?Y)

proof(*rule use-premise, rule well-def-wp-healthy[OF wd-reveal], clarify*)

fix *s*

assume $\text{guess } s \neq \text{prize } s$
and $\text{inv-prize } s$
and $\text{inv-guess } s$

moreover then obtain *c*

where *singleton*: $\{ \text{Suc } 0, 2, 3 \} - \{ \text{prize } s, \text{guess } s \} = \{c\}$
and $c \neq \text{prize } s$
and $c \neq \text{guess } s$
and $c \in \{ \text{Suc } 0, 2, 3 \}$

unfolding *inv-prize-def inv-guess-def*

by(*force dest:last-one elim!:ex1E*)

ultimately show $1 \leq \text{wp reveal } ?Y s$

by(*simp add:reveal-def open-door-def wp-eval singleton o-def embed-bool-def inv-prize-def inv-guess-def inv-clue-def*)

qed

Showing that the three doors are all distinct is a largeish first-order problem, for which sledgehammer gives us a reasonable script.

lemma *distinct-game*:

$\llbracket \text{guess } g \neq \text{prize } g; \text{clue } g \neq \text{prize } g; \text{clue } g \neq \text{guess } g;$
 $\text{inv-prize } g; \text{inv-guess } g; \text{inv-clue } g \rrbracket \implies$
 $\{1, 2, 3\} = \{\text{guess } g, \text{prize } g, \text{clue } g\}$
unfolding *inv-prize-def inv-guess-def inv-clue-def*
apply(*rule set-eq1*)
apply(*rule iff1*)
apply(*clarify*)
apply(*metis (full-types) empty-iff insert-iff*)

```
apply(metis insert-iff)
done
```

Given the invariants, switching from the wrong guess gives the right one.

```
lemma wp-switch-guess:
  « $\lambda g. \text{guess } g \neq \text{prize } g \wedge \text{clue } g \neq \text{prize } g \wedge \text{clue } g \neq \text{guess } g \wedge$ 
     $\text{inv-prize } g \wedge \text{inv-guess } g \wedge \text{inv-clue } g$ »  $\Vdash$ 
  wp switch-guess «player-wins»
proof(rule use-premise, safe)
from wd-switch-guess show healthy (wp switch-guess) by(auto)

fix s
assume guess s  $\neq$  prize s and clue s  $\neq$  prize s
  and clue s  $\neq$  guess s and inv-prize s
  and inv-guess s and inv-clue s
note state = this
hence  $1 \leq \text{Inf } ((\lambda a. \langle \text{player-wins} \rangle (s(\text{guess} := a))))$  ‘
  ( $\{\text{guess } s, \text{prize } s, \text{clue } s\} - \{\text{clue } s, \text{guess } s\}$ )
  by(auto simp:insert-Diff-if player-wins-def)
also from state
have ... =  $\text{Inf } ((\lambda a. \langle \text{player-wins} \rangle (s(\text{guess} := a))))$  ‘
  ( $\{1, 2, 3\} - \{\text{clue } s, \text{guess } s\}$ )
  by(simp add:distinct-game[symmetric])
also have ... = wp switch-guess «player-wins» s
  by(simp add:switch-guess-def guess-behind-def wp-eval o-def)
finally show  $1 \leq \text{wp switch-guess } \langle \text{player-wins} \rangle s$  .
qed
```

Given componentwise specifications, we can glue them together with calculational reasoning to get our result.

```
lemma wp-monty-switch-modular:
  ( $\lambda s. 2/3$ )  $\Vdash$  wp (monty True) «player-wins»
proof(rule wp-validD) — Work in probabilistic Hoare triples
note wp-validI[OF wp-scale, OF wp-hide-prize, simplified]
  — Here we apply scaling to match our pre-expectation
also note wp-validI[OF wp-make-guess]
also note wp-validI[OF wp-reveal]
also note wp-validI[OF wp-switch-guess]
finally show  $\{\lambda s. 2/3\}$  monty True  $\{\langle \text{player-wins} \rangle\}$  p
  unfolding monty-def
  by(simp add:wd-intros sound-intros monty-healthy)
qed
```

Using the VCG

```
lemmas scaled-hide = wp-scale[OF wp-hide-prize, simplified]
declare scaled-hide[pwp] wp-make-guess[pwp] wp-reveal[pwp] wp-switch-guess[pwp]
declare wd-hide-prize[wd] wd-make-guess[wd] wd-reveal[wd] wd-switch-guess[wd]
```

Alternatively, the VCG will get this using the same annotations.

```
lemma wp-monty-switch-vcg:  
  ( $\lambda s. 2/3$ )  $\Vdash$  wp (monty True)  $\langle$ player-wins $\rangle$   
  unfolding monty-def  
  by(simp, pvcg)  
  
end
```


Chapter 3

Semantic Structures

3.1 Expectations

theory *Expectations* **imports** *Misc* **begin** **type-synonym** 's expect = 's \Rightarrow real

Expectations are a real-valued generalisation of boolean predicates: An expectation on state 's is a function 's \Rightarrow real. A predicate *P* on 's is embedded as an expectation by mapping *True* to 1 and *False* to 0. Under this embedding, implication becomes comparison, as the truth tables demonstrate:

<i>a</i>	<i>b</i>	$a \rightarrow b$	<i>x</i>	<i>y</i>	$x \leq y$
F	F	T	0	0	T
F	T	T	0	1	T
T	F	F	1	0	F
T	T	T	1	1	T

For probabilistic automata, an expectation gives the current expected value of some expression, if it were to be evaluated in the final state. For example, consider the automaton of [Figure 3.1](#), with transition probabilities affixed to edges. Let $P b = 2.0$ and $P c = 3.0$. Both states *b* and *c* are final (accepting) states, and thus the 'final expected value' of *P* in state *b* is 2.0 and in state *c* is 3.0. The expected value from state *a* is the weighted sum of these, or $0.7 \times 2.0 + 0.3 \times 3.0 = 2.3$.

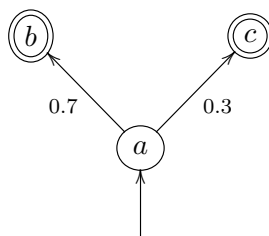


Figure 3.1: A probabilistic automaton

All expectations must be non-negative and bounded i.e. $\forall s. 0 \leq P s$ and $\exists b. \forall s. P s \leq b$. Note that although every expectation must have a bound, there is no bound on all expectations; In particular, the following series has no global bound, although each element is clearly bounded:

$$P_i = \lambda s. i \quad \text{where } i \in \mathbb{N}$$

3.1.1 Bounded Functions

definition *bounded-by* :: $real \Rightarrow ('a \Rightarrow real) \Rightarrow bool$
where *bounded-by* $b P \equiv \forall x. P x \leq b$

By instantiating the classical reasoner, both establishing and appealing to boundedness is largely automatic.

lemma *bounded-byI*[*intro*]:
 $\llbracket \bigwedge x. P x \leq b \rrbracket \Longrightarrow \text{bounded-by } b P$
by (*simp add:bounded-by-def*)

lemma *bounded-byI2*[*intro*]:
 $P \leq (\lambda s. b) \Longrightarrow \text{bounded-by } b P$
by (*blast dest:le-funD*)

lemma *bounded-byD*[*dest*]:
 $\text{bounded-by } b P \Longrightarrow P x \leq b$
by (*simp add:bounded-by-def*)

lemma *bounded-byD2*[*dest*]:
 $\text{bounded-by } b P \Longrightarrow P \leq (\lambda s. b)$
by (*blast intro:le-funI*)

A function is bounded if there exists at least one upper bound on it.

definition *bounded* :: $('a \Rightarrow real) \Rightarrow bool$
where *bounded* $P \equiv (\exists b. \text{bounded-by } b P)$

In the reals, if there exists any upper bound, then there must exist a least upper bound.

definition *bound-of* :: $('a \Rightarrow real) \Rightarrow real$
where *bound-of* $P \equiv \text{Sup } (P \text{ ` UNIV})$

lemma *bounded-bdd-above*[*intro*]:
assumes $bP: \text{bounded } P$
shows *bdd-above* (*range* P)
proof
fix x **assume** $x \in \text{range } P$
with bP **show** $x \leq \text{Inf } \{b. \text{bounded-by } b P\}$
unfolding *bounded-def* **by** (*auto intro:cInf-greatest*)
qed

The least upper bound has the usual properties:

lemma *bound-of-least*[intro]:
assumes bP : *bounded-by* b P
shows *bound-of* $P \leq b$
unfolding *bound-of-def*
using bP **by** (*intro cSup-least, auto*)

lemma *bounded-by-bound-of*[intro!]:
fixes $P::'a \Rightarrow \text{real}$
assumes bP : *bounded* P
shows *bounded-by* (*bound-of* P) P
unfolding *bound-of-def*
using bP **by** (*intro bounded-byI cSup-upper bounded-bdd-above, auto*)

lemma *bound-of-greater*[intro]:
bounded $P \Longrightarrow P\ x \leq \text{bound-of } P$
by (*blast intro:bounded-byD*)

lemma *bounded-by-mono*:
 $\llbracket \text{bounded-by } a\ P; a \leq b \rrbracket \Longrightarrow \text{bounded-by } b\ P$
unfolding *bounded-by-def* **by** (*blast intro:order-trans*)

lemma *bounded-by-imp-bounded*[intro]:
bounded-by $b\ P \Longrightarrow \text{bounded } P$
unfolding *bounded-def* **by** (*blast*)

This is occasionally easier to apply:

lemma *bounded-by-bound-of-alt*:
 $\llbracket \text{bounded } P; \text{bound-of } P = a \rrbracket \Longrightarrow \text{bounded-by } a\ P$
by (*blast*)

lemma *bounded-const*[simp]:
bounded $(\lambda x. c)$
by (*blast*)

lemma *bounded-by-const*[intro]:
 $c \leq b \Longrightarrow \text{bounded-by } b\ (\lambda x. c)$
by (*blast*)

lemma *bounded-by-mono-alt*[intro]:
 $\llbracket \text{bounded-by } b\ Q; P \leq Q \rrbracket \Longrightarrow \text{bounded-by } b\ P$
by (*blast intro:order-trans dest:le-funD*)

lemma *bound-of-const*[simp, intro]:
bound-of $(\lambda x. c) = (c::\text{real})$
unfolding *bound-of-def*
by (*intro antisym cSup-least cSup-upper bounded-bdd-above bounded-const, auto*)

lemma *bound-of-leI*:

assumes $\bigwedge x. P x \leq (c::real)$
shows $bound\text{-}of\ P \leq c$
unfolding $bound\text{-}of\text{-}def$
using $assms$ **by** $(intro\ cSup\text{-}least, auto)$

lemma $bound\text{-}of\text{-}mono[intro]$:
 $\llbracket P \leq Q; bounded\ P; bounded\ Q \rrbracket \implies bound\text{-}of\ P \leq bound\text{-}of\ Q$
by $(blast\ intro:order\text{-}trans\ dest:le\text{-}funD)$

lemma $bounded\text{-}by\text{-}o[intro,simp]$:
 $\bigwedge b. bounded\text{-}by\ b\ P \implies bounded\text{-}by\ b\ (P\ o\ f)$
unfolding $o\text{-}def$ **by** $(blast)$

lemma $le\text{-}bound\text{-}of[intro]$:
 $\bigwedge x. bounded\ f \implies f\ x \leq bound\text{-}of\ f$
by $(blast)$

3.1.2 Non-Negative Functions.

The definitions for non-negative functions are analogous to those for bounded functions.

definition
 $nneg :: ('a \Rightarrow 'b::\{zero,order\}) \Rightarrow bool$
where
 $nneg\ P \longleftrightarrow (\forall x. 0 \leq P\ x)$

lemma $nnegI[intro]$:
 $\llbracket \bigwedge x. 0 \leq P\ x \rrbracket \implies nneg\ P$
by $(simp\ add:nneg\text{-}def)$

lemma $nnegI2[intro]$:
 $(\lambda s. 0) \leq P \implies nneg\ P$
by $(blast\ dest:le\text{-}funD)$

lemma $nnegD[dest]$:
 $nneg\ P \implies 0 \leq P\ x$
by $(simp\ add:nneg\text{-}def)$

lemma $nnegD2[dest]$:
 $nneg\ P \implies (\lambda s. 0) \leq P$
by $(blast\ intro:le\text{-}funI)$

lemma $nneg\text{-}bdd\text{-}below[intro]$:
 $nneg\ P \implies bdd\text{-}below\ (range\ P)$
by $(auto)$

lemma $nneg\text{-}const[iff]$:
 $nneg\ (\lambda x. c) \longleftrightarrow 0 \leq c$
by $(simp\ add:nneg\text{-}def)$

lemma *nneg-o*[*intro,simp*]:
 $nneg\ P \Longrightarrow nneg\ (P\ o\ f)$
by (*force*)

lemma *nneg-bound-nneg*[*intro*]:
 $\llbracket bounded\ P; nneg\ P \rrbracket \Longrightarrow 0 \leq bound-of\ P$
by (*blast intro:order-trans*)

lemma *nneg-bounded-by-nneg*[*dest*]:
 $\llbracket bounded-by\ b\ P; nneg\ P \rrbracket \Longrightarrow 0 \leq (b::real)$
by (*blast intro:order-trans*)

lemma *bounded-by-nneg*[*dest*]:
fixes $P::'s \Rightarrow real$
shows $\llbracket bounded-by\ b\ P; nneg\ P \rrbracket \Longrightarrow 0 \leq b$
by (*blast intro:order-trans*)

3.1.3 Sound Expectations

definition *sound* :: $('s \Rightarrow real) \Rightarrow bool$
where $sound\ P \equiv bounded\ P \wedge nneg\ P$

Combining *nneg* and *Expectations.bounded*, we have *sound* expectations. We set up the classical reasoner and the simplifier, such that showing soundness, or deriving a simple consequence (e.g. $sound\ P \Longrightarrow 0 \leq P\ s$) will usually follow by *blast*, *force* or *simp*.

lemma *soundI*:
 $\llbracket bounded\ P; nneg\ P \rrbracket \Longrightarrow sound\ P$
by (*simp add:sound-def*)

lemma *soundI2*[*intro*]:
 $\llbracket bounded-by\ b\ P; nneg\ P \rrbracket \Longrightarrow sound\ P$
by(*blast intro:soundI*)

lemma *sound-bounded*[*dest*]:
 $sound\ P \Longrightarrow bounded\ P$
by (*simp add:sound-def*)

lemma *sound-nneg*[*dest*]:
 $sound\ P \Longrightarrow nneg\ P$
by (*simp add:sound-def*)

lemma *bound-of-sound*[*intro*]:
assumes $sP: sound\ P$
shows $0 \leq bound-of\ P$
using *assms* **by**(*auto*)

This proof demonstrates the use of the classical reasoner (specifically *blast*), to

both introduce and eliminate soundness terms.

lemma *sound-sum*[*simp,intro*]:

assumes sP : *sound* P **and** sQ : *sound* Q

shows *sound* $(\lambda s. P s + Q s)$

proof

from sP **have** $\bigwedge s. P s \leq \text{bound-of } P$ **by**(*blast*)

moreover from sQ **have** $\bigwedge s. Q s \leq \text{bound-of } Q$ **by**(*blast*)

ultimately have $\bigwedge s. P s + Q s \leq \text{bound-of } P + \text{bound-of } Q$
by(*rule add-mono*)

thus bounded-by $(\text{bound-of } P + \text{bound-of } Q)$ $(\lambda s. P s + Q s)$ **by**(*blast*)

from sP **have** $\bigwedge s. 0 \leq P s$ **by**(*blast*)

moreover from sQ **have** $\bigwedge s. 0 \leq Q s$ **by**(*blast*)

ultimately have $\bigwedge s. 0 \leq P s + Q s$ **by**(*simp add:add-mono*)

thus nneg $(\lambda s. P s + Q s)$ **by**(*blast*)

qed

lemma *mult-sound*:

assumes sP : *sound* P **and** sQ : *sound* Q

shows *sound* $(\lambda s. P s * Q s)$

proof

from sP **have** $\bigwedge s. P s \leq \text{bound-of } P$ **by**(*blast*)

moreover from sQ **have** $\bigwedge s. Q s \leq \text{bound-of } Q$ **by**(*blast*)

ultimately have $\bigwedge s. P s * Q s \leq \text{bound-of } P * \text{bound-of } Q$
using sP **and** sQ **by**(*blast intro:mult-mono*)

thus bounded-by $(\text{bound-of } P * \text{bound-of } Q)$ $(\lambda s. P s * Q s)$ **by**(*blast*)

from sP **and** sQ **show** *nneg* $(\lambda s. P s * Q s)$

by(*blast intro:mult-nonneg-nonneg*)

qed

lemma *div-sound*:

assumes sP : *sound* P **and** $cpos$: $0 < c$

shows *sound* $(\lambda s. P s / c)$

proof

from sP **and** $cpos$ **have** $\bigwedge s. P s / c \leq \text{bound-of } P / c$

by(*blast intro:divide-right-mono less-imp-le*)

thus bounded-by $(\text{bound-of } P / c)$ $(\lambda s. P s / c)$ **by**(*blast*)

from *assms* **show** *nneg* $(\lambda s. P s / c)$

by(*blast intro:divide-nonneg-pos*)

qed

lemma *tminus-sound*:

assumes sP : *sound* P **and** nnc : $0 \leq c$

shows *sound* $(\lambda s. P s \ominus c)$

proof(*rule soundI*)

from sP **have** $\bigwedge s. P s \leq \text{bound-of } P$ **by**(*blast*)

with nnc **have** $\bigwedge s. P s \ominus c \leq \text{bound-of } P \ominus c$

```

by(blast intro:tminus-left-mono)
thus bounded ( $\lambda s. P s \ominus c$ ) by(blast)
show nneg ( $\lambda s. P s \ominus c$ ) by(blast)
qed

```

```

lemma const-sound:
 $0 \leq c \implies$  sound ( $\lambda s. c$ )
by (blast)

```

```

lemma sound-o[intro,simp]:
sound  $P \implies$  sound ( $P$  of)
unfolding o-def by(blast)

```

```

lemma sc-bounded-by[intro,simp]:
 $\llbracket$  sound  $P; 0 \leq c \rrbracket \implies$  bounded-by ( $c *$  bound-of  $P$ ) ( $\lambda x. c * P x$ )
by(blast intro!:mult-left-mono)

```

```

lemma sc-bounded[intro,simp]:
assumes sP: sound  $P$  and pos:  $0 \leq c$ 
shows bounded ( $\lambda x. c * P x$ )
using assms by(blast)

```

```

lemma sc-bound[simp]:
assumes sP: sound  $P$ 
and cnn:  $0 \leq c$ 
shows  $c *$  bound-of  $P =$  bound-of ( $\lambda x. c * P x$ )
proof(cases  $c = 0$ )
case True then show ?thesis by(simp)
next
case False with cnn have cpos:  $0 < c$  by(auto)
show ?thesis
proof (rule antisym)
from sP and cnn have bounded ( $\lambda x. c * P x$ ) by(simp)
hence  $\bigwedge x. c * P x \leq$  bound-of ( $\lambda x. c * P x$ )
by(rule le-bound-of)
with cpos have  $\bigwedge x. P x \leq$  inverse  $c *$  bound-of ( $\lambda x. c * P x$ )
by(force intro:mult-div-mono-right)
hence bound-of  $P \leq$  inverse  $c *$  bound-of ( $\lambda x. c * P x$ )
by(blast)
with cpos show  $c *$  bound-of  $P \leq$  bound-of ( $\lambda x. c * P x$ )
by(force intro:mult-div-mono-left)
next
from sP and cpos have  $\bigwedge x. c * P x \leq c *$  bound-of  $P$ 
by(blast intro:mult-left-mono less-imp-le)
thus bound-of ( $\lambda x. c * P x$ )  $\leq c *$  bound-of  $P$ 
by(blast)
qed
qed

```

lemma *sc-sound*:

$\llbracket \text{sound } P; 0 \leq c \rrbracket \implies \text{sound } (\lambda s. c * P s)$
by (*blast intro:mult-nonneg-nonneg*)

lemma *bounded-by-mult*:

assumes *sP*: *sound P* **and** *bP*: *bounded-by a P*
and *sQ*: *sound Q* **and** *bQ*: *bounded-by b Q*
shows *bounded-by (a * b) (λs. P s * Q s)*
using *assms by(intro bounded-byI, auto intro:mult-mono)*

lemma *bounded-by-add*:

fixes *P::'s ⇒ real* **and** *Q*
assumes *bP*: *bounded-by a P*
and *bQ*: *bounded-by b Q*
shows *bounded-by (a + b) (λs. P s + Q s)*
using *assms by(intro bounded-byI, auto intro:add-mono)*

lemma *sound-unit[intro!,simp]*:

sound (λs. 1)
by(*auto*)

lemma *unit-mult[intro]*:

assumes *sP*: *sound P* **and** *bP*: *bounded-by 1 P*
and *sQ*: *sound Q* **and** *bQ*: *bounded-by 1 Q*
shows *bounded-by 1 (λs. P s * Q s)*
proof(*rule bounded-byI*)
fix *s*
have $P s * Q s \leq 1 * 1$
using *assms by(blast dest:bounded-by-mult)*
thus $P s * Q s \leq 1$ **by**(*simp*)
qed

lemma *sum-sound*:

assumes *sP*: $\forall x \in S. \text{sound } (P x)$
shows *sound (λs. ∑ x ∈ S. P x s)*
proof(*rule soundI2*)
from *sP* **show** *bounded-by (∑ x ∈ S. bound-of (P x)) (λs. ∑ x ∈ S. P x s)*
by(*auto intro!:sum-mono*)
from *sP* **show** *nneg (λs. ∑ x ∈ S. P x s)*
by(*auto intro!:sum-nonneg*)
qed

3.1.4 Unitary expectations

A unitary expectation is a sound expectation that is additionally bounded by one. This is the domain on which the *liberal* (partial correctness) semantics operates.

definition *unitary :: 's expect ⇒ bool*

where *unitary P* $\longleftrightarrow \text{sound } P \wedge \text{bounded-by } 1 P$

lemma *unitaryI*[*intro*]:
 $\llbracket \text{sound } P; \text{bounded-by } 1 P \rrbracket \Longrightarrow \text{unitary } P$
by(*simp add:unitary-def*)

lemma *unitaryI2*:
 $\llbracket \text{nneg } P; \text{bounded-by } 1 P \rrbracket \Longrightarrow \text{unitary } P$
by(*auto*)

lemma *unitary-sound*[*dest*]:
 $\text{unitary } P \Longrightarrow \text{sound } P$
by(*simp add:unitary-def*)

lemma *unitary-bound*[*dest*]:
 $\text{unitary } P \Longrightarrow \text{bounded-by } 1 P$
by(*simp add:unitary-def*)

3.1.5 Standard Expectations

definition

embed-bool :: ($'s \Rightarrow \text{bool}$) $\Rightarrow 's \Rightarrow \text{real}$ ($\llbracket - \rrbracket 1000$)

where

$\llbracket P \rrbracket \equiv (\lambda s. \text{if } P s \text{ then } 1 \text{ else } 0)$

Standard expectations are the embeddings of boolean predicates, mapping *False* to 0 and *True* to 1. We write $\llbracket P \rrbracket$ rather than $[P]$ (the syntax employed by [McIver and Morgan \[2004\]](#)) for boolean embedding to avoid clashing with the HOL syntax for lists.

lemma *embed-bool-nneg*[*simp,intro*]:
 $\text{nneg } \llbracket P \rrbracket$
unfolding *embed-bool-def* **by**(*force*)

lemma *embed-bool-bounded-by-1*[*simp,intro*]:
 $\text{bounded-by } 1 \llbracket P \rrbracket$
unfolding *embed-bool-def* **by**(*force*)

lemma *embed-bool-bounded*[*simp,intro*]:
 $\text{bounded } \llbracket P \rrbracket$
by (*blast*)

Standard expectations have a number of convenient properties, which mostly follow from boolean algebra.

lemma *embed-bool-idem*:
 $\llbracket P \rrbracket s * \llbracket P \rrbracket s = \llbracket P \rrbracket s$
by (*simp add:embed-bool-def*)

lemma *eval-embed-true*[*simp*]:
 $P s \Longrightarrow \llbracket P \rrbracket s = 1$
by (*simp add:embed-bool-def*)

lemma *eval-embed-false*[simp]:
 $\neg P\ s \implies \langle\langle P \rangle\rangle\ s = 0$
by (*simp add:embed-bool-def*)

lemma *embed-ge-0*[simp,intro]:
 $0 \leq \langle\langle G \rangle\rangle\ s$
by (*simp add:embed-bool-def*)

lemma *embed-le-1*[simp,intro]:
 $\langle\langle G \rangle\rangle\ s \leq 1$
by(*simp add:embed-bool-def*)

lemma *embed-le-1-alt*[simp,intro]:
 $0 \leq 1 - \langle\langle G \rangle\rangle\ s$
by(*subst add-le-cancel-right*[**where** $c = \langle\langle G \rangle\rangle\ s$, *symmetric*], *simp*)

lemma *expect-1-I*:
 $P\ x \implies 1 \leq \langle\langle P \rangle\rangle\ x$
by(*simp*)

lemma *standard-sound*[intro,simp]:
sound $\langle\langle P \rangle\rangle$
by(*blast*)

lemma *embed-o*[simp]:
 $\langle\langle P \rangle\rangle\ o\ f = \langle\langle P\ o\ f \rangle\rangle$
unfolding *embed-bool-def o-def* **by**(*simp*)

Negating a predicate has the expected effect in its embedding as an expectation:

definition *negate* :: $(s \Rightarrow \text{bool}) \Rightarrow s \Rightarrow \text{bool}$ (\mathcal{N})
where $\text{negate } P = (\lambda s. \neg P\ s)$

lemma *negateI*:
 $\neg P\ s \implies \mathcal{N}\ P\ s$
by (*simp add:negate-def*)

lemma *embed-split*:
 $f\ s = \langle\langle P \rangle\rangle\ s * f\ s + \langle\langle \mathcal{N}\ P \rangle\rangle\ s * f\ s$
by (*simp add:negate-def embed-bool-def*)

lemma *negate-embed*:
 $\langle\langle \mathcal{N}\ P \rangle\rangle\ s = 1 - \langle\langle P \rangle\rangle\ s$
by (*simp add:embed-bool-def negate-def*)

lemma *eval-nembed-true*[simp]:
 $P\ s \implies \langle\langle \mathcal{N}\ P \rangle\rangle\ s = 0$
by (*simp add:embed-bool-def negate-def*)

lemma *eval-nembed-false*[simp]:

$\neg P s \implies \ll \mathcal{N} P \gg s = 1$
by (*simp add:embed-bool-def negate-def*)

lemma *negate-Not*[*simp*]:
 $\mathcal{N} \text{Not} = (\lambda x. x)$
by(*simp add:negate-def*)

lemma *negate-negate*[*simp*]:
 $\mathcal{N} (\mathcal{N} P) = P$
by(*simp add:negate-def*)

lemma *embed-bool-cancel*:
 $\ll G \gg s * \ll \mathcal{N} G \gg s = 0$
by(*cases G s, simp-all*)

3.1.6 Entailment

Entailment on expectations is a generalisation of that on predicates, and is defined by pointwise comparison:

abbreviation *entails* :: ($'s \Rightarrow \text{real}$) \Rightarrow ($'s \Rightarrow \text{real}$) \Rightarrow *bool* (- \Vdash - 50)
where $P \Vdash Q \equiv P \leq Q$

lemma *entailsI*[*intro*]:
 $\ll \bigwedge s. P s \leq Q s \gg \implies P \Vdash Q$
by(*simp add:le-funI*)

lemma *entailsD*[*dest*]:
 $P \Vdash Q \implies P s \leq Q s$
by(*simp add:le-funD*)

lemma *eq-entails*[*intro*]:
 $P = Q \implies P \Vdash Q$
by(*blast*)

lemma *entails-trans*[*trans*]:
 $\ll P \Vdash Q; Q \Vdash R \gg \implies P \Vdash R$
by(*blast intro:order-trans*)

For standard expectations, both notions of entailment coincide. This result justifies the above claim that our definition generalises predicate entailment:

lemma *implies-entails*:
 $\ll \bigwedge s. P s \implies Q s \gg \implies \ll P \gg \Vdash \ll Q \gg$
by(*rule entailsI, case-tac P s, simp-all*)

lemma *entails-implies*:
 $\bigwedge s. \ll \ll P \gg \Vdash \ll Q \gg; P s \gg \implies Q s$
by(*rule ccontr, drule-tac s=s in entailsD, simp*)

3.1.7 Expectation Conjunction

definition

$pconj :: real \Rightarrow real \Rightarrow real$ (**infixl** $.\&$ 71)

where

$p.\&q \equiv p + q \ominus 1$

definition

$exp-conj :: ('s \Rightarrow real) \Rightarrow ('s \Rightarrow real) \Rightarrow ('s \Rightarrow real)$ (**infixl** $\&\&$ 71)

where $a \&\&b \equiv \lambda s. (a\ s.\&b\ s)$

Expectation conjunction likewise generalises (boolean) predicate conjunction. We show that the expected properties are preserved, and instantiate both the classical reasoner, and the simplifier (in the case of associativity and commutativity).

lemma $pconj-lzero$ [*intro,simp*]:

$b \leq 1 \implies 0.\&b = 0$

by(*simp add:pconj-def tminus-def*)

lemma $pconj-rzero$ [*intro,simp*]:

$b \leq 1 \implies b.\&0 = 0$

by(*simp add:pconj-def tminus-def*)

lemma $pconj-lone$ [*intro,simp*]:

$0 \leq b \implies 1.\&b = b$

by(*simp add:pconj-def tminus-def*)

lemma $pconj-rone$ [*intro,simp*]:

$0 \leq b \implies b.\&1 = b$

by(*simp add:pconj-def tminus-def*)

lemma $pconj-bconj$:

$\langle\langle a \rangle\rangle s.\&\langle\langle b \rangle\rangle s = \langle\lambda s. a\ s \wedge b\ s\rangle s$

unfolding *embed-bool-def pconj-def tminus-def* **by**(*force*)

lemma $pconj-comm$ [*ac-simps*]:

$a.\&b = b.\&a$

by(*simp add:pconj-def ac-simps*)

lemma $pconj-assoc$:

$\llbracket 0 \leq a; a \leq 1; 0 \leq b; b \leq 1; 0 \leq c; c \leq 1 \rrbracket \implies$

$a.\&(b.\&c) = (a.\&b).\&c$

unfolding *pconj-def tminus-def* **by**(*simp*)

lemma $pconj-mono$:

$\llbracket a \leq b; c \leq d \rrbracket \implies a.\&c \leq b.\&d$

unfolding *pconj-def tminus-def* **by**(*simp*)

lemma $pconj-nneg$ [*intro,simp*]:

$0 \leq a.\&b$

unfolding *pconj-def tminus-def* **by**(*auto*)

lemma *min-pconj*:

$(\min a b) \cdot \& (\min c d) \leq \min (a \cdot \& c) (b \cdot \& d)$

by(*cases a ≤ b*,
cases c ≤ d,
simp-all add:min.absorb1 min.absorb2 pconj-mono),
cases c ≤ d,
simp-all add:min.absorb1 min.absorb2 pconj-mono)

lemma *pconj-less-one*[*simp*]:

$a + b < 1 \implies a \cdot \& b = 0$

unfolding *pconj-def* **by**(*simp*)

lemma *pconj-ge-one*[*simp*]:

$1 \leq a + b \implies a \cdot \& b = a + b - 1$

unfolding *pconj-def* **by**(*simp*)

lemma *pconj-idem*[*simp*]:

$\langle\langle P \rangle\rangle s \cdot \& \langle\langle P \rangle\rangle s = \langle\langle P \rangle\rangle s$

unfolding *pconj-def* **by**(*cases P s*, *simp-all*)

3.1.8 Rules Involving Conjunction.

lemma *exp-conj-mono-left*:

$P \Vdash Q \implies P \&\& R \Vdash Q \&\& R$

unfolding *exp-conj-def pconj-def*
by(*auto intro:tminus-left-mono add-right-mono*)

lemma *exp-conj-mono-right*:

$Q \Vdash R \implies P \&\& Q \Vdash P \&\& R$

unfolding *exp-conj-def pconj-def*
by(*auto intro:tminus-left-mono add-left-mono*)

lemma *exp-conj-comm*[*ac-simps*]:

$a \&\& b = b \&\& a$

by(*simp add:exp-conj-def ac-simps*)

lemma *exp-conj-bounded-by*[*intro, simp*]:

assumes *bP: bounded-by 1 P*

and *bQ: bounded-by 1 Q*

shows *bounded-by 1 (P &\& Q)*

proof(*rule bounded-byI, unfold exp-conj-def pconj-def*)

fix *x*

from *bP* **have** $P x \leq 1$ **by**(*blast*)

moreover from *bQ* **have** $Q x \leq 1$ **by**(*blast*)

ultimately have $P x + Q x \leq 2$ **by**(*auto*)

thus $P x + Q x \ominus 1 \leq 1$

unfolding *tminus-def* **by**(*simp*)

qed

lemma *exp-conj-o-distrib*[simp]:
 $(P \ \&\& \ Q) \ of = (P \ of) \ \&\& \ (Q \ of)$
unfolding *exp-conj-def o-def* **by**(*simp*)

lemma *exp-conj-assoc*:
assumes *unitary P and unitary Q and unitary R*
shows $P \ \&\& \ (Q \ \&\& \ R) = (P \ \&\& \ Q) \ \&\& \ R$
unfolding *exp-conj-def*
proof(*rule ext*)
fix s
from *assms* **have** $0 \leq P \ s$ **by**(*blast*)
moreover from *assms* **have** $0 \leq Q \ s$ **by**(*blast*)
moreover from *assms* **have** $0 \leq R \ s$ **by**(*blast*)
moreover from *assms* **have** $P \ s \leq 1$ **by**(*blast*)
moreover from *assms* **have** $Q \ s \leq 1$ **by**(*blast*)
moreover from *assms* **have** $R \ s \leq 1$ **by**(*blast*)
ultimately
show $P \ s \ .\& \ (Q \ s \ .\& \ R \ s) = (P \ s \ .\& \ Q \ s) \ .\& \ R \ s$
by(*simp add:pconj-assoc*)
qed

lemma *exp-conj-top-left*[simp]:
 $\text{sound } P \implies \langle \lambda_. \text{True} \rangle \ \&\& \ P = P$
unfolding *exp-conj-def* **by**(*force*)

lemma *exp-conj-top-right*[simp]:
 $\text{sound } P \implies P \ \&\& \ \langle \lambda_. \text{True} \rangle = P$
unfolding *exp-conj-def* **by**(*force*)

lemma *exp-conj-idem*[simp]:
 $\langle P \rangle \ \&\& \ \langle P \rangle = \langle P \rangle$
unfolding *exp-conj-def*
by(*rule ext, cases P s, simp-all*)

lemma *exp-conj-nneg*[intro,simp]:
 $(\lambda s. 0) \leq P \ \&\& \ Q$
unfolding *exp-conj-def*
by(*blast intro:le-funI*)

lemma *exp-conj-sound*[intro,simp]:
assumes $s\text{-}P$: *sound P*
and $s\text{-}Q$: *sound Q*
shows *sound (P && Q)*
unfolding *exp-conj-def*
proof(*rule soundI*)
from $s\text{-}P$ **and** $s\text{-}Q$ **have** $\bigwedge s. 0 \leq P \ s + Q \ s$ **by**(*blast intro:add-nonneg-nonneg*)
hence $\bigwedge s. P \ s \ .\& \ Q \ s \leq P \ s + Q \ s$

```

unfolding pconj-def by(force intro:tminus-less)
also from assms have  $\bigwedge s. \dots s \leq \text{bound-of } P + \text{bound-of } Q$ 
by(blast intro:add-mono)
finally have bounded-by (bound-of P + bound-of Q) ( $\lambda s. P s \cdot\& Q s$ )
by(blast)
thus bounded ( $\lambda s. P s \cdot\& Q s$ ) by(blast)

```

```

show nneg ( $\lambda s. P s \cdot\& Q s$ )
unfolding pconj-def tminus-def by(force)
qed

```

```

lemma exp-conj-rzero[simp]:
bounded-by 1 P  $\implies P \&\& (\lambda s. 0) = (\lambda s. 0)$ 
unfolding exp-conj-def by(force)

```

```

lemma exp-conj-1-right[simp]:
assumes nn: nneg A
shows  $A \&\& (\lambda \cdot. I) = A$ 
unfolding exp-conj-def pconj-def tminus-def
proof(rule ext, simp)
fix s
from nn have  $0 \leq A s$  by(blast)
thus  $\max (A s) 0 = A s$  by(force)
qed

```

```

lemma exp-conj-std-split:
 $\llbracket \lambda s. P s \wedge Q s \rrbracket = \llbracket P \rrbracket \&\& \llbracket Q \rrbracket$ 
unfolding exp-conj-def embed-bool-def pconj-def
by(auto)

```

3.1.9 Rules Involving Entailment and Conjunction Together

Meta-conjunction distributes over expectation entailment, becoming expectation conjunction:

```

lemma entails-frame:
assumes ePR:  $P \Vdash R$ 
and eQS:  $Q \Vdash S$ 
shows  $P \&\& Q \Vdash R \&\& S$ 
proof(rule le-funI)
fix s
from ePR have  $P s \leq R s$  by(blast)
moreover from eQS have  $Q s \leq S s$  by(blast)
ultimately have  $P s + Q s \leq R s + S s$  by(rule add-mono)
hence  $P s + Q s \ominus I \leq R s + S s \ominus I$  by(rule tminus-left-mono)
thus  $(P \&\& Q) s \leq (R \&\& S) s$ 
unfolding exp-conj-def pconj-def .
qed

```

This rule allows something very much akin to a case distinction on the pre-expectation.

lemma *pentails-cases*:
assumes $PQe: \bigwedge x. P x \Vdash Q x$
and *exhaust*: $\bigwedge s. \exists x. P (x s) s = 1$
and *framed*: $\bigwedge x. P x \&\& R \Vdash Q x \&\& S$
and *sR*: *sound R* **and** *sS*: *sound S*
and *bQ*: $\bigwedge x. \text{bounded-by } 1 (Q x)$
shows $R \Vdash S$
proof(*rule le-funI*)
fix s
from *exhaust* **obtain** x **where** $P\text{-}xs: P x s = 1$ **by**(*blast*)
moreover {
hence $1 = P x s$ **by**(*simp*)
also from PQe **have** $P x s \leq Q x s$ **by**(*blast dest:le-funD*)
finally have $Q x s = 1$
using bQ **by**(*blast intro:antisym*)
}
moreover note $le\text{-}funD[OF\ framed[\text{where } x=x], \text{where } x=s]$
moreover from sR **have** $0 \leq R s$ **by**(*blast*)
moreover from sS **have** $0 \leq S s$ **by**(*blast*)
ultimately show $R s \leq S s$ **by**(*simp add:exp-conj-def*)
qed

lemma *unitary-bot*[*iff*]:
unitary ($\lambda s. 0::real$)
by(*auto*)

lemma *unitary-top*[*iff*]:
unitary ($\lambda s. 1::real$)
by(*auto*)

lemma *unitary-embed*[*iff*]:
unitary $\langle\langle P \rangle\rangle$
by(*auto*)

lemma *unitary-const*[*iff*]:
 $\llbracket 0 \leq c; c \leq 1 \rrbracket \implies \text{unitary } (\lambda s. c)$
by(*auto*)

lemma *unitary-mult*:
assumes $uA: \text{unitary } A$ **and** $uB: \text{unitary } B$
shows *unitary* ($\lambda s. A s * B s$)
proof(*intro unitaryI2 nnegI bounded-byI*)
fix s
from *assms* **have** $nnA: 0 \leq A s$ **and** $nnB: 0 \leq B s$ **by**(*auto*)
thus $0 \leq A s * B s$ **by**(*rule mult-nonneg-nonneg*)
from *assms* **have** $A s \leq 1$ **and** $B s \leq 1$ **by**(*auto*)
with nnB **have** $A s * B s \leq 1 * 1$ **by**(*intro mult-mono, auto*)
also have $\dots = 1$ **by**(*simp*)
finally show $A s * B s \leq 1$.

qed

lemma *exp-conj-unitary*:

$\llbracket \text{unitary } P; \text{unitary } Q \rrbracket \Longrightarrow \text{unitary } (P \&\& Q)$
by(*intro unitaryI2 nnegI2, auto*)

lemma *unitary-comp[simp]*:

$\text{unitary } P \Longrightarrow \text{unitary } (P \circ f)$
by(*intro unitaryI2 nnegI bounded-byI, auto simp:o-def*)

lemmas *unitary-intros* =

unitary-bot unitary-top unitary-embed unitary-mult exp-conj-unitary
unitary-comp unitary-const

lemmas *sound-intros* =

mult-sound div-sound const-sound sound-o sound-sum
tminus-sound sc-sound exp-conj-sound sum-sound

end

3.2 Expectation Transformers

theory *Transformers* **imports** *Expectations* **begin** **type-synonym** *'s trans* = *'s expect* \Rightarrow *'s expect*

Transformers are functions from expectations to expectations i.e. $(\text{'s} \Rightarrow \text{real}) \Rightarrow \text{'s} \Rightarrow \text{real}$.

The set of *healthy* transformers is the universe into which we place our semantic interpretation of pGCL programs. In its standard presentation, the healthiness condition for pGCL programs is *sublinearity*, for demonic programs, and *superlinearity* for angelic programs. We extract a minimal core property, consisting of monotonicity, feasibility and scaling to form our healthiness property, which holds across all programs. The additional components of sublinearity are broken out separately, and shown later. The two reasons for this are firstly to avoid the effort of establishing sub-(super-)linearity globally, and to allow us to define primitives whose sublinearity, and indeed healthiness, depend on context.

Consider again the automaton of [Figure 3.1](#). Here, the effect of executing the automaton from its initial state (a) until it reaches some final state (b or c) is to *transform* the expectation on final states (P), into one on initial states, giving the *expected* value of the function on termination. Here, the transformation is linear: $P_{\text{prior}}(a) = 0.7 * P_{\text{post}}(b) + 0.3 * P_{\text{post}}(c)$, but this need not be the case.

Consider the automaton of [Figure 3.2](#). Here, we have extended that of [Figure 3.1](#) with two additional states, d and e , and a pair of silent (unlabelled) transitions. From the initial state, e , this automaton is free to transition either to the original starting state (a), and thence behave exactly as the previous automaton did, or to d , which has the same set of available transitions, now with different probabilities.

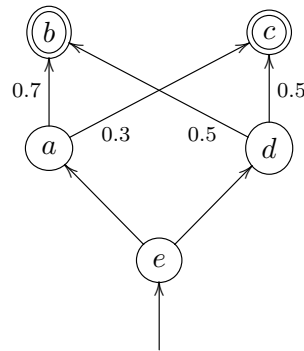


Figure 3.2: A nondeterministic-probabilistic automaton.

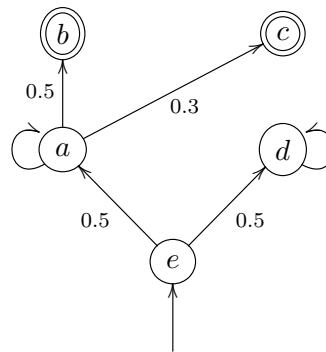


Figure 3.3: A diverging automaton.

Where previously we could state that the automaton would terminate in state b with probability 0.7 (and in c with probability 0.3), this now depends on the outcome of the *nondeterministic* transition from e to either a or d . The most we can now say is that we must reach b with probability *at least* 0.5 (the minimum from either a or d) and c with at least probability 0.3. Note that these probabilities do not sum to one (although the sum will still always be less than one). The associated expectation transformer is now *sub-linear*: $P_{\text{prior}}(e) = 0.5 * P_{\text{post}}(b) + 0.3 * P_{\text{post}}(c)$.

Finally, [Figure 3.3](#) shows the other way in which strict sublinearity arises: divergence. This automaton transitions with probability 0.5 to state d , from which it never escapes. Once there, the probability of reaching any terminating state is zero, and thus the probability of terminating from the initial state (e) is no higher than 0.5. If it instead takes the edge to state a , we again see a self loop, and thus in theory an infinite trace. In this case, however, every time the automaton reaches state a , with probability $0.5 + 0.3 = 0.8$, it transitions to a terminating state. An infinite trace of transitions $a \rightarrow a \rightarrow \dots$ thus has probability 0, and the automaton terminates with probability 1. We formalise such probabilistic termination argu-

ments in [Section 4.11](#).

Having reached a , the automaton will proceed to b with probability $0.5 * (1/(0.5 + 0.3)) = 0.625$, and to c with probability 0.375 . As a is in turn reached half the time, the final probability of ending in b is 0.3125 , and in c , 0.1875 , which sum to only 0.5 . The remaining probability is that the automaton diverges via d . We view nondeterminism and divergence demonically: we take the *least* probability of reaching a given final state, and use it to calculate the expectation. Thus for this automaton, $P_{\text{prior}}(e) = 0.3125 * P_{\text{post}}(b) + 0.1875 * P_{\text{post}}(c)$. The end result is the same as for nondeterminism: a sublinear transformation (the weights sum to less than one). The two outcomes are thus unified in the semantic interpretation, although as we will establish in [Section 4.6](#), the two have slightly different algebraic properties.

This pattern holds for all pGCL programs: probabilistic choices are always linear, while struct sublinearity is introduced both nondeterminism and divergence.

Healthiness, again, is the combination of three properties: feasibility, monotonicity and scaling. Feasibility requires that a transformer take non-negative expectations to non-negative expectations, and preserve bounds. Thus, starting with an expectation bounded between 0 and some bound, b , after applying any number of feasible transformers, the result will still be bounded between 0 and b . This closure property allows us to treat expectations almost as a complete lattice. Specifically, for any b , the set of expectations bounded by b is a complete lattice ($\perp = (\lambda s.0)$, $\top = (\lambda s.b)$), and is closed under the action of feasible transformers, including \sqcap and \sqcup , which are themselves feasible. We are thus able to define both least and greatest fixed points on this set, and thus give semantics to recursive programs built from feasible components.

3.2.1 Comparing Transformers

Transformers are compared pointwise, but only on *sound* expectations. From the preorder so generated, we define equivalence by antisymmetry, giving a partial order.

definition

$$\text{le-trans} :: 's \text{ trans} \Rightarrow 's \text{ trans} \Rightarrow \text{bool}$$

where

$$\text{le-trans } t \ u \equiv \forall P. \text{ sound } P \longrightarrow t \ P \leq u \ P$$

We also need to define relations restricted to *unitary* transformers, for the liberal (wlp) semantics.

definition

$$\text{le-utrans} :: 's \text{ trans} \Rightarrow 's \text{ trans} \Rightarrow \text{bool}$$

where

$$\text{le-utrans } t \ u \longleftrightarrow (\forall P. \text{ unitary } P \longrightarrow t \ P \leq u \ P)$$

lemma $\text{le-transI}[\text{intro}]$:

$\llbracket \bigwedge P. \text{sound } P \implies t P \leq u P \rrbracket \implies \text{le-trans } t u$
by(simp add:le-trans-def)

lemma *le-utransI*[intro]:
 $\llbracket \bigwedge P. \text{unitary } P \implies t P \leq u P \rrbracket \implies \text{le-utrans } t u$
by(simp add:le-utrans-def)

lemma *le-transD*[dest]:
 $\llbracket \text{le-trans } t u; \text{sound } P \rrbracket \implies t P \leq u P$
by(simp add:le-trans-def)

lemma *le-utransD*[dest]:
 $\llbracket \text{le-utrans } t u; \text{unitary } P \rrbracket \implies t P \leq u P$
by(simp add:le-utrans-def)

lemma *le-trans-trans*[trans]:
 $\llbracket \text{le-trans } x y; \text{le-trans } y z \rrbracket \implies \text{le-trans } x z$
unfolding *le-trans-def* **by**(blast dest:order-trans)

lemma *le-utrans-trans*[trans]:
 $\llbracket \text{le-utrans } x y; \text{le-utrans } y z \rrbracket \implies \text{le-utrans } x z$
unfolding *le-utrans-def* **by**(blast dest:order-trans)

lemma *le-trans-refl*[iff]:
 $\text{le-trans } x x$
by(simp add:le-trans-def)

lemma *le-utrans-refl*[iff]:
 $\text{le-utrans } x x$
by(simp add:le-utrans-def)

lemma *le-trans-le-utrans*[dest]:
 $\text{le-trans } t u \implies \text{le-utrans } t u$
unfolding *le-trans-def* *le-utrans-def* **by**(auto)

definition
 $l\text{-trans} :: 's \text{ trans} \Rightarrow 's \text{ trans} \Rightarrow \text{bool}$
where
 $l\text{-trans } t u \iff \text{le-trans } t u \wedge \neg \text{le-trans } u t$

Transformer equivalence is induced by comparison:

definition
 $\text{equiv-trans} :: 's \text{ trans} \Rightarrow 's \text{ trans} \Rightarrow \text{bool}$
where
 $\text{equiv-trans } t u \iff \text{le-trans } t u \wedge \text{le-trans } u t$

definition
 $\text{equiv-utrans} :: 's \text{ trans} \Rightarrow 's \text{ trans} \Rightarrow \text{bool}$
where

$equiv\text{-}utrans\ t\ u \iff le\text{-}utrans\ t\ u \wedge le\text{-}utrans\ u\ t$

lemma *equiv-transI[intro]*:

$\llbracket \bigwedge P. sound\ P \implies t\ P = u\ P \rrbracket \implies equiv\text{-}trans\ t\ u$
unfolding *equiv-trans-def* **by**(*force*)

lemma *equiv-utransI[intro]*:

$\llbracket \bigwedge P. sound\ P \implies t\ P = u\ P \rrbracket \implies equiv\text{-}utrans\ t\ u$
unfolding *equiv-utrans-def* **by**(*force*)

lemma *equiv-transD[dest]*:

$\llbracket equiv\text{-}trans\ t\ u; sound\ P \rrbracket \implies t\ P = u\ P$
unfolding *equiv-trans-def* **by**(*blast intro:antisym*)

lemma *equiv-utransD[dest]*:

$\llbracket equiv\text{-}utrans\ t\ u; unitary\ P \rrbracket \implies t\ P = u\ P$
unfolding *equiv-utrans-def* **by**(*blast intro:antisym*)

lemma *equiv-trans-refl[iff]*:

$equiv\text{-}trans\ t\ t$
by(*blast*)

lemma *equiv-utrans-refl[iff]*:

$equiv\text{-}utrans\ t\ t$
by(*blast*)

lemma *le-trans-antisym*:

$\llbracket le\text{-}trans\ x\ y; le\text{-}trans\ y\ x \rrbracket \implies equiv\text{-}trans\ x\ y$
unfolding *equiv-trans-def* **by**(*simp*)

lemma *le-utrans-antisym*:

$\llbracket le\text{-}utrans\ x\ y; le\text{-}utrans\ y\ x \rrbracket \implies equiv\text{-}utrans\ x\ y$
unfolding *equiv-utrans-def* **by**(*simp*)

lemma *equiv-trans-comm[ac-simps]*:

$equiv\text{-}trans\ t\ u \iff equiv\text{-}trans\ u\ t$
unfolding *equiv-trans-def* **by**(*blast*)

lemma *equiv-utrans-comm[ac-simps]*:

$equiv\text{-}utrans\ t\ u \iff equiv\text{-}utrans\ u\ t$
unfolding *equiv-utrans-def* **by**(*blast*)

lemma *equiv-imp-le[intro]*:

$equiv\text{-}trans\ t\ u \implies le\text{-}trans\ t\ u$
unfolding *equiv-trans-def* **by**(*clarify*)

lemma *equivu-imp-le[intro]*:

$equiv\text{-}utrans\ t\ u \implies le\text{-}utrans\ t\ u$
unfolding *equiv-utrans-def* **by**(*clarify*)

lemma *equiv-imp-le-alt*:

$equiv-trans\ t\ u \implies le-trans\ u\ t$

by(*force simp:ac-simps*)

lemma *equiv-uimp-le-alt*:

$equiv-utrans\ t\ u \implies le-utrans\ u\ t$

by(*force simp:ac-simps*)

lemma *le-trans-equiv-rsp[simp]*:

$equiv-trans\ t\ u \implies le-trans\ t\ v \longleftrightarrow le-trans\ u\ v$

unfolding *equiv-trans-def* **by**(*blast intro:le-trans-trans*)

lemma *le-utrans-equiv-rsp[simp]*:

$equiv-utrans\ t\ u \implies le-utrans\ t\ v \longleftrightarrow le-utrans\ u\ v$

unfolding *equiv-utrans-def* **by**(*blast intro:le-utrans-trans*)

lemma *equiv-trans-le-trans[trans]*:

$\llbracket equiv-trans\ t\ u; le-trans\ u\ v \rrbracket \implies le-trans\ t\ v$

by(*simp*)

lemma *equiv-utrans-le-utrans[trans]*:

$\llbracket equiv-utrans\ t\ u; le-utrans\ u\ v \rrbracket \implies le-utrans\ t\ v$

by(*simp*)

lemma *le-trans-equiv-rsp-right[simp]*:

$equiv-trans\ t\ u \implies le-trans\ v\ t \longleftrightarrow le-trans\ v\ u$

unfolding *equiv-trans-def* **by**(*blast intro:le-trans-trans*)

lemma *le-utrans-equiv-rsp-right[simp]*:

$equiv-utrans\ t\ u \implies le-utrans\ v\ t \longleftrightarrow le-utrans\ v\ u$

unfolding *equiv-utrans-def* **by**(*blast intro:le-utrans-trans*)

lemma *le-trans-equiv-trans[trans]*:

$\llbracket le-trans\ t\ u; equiv-trans\ u\ v \rrbracket \implies le-trans\ t\ v$

by(*simp*)

lemma *le-utrans-equiv-utrans[trans]*:

$\llbracket le-utrans\ t\ u; equiv-utrans\ u\ v \rrbracket \implies le-utrans\ t\ v$

by(*simp*)

lemma *equiv-trans-trans[trans]*:

assumes *xy*: *equiv-trans* *x* *y*

and *yz*: *equiv-trans* *y* *z*

shows *equiv-trans* *x* *z*

proof(*rule le-trans-antisym*)

from *xy* **have** *le-trans* *x* *y* **by**(*blast*)

also from *yz* **have** *le-trans* *y* *z* **by**(*blast*)

finally show *le-trans* *x* *z* .

from yz **have** $le\text{-}trans\ z\ y$ **by**(*force simp:ac-simps*)
also from xy **have** $le\text{-}trans\ y\ x$ **by**(*force simp:ac-simps*)
finally show $le\text{-}trans\ z\ x$.
qed

lemma *equiv-utrans-trans*[*trans*]:
assumes xy : *equiv-utrans* $x\ y$
and yz : *equiv-utrans* $y\ z$
shows *equiv-utrans* $x\ z$
proof(*rule le-utrans-antisym*)
from xy **have** $le\text{-}utrans\ x\ y$ **by**(*blast*)
also from yz **have** $le\text{-}utrans\ y\ z$ **by**(*blast*)
finally show $le\text{-}utrans\ x\ z$.
from yz **have** $le\text{-}utrans\ z\ y$ **by**(*force simp:ac-simps*)
also from xy **have** $le\text{-}utrans\ y\ x$ **by**(*force simp:ac-simps*)
finally show $le\text{-}utrans\ z\ x$.
qed

lemma *equiv-trans-equiv-utrans*[*dest*]:
 $equiv\text{-}trans\ t\ u \implies equiv\text{-}utrans\ t\ u$
by(*auto*)

3.2.2 Healthy Transformers

Feasibility

definition *feasible* :: $((a \Rightarrow real) \Rightarrow (a \Rightarrow real)) \Rightarrow bool$
where $feasible\ t \iff (\forall P\ b.\ bounded\text{-}by\ b\ P \wedge nneg\ P \implies$
 $bounded\text{-}by\ b\ (t\ P) \wedge nneg\ (t\ P))$

A *feasible* transformer preserves non-negativity, and bounds. A *feasible* transformer always takes its argument ‘closer to 0’ (or leaves it where it is). Note that any particular value of the expectation may increase, but no element of the new expectation may exceed any bound on the old. This is thus a relatively weak condition.

lemma *feasibleI*[*intro*]:
 $\llbracket \bigwedge b\ P.\ \llbracket bounded\text{-}by\ b\ P;\ nneg\ P \rrbracket \implies bounded\text{-}by\ b\ (t\ P);$
 $\bigwedge b\ P.\ \llbracket bounded\text{-}by\ b\ P;\ nneg\ P \rrbracket \implies nneg\ (t\ P) \rrbracket \implies feasible\ t$
by(*force simp:feasible-def*)

lemma *feasible-boundedD*[*dest*]:
 $\llbracket feasible\ t;\ bounded\text{-}by\ b\ P;\ nneg\ P \rrbracket \implies bounded\text{-}by\ b\ (t\ P)$
by(*simp add:feasible-def*)

lemma *feasible-nnegD*[*dest*]:
 $\llbracket feasible\ t;\ bounded\text{-}by\ b\ P;\ nneg\ P \rrbracket \implies nneg\ (t\ P)$
by(*simp add:feasible-def*)

lemma *feasible-sound*[*dest*]:

$\llbracket \text{feasible } t; \text{ sound } P \rrbracket \implies \text{sound } (t P)$
by(rule soundI, unfold sound-def, (blast)+)

lemma *feasible-pr-0[simp]*:
fixes $t::('s \Rightarrow \text{real}) \Rightarrow 's \Rightarrow \text{real}$
assumes $ft: \text{feasible } t$
shows $t (\lambda x. 0) = (\lambda x. 0)$
proof(rule ext, rule antisym)
fix s

have *bounded-by 0* $(\lambda::'s. 0::\text{real})$ **by**(blast)
with ft **have** *bounded-by 0* $(t (\lambda-. 0))$ **by**(blast)
thus $t (\lambda-. 0) s \leq 0$ **by**(blast)

have *nneg* $(\lambda::'s. 0::\text{real})$ **by**(blast)
with ft **have** *nneg* $(t (\lambda-. 0))$ **by**(blast)
thus $0 \leq t (\lambda-. 0) s$ **by**(blast)
qed

lemma *feasible-id*:
feasible $(\lambda x. x)$
unfolding *feasible-def* **by**(blast)

lemma *feasible-bounded-by[dest]*:
 $\llbracket \text{feasible } t; \text{ sound } P; \text{ bounded-by } b P \rrbracket \implies \text{bounded-by } b (t P)$
by(auto)

lemma *feasible-fixes-top*:
feasible $t \implies t (\lambda s. 1) \leq (\lambda s. (1::\text{real}))$
by(drule bounded-byD2[OF *feasible-bounded-by*], auto)

lemma *feasible-fixes-bot*:
assumes $ft: \text{feasible } t$
shows $t (\lambda s. 0) = (\lambda s. 0)$
proof(rule antisym)
have $sb: \text{sound } (\lambda s. 0)$ **by**(auto)
with ft **show** $(\lambda s. 0) \leq t (\lambda s. 0)$ **by**(auto)
thm *bound-of-const*
from sb **have** *bounded-by* $(\text{bound-of } (\lambda s. 0::\text{real})) (\lambda s. 0)$ **by**(auto)
hence *bounded-by 0* $(\lambda s. 0::\text{real})$ **by**(simp add:bound-of-const)
with ft **have** *bounded-by 0* $(t (\lambda s. 0))$ **by**(auto)
thus $t (\lambda s. 0) \leq (\lambda s. 0)$ **by**(auto)
qed

lemma *feasible-unitaryD[dest]*:
assumes $ft: \text{feasible } t$ **and** $uP: \text{unitary } P$
shows *unitary* $(t P)$
proof(rule unitaryI)
from uP **have** *sound P* **by**(auto)

```

with ft show sound (t P) by(auto)
from assms show bounded-by 1 (t P) by(auto)
qed

```

Monotonicity

definition

$$\text{mono-trans} :: (('s \Rightarrow \text{real}) \Rightarrow ('s \Rightarrow \text{real})) \Rightarrow \text{bool}$$

where

$$\text{mono-trans } t \equiv \forall P Q. (\text{sound } P \wedge \text{sound } Q \wedge P \leq Q) \longrightarrow t P \leq t Q$$

Monotonicity allows us to compose transformers, and thus model sequential computation. Recall the definition of predicate entailment (Section 3.1.6) as less-than-or-equal. The statement $Q \Vdash t R$ means that Q is everywhere below $t R$. For standard expectations (Section 3.1.5), this simply means that Q implies $t R$, the *weakest precondition* of R under t .

Given another, monotonic, transformer u , we have that $u Q \Vdash u (t R)$, or that the weakest precondition of Q under u entails that of R under the composition $u \circ t$. If we additionally know that $P \Vdash u Q$, then by transitivity we have $P \Vdash u (t R)$. We thus derive a probabilistic form of the standard rule for sequential composition: $\llbracket \text{mono-trans } t; P \Vdash u Q; Q \Vdash t R \rrbracket \Longrightarrow P \Vdash u (t R)$.

lemma mono-transI[*intro*]:

$$\llbracket \bigwedge P Q. \llbracket \text{sound } P; \text{sound } Q; P \leq Q \rrbracket \Longrightarrow t P \leq t Q \rrbracket \Longrightarrow \text{mono-trans } t$$

by(simp add:mono-trans-def)

lemma mono-transD[*dest*]:

$$\llbracket \text{mono-trans } t; \text{sound } P; \text{sound } Q; P \leq Q \rrbracket \Longrightarrow t P \leq t Q$$

by(simp add:mono-trans-def)

Scaling

A healthy transformer commutes with scaling by a non-negative constant.

definition

$$\text{scaling} :: (('s \Rightarrow \text{real}) \Rightarrow ('s \Rightarrow \text{real})) \Rightarrow \text{bool}$$

where

$$\text{scaling } t \equiv \forall P c x. \text{sound } P \wedge 0 \leq c \longrightarrow c * t P x = t (\lambda x. c * P x) x$$

The *scaling* and feasibility properties together allow us to treat transformers as a complete lattice, when operating on bounded expectations. The action of a transformer on such a bounded expectation is completely determined by its action on *unitary* expectations (those bounded by 1): $t P s = \text{bound-of } P * t (\lambda s. P s / \text{bound-of } P) s$. Feasibility in turn ensures that the lattice of unitary expectations is closed under the action of a healthy transformer. We take advantage of this fact in Section 3.3, in order to define the fixed points of healthy transformers.

lemma scalingI[*intro*]:

$$\llbracket \bigwedge P c x. \llbracket \text{sound } P; 0 \leq c \rrbracket \Longrightarrow c * t P x = t (\lambda x. c * P x) x \rrbracket \Longrightarrow \text{scaling } t$$

by(*simp add:scaling-def*)

lemma *scalingD*[*dest*]:

$\llbracket \text{scaling } t; \text{sound } P; 0 \leq c \rrbracket \implies c * t P x = t (\lambda x. c * P x) x$

by(*simp add:scaling-def*)

lemma *right-scalingD*:

assumes *st*: *scaling t*

and *sP*: *sound P*

and *nnc*: $0 \leq c$

shows $t P s * c = t (\lambda s. P s * c) s$

proof –

have $t P s * c = c * t P s$ **by**(*simp add:algebra-simps*)

also from *assms* **have** $\dots = t (\lambda s. c * P s) s$ **by**(*rule scalingD*)

also have $\dots = t (\lambda s. P s * c) s$ **by**(*simp add:algebra-simps*)

finally show *?thesis* .

qed

Healthiness

Healthy transformers are feasible and monotonic, and respect scaling

definition

healthy :: $((s \Rightarrow \text{real}) \Rightarrow (s \Rightarrow \text{real})) \Rightarrow \text{bool}$

where

healthy $t \iff \text{feasible } t \wedge \text{mono-trans } t \wedge \text{scaling } t$

lemma *healthyI*[*intro*]:

$\llbracket \text{feasible } t; \text{mono-trans } t; \text{scaling } t \rrbracket \implies \text{healthy } t$

by(*simp add:healthy-def*)

lemmas *healthy-parts* = *healthyI*[*OF feasibleI mono-transI scalingI*]

lemma *healthy-monoD*[*dest*]:

healthy t $\implies \text{mono-trans } t$

by(*simp add:healthy-def*)

lemmas *healthy-monoD2* = *mono-transD*[*OF healthy-monoD*]

lemma *healthy-feasibleD*[*dest*]:

healthy t $\implies \text{feasible } t$

by(*simp add:healthy-def*)

lemma *healthy-scalingD*[*dest*]:

healthy t $\implies \text{scaling } t$

by(*simp add:healthy-def*)

lemma *healthy-bounded-byD*[*intro*]:

$\llbracket \text{healthy } t; \text{bounded-by } b P; \text{nneg } P \rrbracket \implies \text{bounded-by } b (t P)$

by(*blast*)

lemma *healthy-bounded-byD2*:

$\llbracket \text{healthy } t; \text{ bounded-by } b \ P; \text{ sound } P \rrbracket \Longrightarrow \text{bounded-by } b \ (t \ P)$
by(*blast*)

lemma *healthy-boundedD[dest,simp]*:

$\llbracket \text{healthy } t; \text{ sound } P \rrbracket \Longrightarrow \text{bounded} \ (t \ P)$
by(*blast*)

lemma *healthy-nnegD[dest,simp]*:

$\llbracket \text{healthy } t; \text{ sound } P \rrbracket \Longrightarrow \text{nneg} \ (t \ P)$
by(*blast intro!:feasible-nnegD*)

lemma *healthy-nnegD2[dest,simp]*:

$\llbracket \text{healthy } t; \text{ bounded-by } b \ P; \text{ nneg } P \rrbracket \Longrightarrow \text{nneg} \ (t \ P)$
by(*blast*)

lemma *healthy-sound[intro]*:

$\llbracket \text{healthy } t; \text{ sound } P \rrbracket \Longrightarrow \text{sound} \ (t \ P)$
by(*rule soundI, blast, blast intro!:feasible-nnegD*)

lemma *healthy-unitary[intro]*:

$\llbracket \text{healthy } t; \text{ unitary } P \rrbracket \Longrightarrow \text{unitary} \ (t \ P)$
by(*blast intro!:unitaryI dest:unitary-bound healthy-bounded-byD*)

lemma *healthy-id[simp,intro!]*:

healthy id
by(*simp add:healthyI feasibleI mono-transI scalingI*)

lemmas *healthy-fixes-bot = feasible-fixes-bot[OF healthy-feasibleD]*

Some additional results on *le-trans*, specific to *healthy* transformers.

lemma *le-trans-bot[intro,simp]*:

healthy t \Longrightarrow *le-trans* $(\lambda P \ s. \ 0) \ t$
by(*blast intro!:le-funI*)

lemma *le-trans-top[intro,simp]*:

healthy t \Longrightarrow *le-trans* $t \ (\lambda P \ s. \ \text{bound-of } P)$
by(*blast intro!:le-transI[OF le-funI]*)

lemma *healthy-pr-bot[simp]*:

healthy t \Longrightarrow $t \ (\lambda s. \ 0) = (\lambda s. \ 0)$
by(*blast intro!:feasible-pr-0*)

The first significant result is that healthiness is preserved by equivalence:

lemma *healthy-equivI*:

fixes $t::('s \Rightarrow \text{real}) \Rightarrow 's \Rightarrow \text{real}$ **and** u
assumes *equiv*: *equiv-trans* $t \ u$
and *healthy*: *healthy* t

shows *healthy* u
proof
have $le-t-u$: $le-trans\ t\ u$ **by**(*blast intro:equiv*)
have $le-u-t$: $le-trans\ u\ t$ **by**(*simp add:equiv-imp-le ac-simps equiv*)
from *equiv* **have** $eq-u-t$: $equiv-trans\ u\ t$ **by**(*simp add:ac-simps*)

show *feasible* u
proof
fix b **and** $P::'s \Rightarrow real$
assume bP : *bounded-by* $b\ P$ **and** nP : *nneg* P
hence sP : *sound* P **by**(*blast*)
with *healthy* **have** $\bigwedge s. 0 \leq t\ P\ s$ **by**(*blast*)
also from sP **and** $le-t-u$ **have** $\bigwedge s. \dots s \leq u\ P\ s$ **by**(*blast*)
finally show *nneg* ($u\ P$) **by**(*blast*)

from sP **and** $le-u-t$ **have** $\bigwedge s. u\ P\ s \leq t\ P\ s$ **by**(*blast*)
also from *healthy* **and** sP **and** bP **have** $\bigwedge s. t\ P\ s \leq b$ **by**(*blast*)
finally show *bounded-by* $b\ (u\ P)$ **by**(*blast*)
qed

show *mono-trans* u
proof
fix $P::'s \Rightarrow real$ **and** $Q::'s \Rightarrow real$
assume sP : *sound* P **and** sQ : *sound* Q
and le : $P \Vdash Q$
from sP **and** $le-u-t$ **have** $u\ P \Vdash t\ P$ **by**(*blast*)
also from sP **and** sQ **and** le **and** *healthy* **have** $t\ P \Vdash t\ Q$ **by**(*blast*)
also from sQ **and** $le-t-u$ **have** $t\ Q \Vdash u\ Q$ **by**(*blast*)
finally show $u\ P \Vdash u\ Q$.
qed

show *scaling* u
proof
fix $P::'s \Rightarrow real$ **and** $c::real$ **and** $x::'s$
assume *sound*: *sound* P
and pos : $0 \leq c$

hence *bounded-by* ($c * bound-of\ P$) ($\lambda x. c * P\ x$)
by(*blast intro!:mult-left-mono dest!:less-imp-le*)
hence *sc-bounded*: *bounded* ($\lambda x. c * P\ x$)
by(*blast*)
moreover from *sound* **and** pos **have** *sc-nneg*: *nneg* ($\lambda x. c * P\ x$)
by(*blast intro!:mult-nonneg-nonneg less-imp-le*)
ultimately have *sc-sound*: *sound* ($\lambda x. c * P\ x$) **by**(*blast*)

show $c * u\ P\ x = u\ (\lambda x. c * P\ x)\ x$
proof –
from *sound* **have** $c * u\ P\ x = c * t\ P\ x$
by(*simp add:equiv-transD[OF eq-u-t]*)

also have $\dots = t (\lambda x. c * P x) x$
using *healthy and sound and pos*
by(*blast intro: scalingD*)

also from *sc-sound and equiv* **have** $\dots = u (\lambda x. c * P x) x$
by(*blast intro:fun-cong*)

finally show *?thesis* .

qed

qed

qed

lemma *healthy-equiv*:

equiv-trans t u \implies healthy t \longleftrightarrow healthy u
by(*rule iffI, rule healthy-equivI, assumption+, simp add:healthy-equivI ac-simps*)

lemma *healthy-scale*:

fixes *t::('s \implies real) \implies 's \implies real*
assumes *ht: healthy t and nc: $0 \leq c$ and bc: $c \leq 1$*
shows *healthy ($\lambda P s. c * t P s$)*

proof

show *feasible ($\lambda P s. c * t P s$)*

proof

fix *b and P::'s \implies real*
assume *nnP: nneg P and bP: bounded-by b P*

from *ht nnP bP* **have** $\bigwedge s. t P s \leq b$ **by**(*blast*)
with *nc* **have** $\bigwedge s. c * t P s \leq c * b$ **by**(*blast intro:mult-left-mono*)
also {
from *nnP and bP* **have** $0 \leq b$ **by**(*auto*)
with *bc* **have** $c * b \leq 1 * b$ **by**(*blast intro:mult-right-mono*)
hence $c * b \leq b$ **by**(*simp*)
}

finally show *bounded-by b ($\lambda s. c * t P s$)* **by**(*blast*)

from *ht nnP bP* **have** $\bigwedge s. 0 \leq t P s$ **by**(*blast*)
with *nc* **have** $\bigwedge s. 0 \leq c * t P s$ **by**(*rule mult-nonneg-nonneg*)
thus *nneg ($\lambda s. c * t P s$)* **by**(*blast*)

qed

show *mono-trans ($\lambda P s. c * t P s$)*

proof

fix *P::'s \implies real and Q*
assume *sP: sound P and sQ: sound Q and le: P \Vdash Q*
with *ht* **have** $\bigwedge s. t P s \leq t Q s$ **by**(*auto intro:le-funD*)
with *nc* **have** $\bigwedge s. c * t P s \leq c * t Q s$
by(*blast intro:mult-left-mono*)
thus $\lambda s. c * t P s \Vdash \lambda s. c * t Q s$ **by**(*blast*)

qed
from *ht show scaling* ($\lambda P s. c * t P s$)
by(*auto simp:scalingD healthy-scalingD ht*)
qed

lemma *healthy-top*[*iff*]:
healthy ($\lambda P s. \text{bound-of } P$)
by(*auto intro!:healthy-parts*)

lemma *healthy-bot*[*iff*]:
healthy ($\lambda P s. 0$)
by(*auto intro!:healthy-parts*)

This weaker healthiness condition is for the liberal (wlp) semantics. We only insist that the transformer preserves *unitarity* (bounded by 1), and drop scaling (it is unnecessary in establishing the lattice structure here, unlike for the strict semantics).

definition

nearly-healthy :: $((s \Rightarrow \text{real}) \Rightarrow (s \Rightarrow \text{real})) \Rightarrow \text{bool}$
where
nearly-healthy $t \iff (\forall P. \text{unitary } P \longrightarrow \text{unitary } (t P)) \wedge$
 $(\forall P Q. \text{unitary } P \longrightarrow \text{unitary } Q \longrightarrow P \Vdash Q \longrightarrow t P \Vdash t Q)$

lemma *nearly-healthyI*[*intro*]:
 $\llbracket \bigwedge P. \text{unitary } P \Longrightarrow \text{unitary } (t P);$
 $\bigwedge P Q. \llbracket \text{unitary } P; \text{unitary } Q; P \Vdash Q \rrbracket \Longrightarrow t P \Vdash t Q \rrbracket \Longrightarrow \text{nearly-healthy } t$
by(*simp add:nearly-healthy-def*)

lemma *nearly-healthy-monoD*[*dest*]:
 $\llbracket \text{nearly-healthy } t; P \Vdash Q; \text{unitary } P; \text{unitary } Q \rrbracket \Longrightarrow t P \Vdash t Q$
by(*simp add:nearly-healthy-def*)

lemma *nearly-healthy-unitaryD*[*dest*]:
 $\llbracket \text{nearly-healthy } t; \text{unitary } P \rrbracket \Longrightarrow \text{unitary } (t P)$
by(*simp add:nearly-healthy-def*)

lemma *healthy-nearly-healthy*[*dest*]:
assumes *ht: healthy t*
shows *nearly-healthy t*
by(*intro nearly-healthyI, auto intro:mono-transD[OF healthy-monoD, OF ht] ht*)

lemmas *nearly-healthy-id*[*iff*] =
healthy-nearly-healthy[*OF healthy-id, unfolded id-def*]

3.2.3 Sublinearity

As already mentioned, the core healthiness property (aside from feasibility and continuity) for transformers is *sublinearity*: The transformation of a quasi-linear combination of sound expectations is greater than the same combination applied

to the transformation of the expectations themselves. The term $x \ominus y$ represents *truncated subtraction* i.e. $\max(x - y, 0)$ (see [Section 4.13.1](#)).

definition *sublinear* ::

$((s \Rightarrow \text{real}) \Rightarrow (s \Rightarrow \text{real})) \Rightarrow \text{bool}$

where

$\text{sublinear } t \iff (\forall a b c P Q s. (\text{sound } P \wedge \text{sound } Q \wedge 0 \leq a \wedge 0 \leq b \wedge 0 \leq c) \longrightarrow$
 $a * t P s + b * t Q s \ominus c$
 $\leq t (\lambda s'. a * P s' + b * Q s' \ominus c) s)$

lemma *sublinearI[intro]*:

$\llbracket \bigwedge a b c P Q s. \llbracket \text{sound } P; \text{sound } Q; 0 \leq a; 0 \leq b; 0 \leq c \rrbracket \implies$
 $a * t P s + b * t Q s \ominus c \leq$
 $t (\lambda s'. a * P s' + b * Q s' \ominus c) s \rrbracket \implies \text{sublinear } t$

by(*simp add:sublinear-def*)

lemma *sublinearD[dest]*:

$\llbracket \text{sublinear } t; \text{sound } P; \text{sound } Q; 0 \leq a; 0 \leq b; 0 \leq c \rrbracket \implies$
 $a * t P s + b * t Q s \ominus c \leq$
 $t (\lambda s'. a * P s' + b * Q s' \ominus c) s$

by(*simp add:sublinear-def*)

It is easier to see the relevance of sublinearity by breaking it into several component properties, as in the following sections.

Sub-additivity

definition *sub-add* ::

$((s \Rightarrow \text{real}) \Rightarrow (s \Rightarrow \text{real})) \Rightarrow \text{bool}$

where

$\text{sub-add } t \iff (\forall P Q s. (\text{sound } P \wedge \text{sound } Q) \longrightarrow$
 $t P s + t Q s \leq t (\lambda s'. P s' + Q s') s)$

Sub-additivity, together with scaling ([Section 3.2.2](#)) gives the *linear* portion of sublinearity. Together, these two properties are equivalent to *convexity*, as [Figure 3.4](#) illustrates by analogy.

Here P is an affine function (expectation) $\text{real} \Rightarrow \text{real}$, restricted to some finite interval. In practice the state space (the left-hand type) is typically discrete and multi-dimensional, but on the reals we have a convenient geometrical intuition. The lines tP and uP represent the effect of two healthy transformers (again affine). Neither monotonicity nor scaling are represented, but both are feasible: Both lines are bounded above by the greatest value of P .

The curve Q is the pointwise minimum of tP and tQ , written $tP \sqcap tQ$. This is, not coincidentally, the syntax for a binary nondeterministic choice in pGCL: The probability that some property is established by the choice between programs a and b cannot be guaranteed to be any higher than either the probability under a , or that under b .

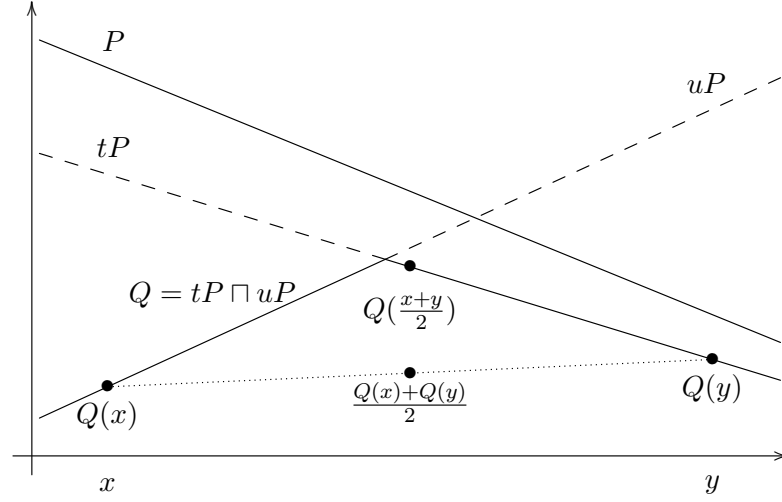


Figure 3.4: A graphical depiction of sub-additivity as convexity.

The original curve, P , is trivially convex—it is linear. Also, both t and u , and the operator \sqcap preserve convexity. A probabilistic choice will also preserve it. The preservation of convexity is a property of sub-additive transformers that respect scaling. Note the form of the definition of convexity:

$$\forall x, y. \frac{Q(x) + Q(y)}{2} \leq Q\left(\frac{x+y}{2}\right)$$

Were we to replace Q by some sub-additive transformer v , and x and y by expectations R and S , the equivalent expression:

$$\frac{vR + vS}{2} \leq v\left(\frac{R + S}{2}\right)$$

Can be rewritten, using scaling, to:

$$\frac{1}{2}(vR + vS) \leq \frac{1}{2}v(R + S)$$

Which holds everywhere exactly when v is sub-additive i.e.:

$$vR + vS \leq v(R + S)$$

lemma *sub-addI[intro]*:

$$\begin{aligned} & \llbracket \bigwedge P Q s. \llbracket \text{sound } P; \text{sound } Q \rrbracket \implies \\ & \quad t P s + t Q s \leq t (\lambda s'. P s' + Q s') s \rrbracket \implies \text{sub-add } t \\ & \text{by}(\text{simp add:sub-add-def}) \end{aligned}$$

lemma *sub-addI2*:

$$\llbracket \wedge P Q. \llbracket \text{sound } P; \text{sound } Q \rrbracket \implies \lambda s. t P s + t Q s \Vdash t (\lambda s. P s + Q s) \rrbracket \implies$$
sub-add t
by(*auto*)

lemma *sub-addD[dest]*:

$$\llbracket \text{sub-add } t; \text{sound } P; \text{sound } Q \rrbracket \implies t P s + t Q s \leq t (\lambda s'. P s' + Q s') s$$
by(*simp add:sub-add-def*)

lemma *equiv-sub-add*:

fixes $t::('s \Rightarrow \text{real}) \Rightarrow 's \Rightarrow \text{real}$

assumes $eq: \text{equiv-trans } t \ u$

and $sa: \text{sub-add } t$

shows $\text{sub-add } u$

proof

fix $P::'s \Rightarrow \text{real}$ **and** $Q::'s \Rightarrow \text{real}$ **and** $s::'s$

assume $sP: \text{sound } P$ **and** $sQ: \text{sound } Q$

with eq **have** $u P s + u Q s = t P s + t Q s$

by(*simp add:equiv-transD*)

also from $sP \ sQ \ sa$ **have** $t P s + t Q s \leq t (\lambda s. P s + Q s) s$

by(*auto*)

also {

from $sP \ sQ$ **have** $\text{sound } (\lambda s. P s + Q s)$ **by**(*auto*)

with eq **have** $t (\lambda s. P s + Q s) s = u (\lambda s. P s + Q s) s$

by(*simp add:equiv-transD*)

}

finally show $u P s + u Q s \leq u (\lambda s. P s + Q s) s$.

qed

Sublinearity and feasibility imply sub-additivity.

lemma *sublinear-subadd*:

fixes $t::('s \Rightarrow \text{real}) \Rightarrow 's \Rightarrow \text{real}$

assumes $slt: \text{sublinear } t$

and $ft: \text{feasible } t$

shows $\text{sub-add } t$

proof

fix $P::'s \Rightarrow \text{real}$ **and** $Q::'s \Rightarrow \text{real}$ **and** $s::'s$

assume $sP: \text{sound } P$ **and** $sQ: \text{sound } Q$

with ft **have** $\text{sound } (t P)$ $\text{sound } (t Q)$ **by**(*auto*)

hence $0 \leq t P s$ **and** $0 \leq t Q s$ **by**(*auto*)

hence $0 \leq t P s + t Q s$ **by**(*auto*)

hence $\dots = \dots \ominus 0$ **by**(*simp*)

also from $sP \ sQ$

have $\dots \leq t (\lambda s. P s + Q s \ominus 0) s$

by(*rule sublinearD[OF slt, where a=1 and b=1 and c=0, simplified]*)

also {
from $sP sQ$ **have** $\bigwedge s. 0 \leq P s$ **and** $\bigwedge s. 0 \leq Q s$ **by**(*auto*)
hence $\bigwedge s. 0 \leq P s + Q s$ **by**(*auto*)
hence $t (\lambda s. P s + Q s \ominus 0) s = t (\lambda s. P s + Q s) s$
by(*simp*)
}

finally show $t P s + t Q s \leq t (\lambda s. P s + Q s) s$.
qed

A few properties following from sub-additivity:

lemma *standard-negate*:

assumes *ht*: *healthy t*
and *sat*: *sub-add t*
shows $t \ll P \gg s + t \ll \mathcal{N} P \gg s \leq I$

proof –

from *sat* **have** $t \ll P \gg s + t \ll \mathcal{N} P \gg s \leq t (\lambda s. \ll P \gg s + \ll \mathcal{N} P \gg s) s$ **by**(*auto*)

also have $\dots = t (\lambda s. I) s$ **by**(*simp add:negate-embed*)

also {

from *ht* **have** *bounded-by I* ($t (\lambda s. I)$) **by**(*auto*)

hence $t (\lambda s. I) s \leq I$ **by**(*auto*)

}

finally show *?thesis*.

qed

lemma *sub-add-sum*:

fixes *t*::*'s trans* **and** *S*::*'a set*

assumes *sat*: *sub-add t*

and *ht*: *healthy t*

and *sP*: $\bigwedge x. \text{sound } (P x)$

shows $(\lambda x. \sum y \in S. t (P y) x) \leq t (\lambda x. \sum y \in S. P y x)$

proof(*cases infinite S, simp-all add:ht*)

assume *fS*: *finite S*

show *?thesis*

proof(*rule finite-induct[OF fS le-funI le-funI], simp-all*)

fix *s*::*'s*

from *ht* **have** *sound* ($t (\lambda s. 0)$) **by**(*auto*)

thus $0 \leq t (\lambda s. 0) s$ **by**(*auto*)

fix *F*::*'a set* **and** *x*::*'a*

assume *IH*: $\lambda a. \sum y \in F. t (P y) a \Vdash t (\lambda x. \sum y \in F. P y x)$

hence $t (P x) s + (\sum y \in F. t (P y) s) \leq$

$t (P x) s + t (\lambda x. \sum y \in F. P y x) s$

by(*auto intro:add-left-mono*)

also from *sat sP*

have $\dots \leq t (\lambda xa. P x xa + (\sum y \in F. P y xa)) s$

by(*auto intro!:sub-addD[OF sat] sum-sound*)

finally

show $t (P x) s + (\sum y \in F. t (P y) s) \leq$

$$t (\lambda x a. P x xa + (\sum y \in F. P y xa)) s .$$

qed**qed****lemma** *sub-add-guard-split*:**fixes** $t :: 's :: \text{finite trans}$ **and** $P :: 's \text{ expect}$ **and** $s :: 's$ **assumes** *sat*: *sub-add t***and** *ht*: *healthy t***and** *sP*: *sound P***shows** $(\sum y \in \{s. G s\}. P y * t \ll \lambda z. z = y \gg s) +$
 $(\sum y \in \{s. \neg G s\}. P y * t \ll \lambda z. z = y \gg s) \leq t P s$ **proof** –**have** $\{s. G s\} \cap \{s. \neg G s\} = \{\}$ **by** (*blast*)**hence** $(\sum y \in \{s. G s\}. P y * t \ll \lambda z. z = y \gg s) +$
 $(\sum y \in \{s. \neg G s\}. P y * t \ll \lambda z. z = y \gg s) =$
 $(\sum y \in (\{s. G s\} \cup \{s. \neg G s\}). P y * t \ll \lambda z. z = y \gg s)$ **by** (*auto intro: sum.union-disjoint[symmetric]*)**also** {**have** $\{s. G s\} \cup \{s. \neg G s\} = UNIV$ **by** (*blast*)**hence** $(\sum y \in (\{s. G s\} \cup \{s. \neg G s\}). P y * t \ll \lambda z. z = y \gg s) =$
 $(\lambda x. \sum y \in UNIV. P y * t (\lambda x. \ll \lambda z. z = y \gg x) x) s$ **by** (*simp*)

}

also {**from** *sP* **have** $\bigwedge y. 0 \leq P y$ **by** (*auto*)**with** *healthy-scalingD* [*OF ht*]**have** $(\lambda x. \sum y \in UNIV. P y * t (\lambda x. \ll \lambda z. z = y \gg x) x) s =$
 $(\lambda x. \sum y \in UNIV. t (\lambda x. P y * \ll \lambda z. z = y \gg x) x) s$ **by** (*simp add: scalingD*)

}

also {**from** *sat ht sP***have** $(\lambda x. \sum y \in UNIV. t (\lambda x. P y * \ll \lambda z. z = y \gg x) x) \leq$
 $t (\lambda x. \sum y \in UNIV. P y * \ll \lambda z. z = y \gg x)$ **by** (*intro sub-add-sum sound-intros, auto*)**hence** $(\lambda x. \sum y \in UNIV. t (\lambda x. P y * \ll \lambda z. z = y \gg x) x) s \leq$
 $t (\lambda x. \sum y \in UNIV. P y * \ll \lambda z. z = y \gg x) s$ **by** (*auto*)

}

also {**have** *rwI*: $(\lambda x. \sum y \in UNIV. P y * \ll \lambda z. z = y \gg x) =$
 $(\lambda x. \sum y \in UNIV. \text{if } y = x \text{ then } P y \text{ else } 0)$ **by** (*rule ext [OF sum.cong]*) *auto***also from** *sP* **have** $\dots \Vdash P$ **by** (*cases finite (UNIV::'s set), auto simp: sum.delta*)**finally have** *leP*: $\lambda x. \sum y \in UNIV. P y * \ll \lambda z. z = y \gg x \Vdash P .$ **moreover have** *sound* $(\lambda x. \sum y \in UNIV. P y * \ll \lambda z. z = y \gg x)$ **proof** (*intro soundI2 bounded-byI nnegI sum-nonneg ballI*)**fix** x **from** *leP* **have** $(\sum y \in UNIV. P y * \ll \lambda z. z = y \gg x) \leq P x$ **by** (*auto*)

also from sP have ... \leq *bound-of P* **by**(*auto*)
finally show $(\sum_{y \in UNIV}. P y * \ll \lambda z. z = y \gg x) \leq$ *bound-of P* .
fix y
from sP show $0 \leq P y * \ll \lambda z. z = y \gg x$
by(*auto intro:mult-nonneg-nonneg*)
qed
ultimately have $t (\lambda x. \sum_{y \in UNIV}. P y * \ll \lambda z. z = y \gg x) s \leq t P s$
using sP by(*auto intro:le-funD[OF mono-transD, OF healthy-monoD, OF ht]*)
}
finally show ?*thesis* .
qed

Sub-distributivity

definition *sub-distrib* ::

$((s \Rightarrow real) \Rightarrow (s \Rightarrow real)) \Rightarrow bool$

where

$sub-distrib\ t \iff (\forall P\ s. sound\ P \longrightarrow t\ P\ s \oplus I \leq t\ (\lambda s'. P\ s' \oplus I)\ s)$

lemma *sub-distribI*[*intro*]:

$\llbracket \bigwedge P\ s. sound\ P \implies t\ P\ s \oplus I \leq t\ (\lambda s'. P\ s' \oplus I)\ s \rrbracket \implies sub-distrib\ t$
by(*simp add:sub-distrib-def*)

lemma *sub-distribI2*:

$\llbracket \bigwedge P. sound\ P \implies \lambda s. t\ P\ s \oplus I \Vdash t\ (\lambda s. P\ s \oplus I) \rrbracket \implies sub-distrib\ t$
by(*auto*)

lemma *sub-distribD*[*dest*]:

$\llbracket sub-distrib\ t; sound\ P \rrbracket \implies t\ P\ s \oplus I \leq t\ (\lambda s'. P\ s' \oplus I)\ s$
by(*simp add:sub-distrib-def*)

lemma *equiv-sub-distrib*:

fixes $t::(s \Rightarrow real) \Rightarrow s \Rightarrow real$

assumes eq : *equiv-trans* $t\ u$

and sd : *sub-distrib* t

shows *sub-distrib* u

proof

fix $P::s \Rightarrow real$ **and** $s::s$

assume sP : *sound* P

with eq **have** $u\ P\ s \oplus I = t\ P\ s \oplus I$ **by**(*simp add:equiv-transD*)

also from sP sd have ... $\leq t (\lambda s. P s \oplus I) s$ **by**(*auto*)

also from sP eq have ... $= u (\lambda s. P s \oplus I) s$

by(*simp add:equiv-transD tminus-sound*)

finally show $u\ P\ s \oplus I \leq u (\lambda s. P s \oplus I) s$.

qed

Sublinearity implies sub-distributivity:

lemma *sublinear-sub-distrib*:

fixes $t::('s \Rightarrow \text{real}) \Rightarrow 's \Rightarrow \text{real}$
assumes $slt: \text{sublinear } t$
shows $\text{sub-distrib } t$
proof
fix $P::'s \Rightarrow \text{real}$ **and** $s::'s$
assume $sP: \text{sound } P$
moreover have $\text{sound } (\lambda s. 0)$ **by** (auto)
ultimately show $t P s \ominus 1 \leq t (\lambda s. P s \ominus 1) s$
by $(\text{rule sublinearD}[OF slt, \text{where } a=1 \text{ and } b=0 \text{ and } c=1, \text{simplified}])$
qed

Healthiness, sub-additivity and sub-distributivity imply sublinearity. This is how we usually show sublinearity.

lemma $sd\text{-}sa\text{-}sublinear$:
fixes $t::('s \Rightarrow \text{real}) \Rightarrow 's \Rightarrow \text{real}$
assumes $sdt: \text{sub-distrib } t$ **and** $sat: \text{sub-add } t$ **and** $ht: \text{healthy } t$
shows $\text{sublinear } t$

proof
fix $P::'s \Rightarrow \text{real}$ **and** $Q::'s \Rightarrow \text{real}$ **and** $s::'s$
and $a::\text{real}$ **and** $b::\text{real}$ **and** $c::\text{real}$
assume $sP: \text{sound } P$ **and** $sQ: \text{sound } Q$
and $nna: 0 \leq a$ **and** $nnb: 0 \leq b$ **and** $nnc: 0 \leq c$

from $ht sP sQ nna nnb$
have $saP: \text{sound } (\lambda s. a * P s)$ **and** $staP: \text{sound } (\lambda s. a * t P s)$
and $sbQ: \text{sound } (\lambda s. b * Q s)$ **and** $stbQ: \text{sound } (\lambda s. b * t Q s)$
by $(\text{auto intro:sc-sound})$
hence $sabPQ: \text{sound } (\lambda s. a * P s + b * Q s)$
and $stabPQ: \text{sound } (\lambda s. a * t P s + b * t Q s)$
by $(\text{auto intro:sound-sum})$

from $ht sP sQ nna nnb$
have $a * t P s + b * t Q s = t (\lambda s. a * P s) s + t (\lambda s. b * Q s) s$
by $(\text{simp add:scalingD healthy-scalingD})$
also from $saP sbQ sat$
have $t (\lambda s. a * P s) s + t (\lambda s. b * Q s) s \leq$
 $t (\lambda s. a * P s + b * Q s) s$ **by** (blast)
finally
have $\text{notm}: a * t P s + b * t Q s \leq t (\lambda s. a * P s + b * Q s) s$.

show $a * t P s + b * t Q s \ominus c \leq t (\lambda s'. a * P s' + b * Q s' \ominus c) s$

proof $(\text{cases } c = 0)$
case True **note** $z = \text{this}$
from $stabPQ$ **have** $\bigwedge s. 0 \leq a * t P s + b * t Q s$ **by** (auto)
moreover from $sabPQ$
have $\bigwedge s. 0 \leq a * P s + b * Q s$ **by** (auto)
ultimately show $?thesis$ **by** (simp add:z notm)
next
case False **note** $nz = \text{this}$

from nz **and** nnc **have** nni : $0 \leq \text{inverse } c \text{ by}(\text{auto})$
have $\bigwedge s. (\text{inverse } c * a) * P s + (\text{inverse } c * b) * Q s =$
 $\text{inverse } c * (a * P s + b * Q s)$
by(*simp add: divide-simps*)
with $sabPQ$ **and** nni
have si : *sound* $(\lambda s. (\text{inverse } c * a) * P s + (\text{inverse } c * b) * Q s)$
by(*auto intro:sc-sound*)
hence sim : *sound* $(\lambda s. (\text{inverse } c * a) * P s + (\text{inverse } c * b) * Q s \ominus 1)$
by(*auto intro!:tminus-sound*)

from nz
have $a * t P s + b * t Q s \ominus c =$
 $(c * \text{inverse } c) * a * t P s +$
 $(c * \text{inverse } c) * b * t Q s \ominus c$
by(*simp*)
also
have $\dots = c * (\text{inverse } c * a * t P s) +$
 $c * (\text{inverse } c * b * t Q s) \ominus c$
by(*simp add:field-simps*)
also from nnc
have $\dots = c * (\text{inverse } c * a * t P s + \text{inverse } c * b * t Q s \ominus 1)$
by(*simp add:distrib-left tminus-left-distrib*)
also {
have X : $\bigwedge s. (\text{inverse } c * a) * t P s + (\text{inverse } c * b) * t Q s =$
 $\text{inverse } c * (a * t P s + b * t Q s)$ **by**(*simp add: divide-simps*)
also from nni **and** $notm$
have $\text{inverse } c * (a * t P s + b * t Q s) \leq$
 $\text{inverse } c * (t (\lambda s. a * P s + b * Q s) s)$
by(*blast intro:mult-left-mono*)
also from nni *ht* $sabPQ$
have $\dots = t (\lambda s. (\text{inverse } c * a) * P s + (\text{inverse } c * b) * Q s) s$
by(*simp add:scalingD[OF healthy-scalingD, OF ht] algebra-simps*)
finally
have $(\text{inverse } c * a) * t P s + (\text{inverse } c * b) * t Q s \ominus 1 \leq$
 $t (\lambda s. (\text{inverse } c * a) * P s + (\text{inverse } c * b) * Q s) s \ominus 1$
by(*rule tminus-left-mono*)
also {
from sdt si
have $t (\lambda s. (\text{inverse } c * a) * P s + (\text{inverse } c * b) * Q s) s \ominus 1 \leq$
 $t (\lambda s. (\text{inverse } c * a) * P s + (\text{inverse } c * b) * Q s \ominus 1) s$
by(*blast*)
}
finally
have $c * (\text{inverse } c * a * t P s + \text{inverse } c * b * t Q s \ominus 1) \leq$
 $c * t (\lambda s. \text{inverse } c * a * P s + \text{inverse } c * b * Q s \ominus 1) s$
using nnc **by**(*blast intro:mult-left-mono*)
}
also from nnc *ht* sim

```

have  $c * t (\lambda s. \text{inverse } c * a * P s + \text{inverse } c * b * Q s \ominus I) s$ 
  =  $t (\lambda s. c * (\text{inverse } c * a * P s + \text{inverse } c * b * Q s \ominus I)) s$ 
  by(simp add:scalingD healthy-scalingD)
also from nnc
have ... =  $t (\lambda s. c * (\text{inverse } c * a * P s) +$ 
   $c * (\text{inverse } c * b * Q s) \ominus c) s$ 
  by(simp add:distrib-left tminus-left-distrib)
also have ... =  $t (\lambda s. (c * \text{inverse } c) * a * P s +$ 
   $(c * \text{inverse } c) * b * Q s \ominus c) s$ 
  by(simp add:field-simps)
finally
show  $a * t P s + b * t Q s \ominus c \leq t (\lambda s'. a * P s' + b * Q s' \ominus c) s$ 
  using nz by(simp)
qed
qed

```

Sub-conjunctivity

definition

sub-conj :: $((s \Rightarrow \text{real}) \Rightarrow s \Rightarrow \text{real}) \Rightarrow \text{bool}$

where

sub-conj $t \equiv \forall P Q. (\text{sound } P \wedge \text{sound } Q) \longrightarrow$
 $t P \ \&\& \ t Q \Vdash t (P \ \&\& \ Q)$

lemma *sub-conjI*[*intro*]:

$\llbracket \bigwedge P Q. \llbracket \text{sound } P; \text{sound } Q \rrbracket \Longrightarrow$
 $t P \ \&\& \ t Q \Vdash t (P \ \&\& \ Q) \rrbracket \Longrightarrow \text{sub-conj } t$

unfolding *sub-conj-def* **by**(*simp*)

lemma *sub-conjD*[*dest*]:

$\llbracket \text{sub-conj } t; \text{sound } P; \text{sound } Q \rrbracket \Longrightarrow t P \ \&\& \ t Q \Vdash t (P \ \&\& \ Q)$

unfolding *sub-conj-def* **by**(*simp*)

lemma *sub-conj-wp-twice*:

fixes $f :: s \Rightarrow ((s \Rightarrow \text{real}) \Rightarrow s \Rightarrow \text{real})$

assumes *all*: $\forall s. \text{sub-conj } (f s)$

shows *sub-conj* $(\lambda P s. f s P s)$

proof(*rule sub-conjI, rule le-funI*)

fix $P :: s \Rightarrow \text{real}$ **and** $Q :: s \Rightarrow \text{real}$ **and** s

assume sP : *sound* P **and** sQ : *sound* Q

have $((\lambda s. f s P s) \ \&\& \ (\lambda s. f s Q s)) s = (f s P \ \&\& \ f s Q) s$

by(*simp add:exp-conj-def*)

also {

from *all* **have** *sub-conj* $(f s)$ **by**(*blast*)

with sP **and** sQ **have** $(f s P \ \&\& \ f s Q) s \leq f s (P \ \&\& \ Q) s$

by(*blast*)

}

finally show $((\lambda s. f s P s) \ \&\& \ (\lambda s. f s Q s)) s \leq f s (P \ \&\& \ Q) s .$

qed

Sublinearity implies sub-conjunctivity:

lemma *sublinear-sub-conj*:

fixes $t::('s \Rightarrow \text{real}) \Rightarrow 's \Rightarrow \text{real}$

assumes slt : *sublinear* t

shows *sub-conj* t

proof(*rule sub-conjI, rule le-funI, unfold exp-conj-def pconj-def*)

fix $P::'s \Rightarrow \text{real}$ **and** $Q::'s \Rightarrow \text{real}$ **and** $s::'s$

assume sP : *sound* P **and** sQ : *sound* Q

thus $t P s + t Q s \ominus I \leq t (\lambda s. P s + Q s \ominus I) s$

by(*rule sublinearD[OF slt, where a=I and b=I and c=I, simplified]*)

qed

Sublinearity under equivalence

Sublinearity is preserved by equivalence.

lemma *equiv-sublinear*:

$\llbracket \text{equiv-trans } t \ u; \text{sublinear } t; \text{healthy } t \rrbracket \Longrightarrow \text{sublinear } u$

by(*iprover intro:sd-sa-sublinear healthy-equivI*

dest:equiv-sub-distrib equiv-sub-add

sublinear-sub-distrib sublinear-subadd

healthy-feasibleD)

3.2.4 Determinism

Transformers which are both additive, and maximal among those that satisfy feasibility are *deterministic*, and will turn out to be maximal in the refinement order.

Additivity

Full additivity is not generally satisfied. It holds for (sub-)probabilistic transformers however.

definition

additive $:: (('a \Rightarrow \text{real}) \Rightarrow 'a \Rightarrow \text{real}) \Rightarrow \text{bool}$

where

$\text{additive } t \equiv \forall P Q. (\text{sound } P \wedge \text{sound } Q) \longrightarrow$

$t (\lambda s. P s + Q s) = (\lambda s. t P s + t Q s)$

lemma *additiveD*:

$\llbracket \text{additive } t; \text{sound } P; \text{sound } Q \rrbracket \Longrightarrow t (\lambda s. P s + Q s) = (\lambda s. t P s + t Q s)$

by(*simp add:additive-def*)

lemma *additiveI[intro]*:

$\llbracket \bigwedge P Q s. \llbracket \text{sound } P; \text{sound } Q \rrbracket \Longrightarrow t (\lambda s. P s + Q s) s = t P s + t Q s \rrbracket \Longrightarrow$
additive t

unfolding *additive-def* **by**(*blast*)

Additivity is strictly stronger than sub-additivity.

lemma *additive-sub-add*:
additive $t \implies$ *sub-add* t
by(*simp add:sub-addI additiveD*)

The additivity property extends to finite summation.

lemma *additive-sum*:
fixes $S::'s$ *set*
assumes *additive*: *additive* t
and *healthy*: *healthy* t
and *finite*: *finite* S
and sPz : $\bigwedge z. \text{sound } (P z)$
shows $t (\lambda x. \sum_{y \in S}. P y x) = (\lambda x. \sum_{y \in S}. t (P y) x)$
proof(*rule finite-induct, simp-all add:assms*)
fix $z::'s$ **and** $T::'s$ *set*
assume $finT$: *finite* T
and IH : $t (\lambda x. \sum_{y \in T}. P y x) = (\lambda x. \sum_{y \in T}. t (P y) x)$

from *additive* sPz
have $t (\lambda x. P z x + (\sum_{y \in T}. P y x)) =$
 $(\lambda x. t (P z) x + t (\lambda x. \sum_{y \in T}. P y x) x)$
by(*auto intro!:sum-sound additiveD*)
also from IH
have $\dots = (\lambda x. t (P z) x + (\sum_{y \in T}. t (P y) x))$
by(*simp*)
finally show $t (\lambda x. P z x + (\sum_{y \in T}. P y x)) =$
 $(\lambda x. t (P z) x + (\sum_{y \in T}. t (P y) x)) .$

qed

An additive transformer (over a finite state space) is linear: it is simply the weighted sum of final expectation values, the weights being the probability of reaching a given final state. This is useful for reasoning using the forward, or “gambling game” interpretation.

lemma *additive-delta-split*:
fixes $t::('s::\text{finite} \implies \text{real}) \implies 's \implies \text{real}$
assumes *additive*: *additive* t
and ht : *healthy* t
and sP : *sound* P
shows $t P x = (\sum_{y \in UNIV}. P y * t \ll \lambda z. z = y \gg x)$
proof –
have $\bigwedge x. (\sum_{y \in UNIV}. P y * \ll \lambda z. z = y \gg x) =$
 $(\sum_{y \in UNIV}. \text{if } y = x \text{ then } P y \text{ else } 0)$
by (*rule sum.cong*) *auto*
also have $\bigwedge x. \dots x = P x$
by(*simp add:sum.delta*)
finally
have $t P x = t (\lambda x. \sum_{y \in UNIV}. P y * \ll \lambda z. z = y \gg x) x$
by(*simp*)

```

also {
  from  $sP$  have  $\bigwedge z. \text{sound} (\lambda a. P z * \ll \lambda z a. za = z \gg a)$ 
    by(auto intro!:mult-sound)
  hence  $t (\lambda x. \sum_{y \in UNIV}. P y * \ll \lambda z. z = y \gg x) x =$ 
     $(\sum_{y \in UNIV}. t (\lambda x. P y * \ll \lambda z. z = y \gg x) x)$ 
    by(subst additive-sum, simp-all add:assms)
}
also from  $sP$ 
have  $(\sum_{y \in UNIV}. t (\lambda x. P y * \ll \lambda z. z = y \gg x) x) =$ 
   $(\sum_{y \in UNIV}. P y * t \ll \lambda z. z = y \gg x)$ 
  by(subst scalingD[OF healthy-scalingD, OF ht], auto)
finally show  $t P x = (\sum_{y \in UNIV}. P y * t \ll \lambda z. z = y \gg x) .$ 
qed

```

We can group the states in the linear form, to split on the value of a predicate (guard).

```

lemma additive-guard-split:
fixes  $t :: ('s :: \text{finite} \Rightarrow \text{real}) \Rightarrow 's \Rightarrow \text{real}$ 
assumes additive: additive t
  and ht: healthy t
  and sP: sound P
shows  $t P x = (\sum_{y \in \{s. G s\}}. P y * t \ll \lambda z. z = y \gg x) +$ 
   $(\sum_{y \in \{s. \neg G s\}}. P y * t \ll \lambda z. z = y \gg x)$ 
proof –
from assms
have  $t P x = (\sum_{y \in UNIV}. P y * t \ll \lambda z. z = y \gg x)$ 
  by(rule additive-delta-split)
also {
  have  $UNIV = \{s. G s\} \cup \{s. \neg G s\}$ 
    by(auto)
  hence  $(\sum_{y \in UNIV}. P y * t \ll \lambda z. z = y \gg x) =$ 
     $(\sum_{y \in \{s. G s\} \cup \{s. \neg G s\}}. P y * t \ll \lambda z. z = y \gg x)$ 
    by(simp)
}
also
have  $(\sum_{y \in \{s. G s\} \cup \{s. \neg G s\}}. P y * t \ll \lambda z. z = y \gg x) =$ 
   $(\sum_{y \in \{s. G s\}}. P y * t \ll \lambda z. z = y \gg x) +$ 
   $(\sum_{y \in \{s. \neg G s\}}. P y * t \ll \lambda z. z = y \gg x)$ 
  by(auto intro:sum.union-disjoint)
finally show ?thesis .
qed

```

Maximality

definition

$\text{maximal} :: (('a \Rightarrow \text{real}) \Rightarrow 'a \Rightarrow \text{real}) \Rightarrow \text{bool}$

where

$\text{maximal } t \equiv \forall c. 0 \leq c \longrightarrow t (\lambda \cdot. c) = (\lambda \cdot. c)$

lemma *maximalI*[*intro*]:
 $\llbracket \bigwedge c. 0 \leq c \implies t (\lambda\cdot. c) = (\lambda\cdot. c) \rrbracket \implies \text{maximal } t$
by(*simp add:maximal-def*)

lemma *maximalD*[*dest*]:
 $\llbracket \text{maximal } t; 0 \leq c \rrbracket \implies t (\lambda\cdot. c) = (\lambda\cdot. c)$
by(*simp add:maximal-def*)

A transformer that is both additive and maximal is deterministic:

definition *determ* :: $((a \Rightarrow \text{real}) \Rightarrow a \Rightarrow \text{real}) \Rightarrow \text{bool}$
where
determ $t \equiv \text{additive } t \wedge \text{maximal } t$

lemma *determI*[*intro*]:
 $\llbracket \text{additive } t; \text{maximal } t \rrbracket \implies \text{determ } t$
by(*simp add:determ-def*)

lemma *determ-additiveD*[*intro*]:
determ $t \implies \text{additive } t$
by(*simp add:determ-def*)

lemma *determ-maximalD*[*intro*]:
determ $t \implies \text{maximal } t$
by(*simp add:determ-def*)

For a fully-deterministic transformer, a transformed standard expectation, and its transformed negation are complementary.

lemma *determ-negate*:
assumes *determ*: *determ* t
shows $t \llbracket P \rrbracket s + t \llbracket \neg P \rrbracket s = I$
proof –
have $t \llbracket P \rrbracket s + t \llbracket \neg P \rrbracket s = t (\lambda s. \llbracket P \rrbracket s + \llbracket \neg P \rrbracket s) s$
by(*simp add:additiveD determ determ-additiveD*)
also {
have $\bigwedge s. \llbracket P \rrbracket s + \llbracket \neg P \rrbracket s = I$
by(*case-tac P s, simp-all*)
hence $t (\lambda s. \llbracket P \rrbracket s + \llbracket \neg P \rrbracket s) = t (\lambda s. I)$
by(*simp*)
}
also have $t (\lambda s. I) = (\lambda s. I)$
by(*simp add:maximalD determ determ-maximalD*)
finally show ?thesis .
qed

3.2.5 Modular Reasoning

The emphasis of a mechanised logic is on automation, and letting the computer tackle the large, uninteresting problems. However, as terms generally grow expo-

nentially in the size of a program, it is still essential to break up a proof and reason in a modular fashion.

The following rules allow proof decomposition, and later will be incorporated into a verification condition generator.

lemma *entails-combine*:

```

assumes  $wp1: P \Vdash t R$ 
and  $wp2: Q \Vdash t S$ 
and  $sc: \text{sub-conj } t$ 
and  $sR: \text{sound } R$ 
and  $sS: \text{sound } S$ 
shows  $P \&\& Q \Vdash t (R \&\& S)$ 
proof –
from  $wp1$  and  $wp2$  have  $P \&\& Q \Vdash t R \&\& t S$ 
by(blast intro:entails-frame)
also from  $sc$  and  $sR$  and  $sS$  have  $\dots \leq t (R \&\& S)$ 
by(rule sub-conjD)
finally show ?thesis .
qed

```

These allow mismatched results to be composed

lemma *entails-strengthen-post*:

```

 $\llbracket P \Vdash t Q; \text{healthy } t; \text{sound } R; Q \Vdash R; \text{sound } Q \rrbracket \implies P \Vdash t R$ 
by(blast intro:entails-trans)

```

lemma *entails-weaken-pre*:

```

 $\llbracket Q \Vdash t R; P \Vdash Q \rrbracket \implies P \Vdash t R$ 
by(blast intro:entails-trans)

```

This rule is unique to pGCL. Use it to scale the post-expectation of a rule to 'fit under' the precondition you need to satisfy.

lemma *entails-scale*:

```

assumes  $wp: P \Vdash t Q$  and  $h: \text{healthy } t$ 
and  $sQ: \text{sound } Q$  and  $pos: 0 \leq c$ 
shows  $(\lambda s. c * P s) \Vdash t (\lambda s. c * Q s)$ 
proof(rule le-funI)
fix  $s$ 
from  $pos$  and  $wp$  have  $c * P s \leq c * t Q s$ 
by(auto intro:mult-left-mono)
with  $sQ$   $pos$   $h$  show  $c * P s \leq t (\lambda s. c * Q s) s$ 
by(simp add:scalingD healthy-scalingD)
qed

```

3.2.6 Transforming Standard Expectations

Reasoning with *standard* expectations, those obtained by embedding a predicate, is often easier, as the analogues of many familiar boolean rules hold in modified form.

One may use a standard pre-expectation as an assumption:

```

lemma use-premise:
  assumes h: healthy t and wP:  $\bigwedge s. P\ s \implies I \leq t \ll Q \gg s$ 
  shows  $\ll P \gg \Vdash t \ll Q \gg$ 
proof(rule entailsI)
  fix s show  $\ll P \gg s \leq t \ll Q \gg s$ 
  proof(cases P s)
    case True with wP show ?thesis by(auto)
  next
    case False with h show ?thesis by(auto)
  qed
qed

```

The other direction works too.

```

lemma fold-premise:
  assumes ht: healthy t
  and wp:  $\ll P \gg \Vdash t \ll Q \gg$ 
  shows  $\forall s. P\ s \longrightarrow I \leq t \ll Q \gg s$ 
proof(clarify)
  fix s assume P s
  hence  $I = \ll P \gg s$  by(simp)
  also from wp have  $\dots \leq t \ll Q \gg s$  by(auto)
  finally show  $I \leq t \ll Q \gg s$  .
qed

```

Predicate conjunction behaves as expected:

```

lemma conj-post:
   $\ll P \Vdash t \ll \lambda s. Q\ s \wedge R\ s \gg; \text{healthy } t \gg \implies P \Vdash t \ll Q \gg$ 
  by(blast intro:entails-strengthen-post implies-entails)

```

Similar to $\ll \text{healthy } ?t; \bigwedge s. ?P\ s \implies I \leq ?t \ll ?Q \gg s \gg \implies \ll ?P \gg \Vdash ?t \ll ?Q \gg$, but more general.

```

lemma entails-pconj-assumption:
  assumes f: feasible t and wP:  $\bigwedge s. P\ s \implies Q\ s \leq t\ R\ s$ 
    and uQ: unitary Q and uR: unitary R
  shows  $\ll P \gg \&\& Q \Vdash t\ R$ 
  unfolding exp-conj-def
proof(rule entailsI)
  fix s show  $\ll P \gg s \&\& Q\ s \leq t\ R\ s$ 
  proof(cases P s)
    case True
      moreover from uQ have  $0 \leq Q\ s$  by(auto)
      ultimately show ?thesis by(simp add:pconj-lone wP)
  next
    case False
      moreover from uQ have  $Q\ s \leq 1$  by(auto)
      ultimately show ?thesis using assms by auto
  qed

```

qed

end

3.3 Induction

```
theory Induction
  imports Expectations Transformers
begin
```

3.3.1 The Lattice of Expectations

Defining recursive (or iterative) programs requires us to reason about fixed points on the semantic objects, in this case expectations. The complication here, compared to the standard Knaster-Tarski theorem (for example, as shown in *HOL.Inductive*), is that we do not have a complete lattice.

Finding a lower bound is easy (it's $\lambda-. 0::'b$), but as we do not insist on any global bound on expectations (and work directly in HOL's real type, rather than extending it with a point at infinity), there is no top element. We solve the problem by defining the least (greatest) fixed point, restricted to an internally-bounded set, allowing us to substitute this bound for the top element. This works as long as the set contains at least one fixed point, which appears as an extra assumption in all the theorems.

This also works semantically, thanks to the definition of healthiness. Given a healthy transformer parameterised by some sound expectation: t . Imagine that we wish to find the least fixed point of $t P$. In practice, t is generally doubly healthy, that is $\forall P. \text{sound } P \longrightarrow \text{healthy } (t P)$ and $\forall Q. \text{sound } Q \longrightarrow \text{healthy } (\lambda P. t P Q)$. Thus by feasibility, $t P Q$ must be bounded by $\text{bound-of } P$. Thus, as by definition $x \leq t P x$ for any fixed point, all must lie in the set of sound expectations bounded above by $\lambda-. \text{bound-of } P$.

definition *Inf-exp* :: 's expect set \Rightarrow 's expect
where *Inf-exp* $S = (\lambda s. \text{Inf } \{f s \mid f. f \in S\})$

lemma *Inf-exp-lower*:

$\llbracket P \in S; \forall P \in S. \text{nneg } P \rrbracket \Longrightarrow \text{Inf-exp } S \leq P$

unfolding *Inf-exp-def*

by(intro le-funI cInf-lower bdd-belowI[**where** $m=0$], auto)

lemma *Inf-exp-greatest*:

$\llbracket S \neq \{\}; \forall P \in S. Q \leq P \rrbracket \Longrightarrow Q \leq \text{Inf-exp } S$

unfolding *Inf-exp-def* **by**(auto intro!:le-funI[OF cInf-greatest])

definition *Sup-exp* :: 's expect set \Rightarrow 's expect

where *Sup-exp* $S = (\text{if } S = \{\} \text{ then } \lambda s. 0 \text{ else } (\lambda s. \text{Sup } \{f s \mid f. f \in S\}))$

lemma *Sup-exp-upper*:

$\llbracket P \in S; \forall P \in S. \text{bounded-by } b P \rrbracket \Longrightarrow P \leq \text{Sup-exp } S$

unfolding *Sup-exp-def*
by(cases $S = \{\}$, simp-all, intro le-funI cSup-upper bdd-aboveI[**where** $M = b$], auto)

lemma *Sup-exp-least*:
 $\llbracket \forall P \in S. P \leq Q; \text{nneg } Q \rrbracket \implies \text{Sup-exp } S \leq Q$
unfolding *Sup-exp-def*
by(cases $S = \{\}$, auto intro!:le-funI[OF cSup-least])

lemma *Sup-exp-sound*:
assumes $sS: \bigwedge P. P \in S \implies \text{sound } P$
and $bS: \bigwedge P. P \in S \implies \text{bounded-by } b P$
shows $\text{sound } (\text{Sup-exp } S)$
proof(cases $S = \{\}$, simp add:Sup-exp-def, blast,
intro soundI2 bounded-byI2 nnegI2)
assume $neS: S \neq \{\}$
then obtain P **where** $Pin: P \in S$ **by**(auto)
with sS bS **have** $nP: \text{nneg } P \text{ bounded-by } b P$ **by**(auto)
hence $nb: 0 \leq b$ **by**(auto)

from bS nb **show** $\text{Sup-exp } S \Vdash \lambda s. b$
by(auto intro:Sup-exp-least)

from nP **have** $\lambda s. 0 \Vdash P$ **by**(auto)
also from Pin bS **have** $P \Vdash \text{Sup-exp } S$
by(auto intro:Sup-exp-upper)
finally show $\lambda s. 0 \Vdash \text{Sup-exp } S$.

qed

definition *lfp-exp* :: 's trans \Rightarrow 's expect
where $\text{lfp-exp } t = \text{Inf-exp } \{P. \text{sound } P \wedge t P \leq P\}$

lemma *lfp-exp-lowerbound*:
 $\llbracket t P \leq P; \text{sound } P \rrbracket \implies \text{lfp-exp } t \leq P$
unfolding *lfp-exp-def* **by**(auto intro:Inf-exp-lower)

lemma *lfp-exp-greatest*:
 $\llbracket \bigwedge P. \llbracket t P \leq P; \text{sound } P \rrbracket \implies Q \leq P; \text{sound } Q; t R \Vdash R; \text{sound } R \rrbracket \implies Q \leq \text{lfp-exp } t$
unfolding *lfp-exp-def* **by**(auto intro:Inf-exp-greatest)

lemma *feasible-lfp-exp-sound*:
 $\text{feasible } t \implies \text{sound } (\text{lfp-exp } t)$
by(intro soundI2 bounded-byI2 nnegI2, auto intro!:lfp-exp-lowerbound lfp-exp-greatest)

lemma *lfp-exp-sound*:
assumes $fR: t R \Vdash R$ **and** $sR: \text{sound } R$
shows $\text{sound } (\text{lfp-exp } t)$
proof(intro soundI2)
from fR sR **have** $\text{lfp-exp } t \Vdash R$
by(auto intro:lfp-exp-lowerbound)

also from sR **have** $R \Vdash \lambda s. \text{bound-of } R$ **by**(*auto*)
finally show *bounded-by* (*bound-of* R) (*lfp-exp* t) **by**(*auto*)
from fR sR **show** *nneg* (*lfp-exp* t) **by**(*auto intro:lfp-exp-greatest*)
qed

lemma *lfp-exp-bound*:
 $(\bigwedge P. \text{unitary } P \implies \text{unitary } (t P)) \implies \text{bounded-by } 1$ (*lfp-exp* t)
by(*auto intro!:lfp-exp-lowerbound*)

lemma *lfp-exp-unitary*:
 $(\bigwedge P. \text{unitary } P \implies \text{unitary } (t P)) \implies \text{unitary } (\text{lfp-exp } t)$
proof(*intro unitaryI[OF lfp-exp-sound lfp-exp-bound]*, *simp-all*)
assume $IH: \bigwedge P. \text{unitary } P \implies \text{unitary } (t P)$
have $\text{unitary } (\lambda s. 1)$ **by**(*auto*)
with IH **have** $\text{unitary } (t (\lambda s. 1))$ **by**(*auto*)
thus $t (\lambda s. 1) \Vdash \lambda s. 1$ **by**(*auto*)
show $\text{sound } (\lambda s. 1)$ **by**(*auto*)
qed

lemma *lfp-exp-lemma2*:
fixes $t::'s \text{ trans}$
assumes $st: \bigwedge P. \text{sound } P \implies \text{sound } (t P)$
and $mt: \text{mono-trans } t$
and $fR: t R \Vdash R$ **and** $sR: \text{sound } R$
shows $t (\text{lfp-exp } t) \leq \text{lfp-exp } t$
proof(*rule lfp-exp-greatest[of t, OF - - fR sR]*)
from fR sR **show** $\text{sound } (t (\text{lfp-exp } t))$ **by**(*auto intro:lfp-exp-sound st*)

fix $P::'s \text{ expect}$
assume $fP: t P \Vdash P$ **and** $sP: \text{sound } P$
hence $\text{lfp-exp } t \Vdash P$ **by**(*rule lfp-exp-lowerbound*)
with fP sP **have** $t (\text{lfp-exp } t) \Vdash t P$ **by**(*auto intro:mono-transD[OF mt] lfp-exp-sound*)
also note fP
finally show $t (\text{lfp-exp } t) \Vdash P$.
qed

lemma *lfp-exp-lemma3*:
assumes $st: \bigwedge P. \text{sound } P \implies \text{sound } (t P)$
and $mt: \text{mono-trans } t$
and $fR: t R \Vdash R$ **and** $sR: \text{sound } R$
shows $\text{lfp-exp } t \leq t (\text{lfp-exp } t)$
by(*iprover intro:lfp-exp-lowerbound lfp-exp-sound lfp-exp-lemma2 assms mono-transD[OF mt]*)

lemma *lfp-exp-unfold*:
assumes $nt: \bigwedge P. \text{sound } P \implies \text{sound } (t P)$
and $mt: \text{mono-trans } t$
and $fR: t R \Vdash R$ **and** $sR: \text{sound } R$
shows $\text{lfp-exp } t = t (\text{lfp-exp } t)$

by(*iprover* *intro:antisym* *lfp-exp-lemma2* *lfp-exp-lemma3* *assms*)

definition *gfp-exp* :: 's trans \Rightarrow 's expect

where *gfp-exp* *t* = *Sup-exp* {*P*. *unitary* *P* \wedge *P* \leq *t* *P*}

lemma *gfp-exp-upperbound*:

$\llbracket P \leq t P; \text{unitary } P \rrbracket \Longrightarrow P \leq \text{gfp-exp } t$

by(*auto* *simp:gfp-exp-def* *intro:Sup-exp-upper*)

lemma *gfp-exp-least*:

$\llbracket \bigwedge P. \llbracket P \leq t P; \text{unitary } P \rrbracket \Longrightarrow P \leq Q; \text{unitary } Q \rrbracket \Longrightarrow \text{gfp-exp } t \leq Q$

unfolding *gfp-exp-def* **by**(*auto* *intro:Sup-exp-least*)

lemma *gfp-exp-bound*:

$(\bigwedge P. \text{unitary } P \Longrightarrow \text{unitary } (t P)) \Longrightarrow \text{bounded-by } 1 (\text{gfp-exp } t)$

unfolding *gfp-exp-def*

by(*rule* *bounded-byI2[OF Sup-exp-least]*, *auto*)

lemma *gfp-exp-nneg*[*iff*]:

nneg (*gfp-exp* *t*)

proof(*intro* *nnegI2*, *simp* *add:gfp-exp-def*, *cases*)

assume *empty*: {*P*. *unitary* *P* \wedge *P* \Vdash *t* *P*} = {}

show $\lambda s. 0 \Vdash \text{Sup-exp } \{P. \text{unitary } P \wedge P \Vdash t P\}$

by(*simp* *only:empty* *Sup-exp-def*, *auto*)

next

assume {*P*. *unitary* *P* \wedge *P* \Vdash *t* *P*} \neq {}

then obtain *Q* **where** *Qin*: *Q* \in {*P*. *unitary* *P* \wedge *P* \Vdash *t* *P*} **by**(*auto*)

hence $\lambda s. 0 \Vdash Q$ **by**(*auto*)

also from *Qin* **have** *Q* $\Vdash \text{Sup-exp } \{P. \text{unitary } P \wedge P \Vdash t P\}$

by(*auto* *intro:Sup-exp-upper*)

finally show $\lambda s. 0 \Vdash \text{Sup-exp } \{P. \text{unitary } P \wedge P \Vdash t P\}$.

qed

lemma *gfp-exp-unitary*:

$(\bigwedge P. \text{unitary } P \Longrightarrow \text{unitary } (t P)) \Longrightarrow \text{unitary } (\text{gfp-exp } t)$

by(*iprover* *intro:gfp-exp-nneg* *gfp-exp-bound* *unitaryI2*)

lemma *gfp-exp-lemma2*:

assumes *ft*: $\bigwedge P. \text{unitary } P \Longrightarrow \text{unitary } (t P)$

and *mt*: $\bigwedge P Q. \llbracket \text{unitary } P; \text{unitary } Q; P \Vdash Q \rrbracket \Longrightarrow t P \Vdash t Q$

shows *gfp-exp* *t* \leq *t* (*gfp-exp* *t*)

proof(*rule* *gfp-exp-least*)

show *unitary* (*t* (*gfp-exp* *t*)) **by**(*auto* *intro:gfp-exp-unitary* *ft*)

fix *P*

assume *fp*: *P* \leq *t* *P* **and** *uP*: *unitary* *P*

with *ft* **have** *P* \leq *gfp-exp* *t* **by**(*auto* *intro:gfp-exp-upperbound*)

with *uP* *gfp-exp-unitary* *ft*

have *t* *P* \leq *t* (*gfp-exp* *t*) **by**(*blast* *intro: mt*)

with *fp* **show** *P* \leq *t* (*gfp-exp* *t*) **by**(*auto*)

qed

lemma *gfp-exp-lemma3*:

assumes *ft*: $\bigwedge P. \text{unitary } P \implies \text{unitary } (t P)$
and *mt*: $\bigwedge P Q. \llbracket \text{unitary } P; \text{unitary } Q; P \Vdash Q \rrbracket \implies t P \Vdash t Q$
shows $t (\text{gfp-exp } t) \leq \text{gfp-exp } t$
by (*iprover* *intro:gfp-exp-upperbound unitary-sound*
gfp-exp-unitary gfp-exp-lemma2 assms)

lemma *gfp-exp-unfold*:

$(\bigwedge P. \text{unitary } P \implies \text{unitary } (t P)) \implies (\bigwedge P Q. \llbracket \text{unitary } P; \text{unitary } Q; P \Vdash Q \rrbracket \implies t P \Vdash t Q) \implies$
 $\text{gfp-exp } t = t (\text{gfp-exp } t)$
by (*iprover* *intro:antisym gfp-exp-lemma2 gfp-exp-lemma3*)

3.3.2 The Lattice of Transformers

In addition to fixed points on expectations, we also need to reason about fixed points on expectation transformers. The interpretation of a recursive program in pGCL is as a fixed point of a function from transformers to transformers. In contrast to the case of expectations, *healthy* transformers do form a complete lattice, where the bottom element is $\lambda -. 0::'c$, and the top element is the greatest allowed by feasibility: $\lambda P -. \text{bound-of } P$.

definition *Inf-trans* :: *'s trans set* \implies *'s trans*
where *Inf-trans* $S = (\lambda P. \text{Inf-exp } \{t P \mid t. t \in S\})$

lemma *Inf-trans-lower*:

$\llbracket t \in S; \forall u \in S. \forall P. \text{sound } P \longrightarrow \text{sound } (u P) \rrbracket \implies \text{le-trans } (\text{Inf-trans } S) t$
unfolding *Inf-trans-def*
by (*rule* *le-transI[OF Inf-exp-lower]*, *blast+*)

lemma *Inf-trans-greatest*:

$\llbracket S \neq \{\}; \forall t \in S. \forall P. \text{le-trans } u t \rrbracket \implies \text{le-trans } u (\text{Inf-trans } S)$
unfolding *Inf-trans-def* **by** (*auto* *intro!:le-transI[OF Inf-exp-greatest]*)

definition *Sup-trans* :: *'s trans set* \implies *'s trans*
where *Sup-trans* $S = (\lambda P. \text{Sup-exp } \{t P \mid t. t \in S\})$

lemma *Sup-trans-upper*:

$\llbracket t \in S; \forall u \in S. \forall P. \text{unitary } P \longrightarrow \text{unitary } (u P) \rrbracket \implies \text{le-utrans } t (\text{Sup-trans } S)$
unfolding *Sup-trans-def*
by (*intro* *le-utransI[OF Sup-exp-upper]*, *auto* *intro:unitary-bound*)

lemma *Sup-trans-upper2*:

$\llbracket t \in S; \forall u \in S. \forall P. (\text{nneg } P \wedge \text{bounded-by } b P) \longrightarrow (\text{nneg } (u P) \wedge \text{bounded-by } b (u P));$
 $\text{nneg } P; \text{bounded-by } b P \rrbracket \implies t P \Vdash \text{Sup-trans } S P$
unfolding *Sup-trans-def* **by** (*blast* *intro:Sup-exp-upper*)

lemma *Sup-trans-least*:

$\llbracket \forall t \in S. \text{le-trans } t \ u; \bigwedge P. \text{unitary } P \implies \text{unitary } (u \ P) \rrbracket \implies \text{le-trans } (\text{Sup-trans } S) \ u$
unfolding *Sup-trans-def*
by(*auto intro!::sound-nneg[OF unitary-sound] le-transI[OF Sup-exp-least]*)

lemma *Sup-trans-least2*:

$\llbracket \forall t \in S. \forall P. \text{nneg } P \longrightarrow \text{bounded-by } b \ P \longrightarrow t \ P \Vdash u \ P;$
 $\forall u \in S. \forall P. (\text{nneg } P \wedge \text{bounded-by } b \ P) \longrightarrow (\text{nneg } (u \ P) \wedge \text{bounded-by } b \ (u \ P));$
 $\text{nneg } P; \text{bounded-by } b \ P; \bigwedge P. \llbracket \text{nneg } P; \text{bounded-by } b \ P \rrbracket \implies \text{nneg } (u \ P) \rrbracket \implies$
 $\text{Sup-trans } S \ P \Vdash u \ P$
unfolding *Sup-trans-def* **by**(*blast intro!::Sup-exp-least*)

lemma *feasible-Sup-trans*:

fixes *S::'s trans set*
assumes *fS: $\forall t \in S. \text{feasible } t$*
shows *feasible (Sup-trans S)*
proof(*cases S={}, simp add:Sup-trans-def Sup-exp-def, blast, intro feasibleI*)
fix *b::real and P::'s expect*
assume *bP: bounded-by b P and nP: nneg P*
and *neS: $S \neq \{\}$*

from *neS obtain t where tin: $t \in S$* **by**(*auto*)

with *fS have ft: feasible t* **by**(*auto*)

with *bP nP have $\lambda s. 0 \Vdash t \ P$* **by**(*auto*)

also {

from *bP nP have sound P* **by**(*auto*)

with *tin fS have $t \ P \Vdash \text{Sup-trans } S \ P$*

by(*auto intro!::Sup-trans-upper2*)

}

finally show *nneg (Sup-trans S P)* **by**(*auto*)

from *fS bP nP*

show *bounded-by b (Sup-trans S P)*

by(*auto intro!::bounded-byI2[OF Sup-trans-least2]*)

qed

definition *lfp-trans :: ('s trans \Rightarrow 's trans) \Rightarrow 's trans*

where *lfp-trans T = Inf-trans $\{t. (\forall P. \text{sound } P \longrightarrow \text{sound } (t \ P)) \wedge \text{le-trans } (T \ t) \ t\}$*

lemma *lfp-trans-lowerbound*:

$\llbracket \text{le-trans } (T \ t) \ t; \bigwedge P. \text{sound } P \implies \text{sound } (t \ P) \rrbracket \implies \text{le-trans } (\text{lfp-trans } T) \ t$

unfolding *lfp-trans-def*

by(*auto intro:Inf-trans-lower*)

lemma *lfp-trans-greatest*:

$\llbracket \bigwedge t \ P. \llbracket \text{le-trans } (T \ t) \ t; \bigwedge P. \text{sound } P \implies \text{sound } (t \ P) \rrbracket \implies \text{le-trans } u \ t;$

$\bigwedge P. \text{sound } P \implies \text{sound } (v \ P); \text{le-trans } (T \ v) \ v \rrbracket \implies$

$\text{le-trans } u \ (\text{lfp-trans } T)$

unfolding *lfp-trans-def* **by**(*rule Inf-trans-greatest, auto*)

lemma *lfp-trans-sound*:
fixes $P Q::'s$ *expect*
assumes sP : *sound* P
and fv : *le-trans* $(T v) v$
and sv : $\bigwedge P. \text{sound } P \implies \text{sound } (v P)$
shows *sound* $(\text{lfp-trans } T P)$
proof(*intro soundI2 bounded-byI2 nnegI2*)
from $fv sv$ **have** *le-trans* $(\text{lfp-trans } T) v$
by(*iprover intro:lfp-trans-lowerbound*)
with sP **have** $\text{lfp-trans } T P \Vdash v P$ **by**(*auto*)
also {
from $sv sP$ **have** *sound* $(v P)$ **by**(*iprover*)
hence $v P \Vdash \lambda s. \text{bound-of } (v P)$ **by**(*auto*)
}
finally show $\text{lfp-trans } T P \Vdash \lambda s. \text{bound-of } (v P)$.

have *le-trans* $(\lambda P s. 0) (\text{lfp-trans } T)$
proof(*intro lfp-trans-greatest*)
fix $t::'s$ *trans*
assume $\bigwedge P. \text{sound } P \implies \text{sound } (t P)$
hence $\bigwedge P. \text{sound } P \implies \lambda s. 0 \Vdash t P$ **by**(*auto*)
thus *le-trans* $(\lambda P s. 0) t$ **by**(*auto*)
next
fix $P::'s$ *expect*
assume *sound* P **thus** *sound* $(v P)$ **by**(*rule sv*)
next
show *le-trans* $(T v) v$ **by**(*rule fv*)
qed
with sP **show** $\lambda s. 0 \Vdash \text{lfp-trans } T P$ **by**(*auto*)
qed

lemma *lfp-trans-unitary*:
fixes $P Q::'s$ *expect*
assumes uP : *unitary* P
and fv : *le-trans* $(T v) v$
and sv : $\bigwedge P. \text{sound } P \implies \text{sound } (v P)$
and fT : *le-trans* $(T (\lambda P s. \text{bound-of } P)) (\lambda P s. \text{bound-of } P)$
shows *unitary* $(\text{lfp-trans } T P)$
proof(*rule unitaryI*)
from *unitary-sound*[*OF* uP] $fv sv$ **show** *sound* $(\text{lfp-trans } T P)$
by(*rule lfp-trans-sound*)

show *bounded-by* $1 (\text{lfp-trans } T P)$
proof(*rule bounded-byI2*)
from fT **have** *le-trans* $(\text{lfp-trans } T) (\lambda P s. \text{bound-of } P)$
by(*auto intro: lfp-trans-lowerbound*)
with uP **have** $\text{lfp-trans } T P \Vdash \lambda s. \text{bound-of } P$ **by**(*auto*)
also from uP **have** $\dots \Vdash \lambda s. 1$ **by**(*auto*)

finally show $\text{lfp-trans } T P \Vdash \lambda s. I$.
qed
qed

lemma *lfp-trans-lemma2*:

fixes $v::'s \text{ trans}$
assumes $\text{mono}: \bigwedge t u. \llbracket \text{le-trans } t u; \bigwedge P. \text{sound } P \implies \text{sound } (t P);$
 $\bigwedge P. \text{sound } P \implies \text{sound } (u P) \rrbracket \implies \text{le-trans } (T t) (T u)$
and $nT: \bigwedge t P. \llbracket \bigwedge Q. \text{sound } Q \implies \text{sound } (t Q); \text{sound } P \rrbracket \implies \text{sound } (T t P)$
and $fv: \text{le-trans } (T v) v$
and $sv: \bigwedge P. \text{sound } P \implies \text{sound } (v P)$
shows $\text{le-trans } (T (\text{lfp-trans } T)) (\text{lfp-trans } T)$
proof(*rule lfp-trans-greatest*[**where** $T=T$ **and** $v=v$], *simp-all add:assms*)
fix $t::'s \text{ trans}$ **and** $P::'s \text{ expect}$
assume $ft: \text{le-trans } (T t) t$ **and** $st: \bigwedge P. \text{sound } P \implies \text{sound } (t P)$
hence $\text{le-trans } (\text{lfp-trans } T) t$ **by**(*auto intro!:lfp-trans-lowerbound*)
with $ft st$ **have** $\text{le-trans } (T (\text{lfp-trans } T)) (T t)$
by(*iprover intro:mono lfp-trans-sound fv sv*)
also note ft
finally show $\text{le-trans } (T (\text{lfp-trans } T)) t$.
qed

lemma *lfp-trans-lemma3*:

fixes $v::'s \text{ trans}$
assumes $\text{mono}: \bigwedge t u. \llbracket \text{le-trans } t u; \bigwedge P. \text{sound } P \implies \text{sound } (t P);$
 $\bigwedge P. \text{sound } P \implies \text{sound } (u P) \rrbracket \implies \text{le-trans } (T t) (T u)$
and $sT: \bigwedge t P. \llbracket \bigwedge Q. \text{sound } Q \implies \text{sound } (t Q); \text{sound } P \rrbracket \implies \text{sound } (T t P)$
and $fv: \text{le-trans } (T v) v$
and $sv: \bigwedge P. \text{sound } P \implies \text{sound } (v P)$
shows $\text{le-trans } (\text{lfp-trans } T) (T (\text{lfp-trans } T))$
proof(*rule lfp-trans-lowerbound*)
fix $P::'s \text{ expect}$
assume $sP: \text{sound } P$
have $n1: \bigwedge P. \text{sound } P \implies \text{sound } (\text{lfp-trans } T P)$
by(*iprover intro:lfp-trans-sound fv sv*)
with sP **have** $n2: \text{sound } (\text{lfp-trans } T P)$
by(*iprover intro:lfp-trans-sound fv sv sT*)
with $n1 sP$ **show** $n3: \text{sound } (T (\text{lfp-trans } T) P)$
by(*iprover intro: sT*)
next
show $\text{le-trans } (T (T (\text{lfp-trans } T))) (T (\text{lfp-trans } T))$
by(*rule mono[OF lfp-trans-lemma2, OF mono],*
(iprover intro:assms lfp-trans-sound)+)
qed

lemma *lfp-trans-unfold*:

fixes $P::'s \text{ expect}$
assumes $\text{mono}: \bigwedge t u. \llbracket \text{le-trans } t u; \bigwedge P. \text{sound } P \implies \text{sound } (t P);$
 $\bigwedge P. \text{sound } P \implies \text{sound } (u P) \rrbracket \implies \text{le-trans } (T t) (T u)$

and sT : $\bigwedge t P. \llbracket \bigwedge Q. \text{sound } Q \implies \text{sound } (t Q); \text{sound } P \rrbracket \implies \text{sound } (T t P)$
and fv : $le\text{-trans } (T v) v$
and sv : $\bigwedge P. \text{sound } P \implies \text{sound } (v P)$
shows $equiv\text{-trans } (lfp\text{-trans } T) (T (lfp\text{-trans } T))$
by(*rule le-trans-antisym*,
rule lfp-trans-lemma3[*OF mono*], (*iprover intro:assms*)+,
rule lfp-trans-lemma2[*OF mono*], (*iprover intro:assms*)+)

definition $gfp\text{-trans} :: ('s \text{ trans} \implies 's \text{ trans}) \implies 's \text{ trans}$
where $gfp\text{-trans } T = \text{Sup-trans } \{t. (\forall P. \text{unitary } P \longrightarrow \text{unitary } (t P)) \wedge le\text{-utrans } t (T t)\}$

lemma $gfp\text{-trans-upperbound}$:
 $\llbracket le\text{-utrans } t (T t); \bigwedge P. \text{unitary } P \implies \text{unitary } (t P) \rrbracket \implies le\text{-utrans } t (gfp\text{-trans } T)$
unfolding $gfp\text{-trans-def}$ **by**(*auto intro:Sup-trans-upper*)

lemma $gfp\text{-trans-least}$:
 $\llbracket \bigwedge t. \llbracket le\text{-utrans } t (T t); \bigwedge P. \text{unitary } P \implies \text{unitary } (t P) \rrbracket \implies le\text{-utrans } t u;$
 $\bigwedge P. \text{unitary } P \implies \text{unitary } (u P) \rrbracket \implies$
 $le\text{-utrans } (gfp\text{-trans } T) u$
unfolding $gfp\text{-trans-def}$ **by**(*auto intro:Sup-trans-least*)

lemma $gfp\text{-trans-unitary}$:
fixes $P :: 's \text{ expect}$
assumes uP : $\text{unitary } P$
shows $\text{unitary } (gfp\text{-trans } T P)$
proof(*intro unitaryI2 nnegI2 bounded-byI2*)
show $gfp\text{-trans } T P \Vdash \lambda s. 1$
unfolding $gfp\text{-trans-def}$ $Sup\text{-trans-def}$
proof(*rule Sup-exp-least, clarify*)
fix $t :: 's \text{ trans}$
assume $\forall P. \text{unitary } P \longrightarrow \text{unitary } (t P)$
with uP **have** $\text{unitary } (t P)$ **by**(*auto*)
thus $t P \Vdash \lambda s. 1$ **by**(*auto*)
next
show $nneg (\lambda s. 1 :: \text{real})$ **by**(*auto*)
qed

let $?S = \{t P \mid t. t \in \{t. (\forall P. \text{unitary } P \longrightarrow \text{unitary } (t P)) \wedge le\text{-utrans } t (T t)\}\}$
show $\lambda s. 0 \Vdash \text{gfp-trans } T P$
unfolding $gfp\text{-trans-def}$ $Sup\text{-trans-def}$
proof(*cases*)
assume $empty$: $?S = \{\}$
show $\lambda s. 0 \Vdash \text{Sup-exp } ?S$
by(*simp only:empty Sup-exp-def, auto*)
next
assume $?S \neq \{\}$
then obtain Q **where** $Q \text{ in: } Q \in ?S$ **by**(*auto*)
with uP **have** $\text{unitary } Q$ **by**(*auto*)
hence $\lambda s. 0 \Vdash Q$ **by**(*auto*)
also with uP $Q \text{ in}$ **have** $Q \Vdash \text{Sup-exp } ?S$

```

proof(intro Sup-exp-upper, blast, clarify)
  fix  $t::'s$  trans
  assume  $\forall Q. \text{unitary } Q \longrightarrow \text{unitary } (t Q)$ 
  with  $uP$  show bounded-by 1 (t P) by(auto)
qed
finally show  $\lambda s. 0 \Vdash \text{Sup-exp } ?S .$ 
qed
qed

```

lemma *gfp-trans-lemma2*:

```

assumes mono:  $\bigwedge t u. \llbracket \text{le-utrans } t u; \bigwedge P. \text{unitary } P \Longrightarrow \text{unitary } (t P);$   

 $\bigwedge P. \text{unitary } P \Longrightarrow \text{unitary } (u P) \rrbracket \Longrightarrow \text{le-utrans } (T t) (T u)$ 
  and hT:  $\bigwedge t P. \llbracket \bigwedge Q. \text{unitary } Q \Longrightarrow \text{unitary } (t Q); \text{unitary } P \rrbracket \Longrightarrow \text{unitary } (T t P)$ 
shows le-utrans (gfp-trans T) (T (gfp-trans T))
proof(rule gfp-trans-least, simp-all add:hT gfp-trans-unitary)
  fix  $t$ 
  assume fp: le-utrans t (T t) and ht:  $\bigwedge P. \text{unitary } P \Longrightarrow \text{unitary } (t P)$ 

  note fp
  also {
    from fp ht have le-utrans t (gfp-trans T) by(rule gfp-trans-upperbound)
    moreover note ht gfp-trans-unitary
    ultimately have le-utrans (T t) (T (gfp-trans T)) by(rule mono)
  }
  finally show le-utrans t (T (gfp-trans T)) .
qed

```

lemma *gfp-trans-lemma3*:

```

assumes mono:  $\bigwedge t u. \llbracket \text{le-utrans } t u; \bigwedge P. \text{unitary } P \Longrightarrow \text{unitary } (t P);$   

 $\bigwedge P. \text{unitary } P \Longrightarrow \text{unitary } (u P) \rrbracket \Longrightarrow \text{le-utrans } (T t) (T u)$ 
  and hT:  $\bigwedge t P. \llbracket \bigwedge Q. \text{unitary } Q \Longrightarrow \text{unitary } (t Q); \text{unitary } P \rrbracket \Longrightarrow \text{unitary } (T t P)$ 
shows le-utrans (T (gfp-trans T)) (gfp-trans T)
by(blast intro!:mono gfp-trans-unitary gfp-trans-upperbound gfp-trans-lemma2 mono hT)

```

lemma *gfp-trans-unfold*:

```

assumes mono:  $\bigwedge t u. \llbracket \text{le-utrans } t u; \bigwedge P. \text{unitary } P \Longrightarrow \text{unitary } (t P);$   

 $\bigwedge P. \text{unitary } P \Longrightarrow \text{unitary } (u P) \rrbracket \Longrightarrow \text{le-utrans } (T t) (T u)$ 
  and hT:  $\bigwedge t P. \llbracket \bigwedge Q. \text{unitary } Q \Longrightarrow \text{unitary } (t Q); \text{unitary } P \rrbracket \Longrightarrow \text{unitary } (T t P)$ 
shows equiv-utrans (gfp-trans T) (T (gfp-trans T))
using assms by(auto intro!: le-utrans-antisym gfp-trans-lemma2 gfp-trans-lemma3)

```

3.3.3 Tail Recursion

The least (greatest) fixed point of a tail-recursive expression on transformers is equivalent (given appropriate side conditions) to the least (greatest) fixed point on expectations.

lemma *gfp-pulldown*:

fixes $P::'s$ expect
assumes tailcall: $\bigwedge u P. \text{unitary } P \implies T u P = t P (u P)$
and ft: $\bigwedge t P. \llbracket \bigwedge Q. \text{unitary } Q \implies \text{unitary } (t Q); \text{unitary } P \rrbracket \implies \text{unitary } (T t P)$
and ft: $\bigwedge P Q. \text{unitary } P \implies \text{unitary } Q \implies \text{unitary } (t P Q)$
and mt: $\bigwedge P Q R. \llbracket \text{unitary } P; \text{unitary } Q; \text{unitary } R; Q \Vdash R \rrbracket \implies t P Q \Vdash t P R$
and uP: $\text{unitary } P$
and monoT: $\bigwedge t u. \llbracket \text{le-utrans } t u; \bigwedge P. \text{unitary } P \implies \text{unitary } (t P); \bigwedge P. \text{unitary } P \implies \text{unitary } (u P) \rrbracket \implies \text{le-utrans } (T t) (T u)$
shows gfp-trans $T P = \text{gfp-exp } (t P)$ (**is** $?X P = ?Y P$)
proof(rule antisym)
show $?X P \leq ?Y P$
proof(rule gfp-exp-upperbound)
from monoT ft uP **have** $(\text{gfp-trans } T) P \leq (T (\text{gfp-trans } T)) P$
by(auto intro!: le-utransD[OF gfp-trans-lemma2])
also from uP **have** $(T (\text{gfp-trans } T)) P = t P (\text{gfp-trans } T P)$ **by**(rule tailcall)
finally show $\text{gfp-trans } T P \Vdash t P (\text{gfp-trans } T P)$.
from uP gfp-trans-unitary **show** $\text{unitary } (\text{gfp-trans } T P)$ **by**(auto)
qed
show $?Y P \leq ?X P$
proof(rule le-utransD[OF gfp-trans-upperbound], simp-all add:assms)
show $\text{le-utrans } (\lambda a. \text{gfp-exp } (t a)) (T (\lambda a. \text{gfp-exp } (t a)))$
proof(rule le-utransI)
fix $Q::'s$ expect **assume** $uQ: \text{unitary } Q$
with ft **have** $\bigwedge R. \text{unitary } R \implies \text{unitary } (t Q R)$ **by**(auto)
with mt[OF uQ] **have** $\text{gfp-exp } (t Q) = t Q (\text{gfp-exp } (t Q))$ **by**(blast intro:gfp-exp-unfold)
also from uQ **have** $\dots = T (\lambda a. \text{gfp-exp } (t a)) Q$ **by**(rule tailcall[symmetric])
finally show $\text{gfp-exp } (t Q) \leq T (\lambda a. \text{gfp-exp } (t a)) Q$ **by**(simp)
qed
fix $Q::'s$ expect **assume** $\text{unitary } Q$
with ft **have** $\bigwedge R. \text{unitary } R \implies \text{unitary } (t Q R)$ **by**(auto)
thus $\text{unitary } (\text{gfp-exp } (t Q))$ **by**(rule gfp-exp-unitary)
qed
qed

lemma lfp-pulldown:
fixes $P::'s$ expect **and** $t::'s$ expect $\implies 's$ trans
and $T::'s$ trans $\implies 's$ trans
assumes tailcall: $\bigwedge u P. \text{sound } P \implies T u P = t P (u P)$
and st: $\bigwedge P Q. \text{sound } P \implies \text{sound } Q \implies \text{sound } (t P Q)$
and mt: $\bigwedge P. \text{sound } P \implies \text{mono-trans } (t P)$
and monoT: $\bigwedge t u. \llbracket \text{le-trans } t u; \bigwedge P. \text{sound } P \implies \text{sound } (t P); \bigwedge P. \text{sound } P \implies \text{sound } (u P) \rrbracket \implies \text{le-trans } (T t) (T u)$
and nT: $\bigwedge t P. \llbracket \bigwedge Q. \text{sound } Q \implies \text{sound } (t Q); \text{sound } P \rrbracket \implies \text{sound } (T t P)$
and fv: $\text{le-trans } (T v) v$
and sv: $\bigwedge P. \text{sound } P \implies \text{sound } (v P)$
and sP: $\text{sound } P$
shows lfp-trans $T P = \text{lfp-exp } (t P)$ (**is** $?X P = ?Y P$)
proof(rule antisym)
show $?Y P \leq ?X P$

```

proof(rule lfp-exp-lowerbound)
  from sP have t P (lfp-trans T P) = (T (lfp-trans T)) P by(rule tailcall[symmetric])
  also have (T (lfp-trans T)) P ≤ (lfp-trans T) P
    by(rule le-transD[OF lfp-trans-lemma2[OF monoT]], (iprover intro:assms)+)
  finally show t P (lfp-trans T P) ≤ lfp-trans T P .
  from sP show sound (lfp-trans T P)
    by(iprover intro:lfp-trans-sound assms)
qed

```

```

have  $\bigwedge P. \text{sound } P \implies t P (v P) = T v P$  by(simp add:tailcall)
also have  $\bigwedge P. \text{sound } P \implies \dots P \Vdash v P$  by(auto intro:le-transD[OF fv])
finally have fvP:  $\bigwedge P. \text{sound } P \implies t P (v P) \Vdash v P$  .
have svP:  $\bigwedge P. \text{sound } P \implies \text{sound } (v P)$  by(rule sv)

```

```

show ?X P ≤ ?Y P
proof(rule le-transD[OF lfp-trans-lowerbound, OF - - sP])
  show le-trans (T (λa. lfp-exp (t a))) (λa. lfp-exp (t a))
  proof(rule le-transI)
    fix P::'s expect
    assume sP: sound P

    from sP have T (λa. lfp-exp (t a)) P = t P (lfp-exp (t P)) by(rule tailcall)
    also have t P (lfp-exp (t P)) = lfp-exp (t P)
      by(iprover intro: lfp-exp-unfold[symmetric] sP st mt fvP svP)
    finally show T (λa. lfp-exp (t a)) P  $\Vdash$  lfp-exp (t P) by(simp)
  qed
  fix P::'s expect
  assume sound P
  with fvP svP show sound (lfp-exp (t P))
    by(blast intro:lfp-exp-sound)
qed
qed

```

definition Inf-utrans :: 's trans set \Rightarrow 's trans
where Inf-utrans S = (if S = {} then $\lambda P s. 1$ else Inf-trans S)

lemma Inf-utrans-lower:

```

 $\llbracket t \in S; \forall t \in S. \forall P. \text{unitary } P \longrightarrow \text{unitary } (t P) \rrbracket \implies \text{le-utrans } (\text{Inf-utrans } S) t$ 
unfolding Inf-utrans-def
by(cases S={},
  auto intro!:le-utransI Inf-exp-lower sound-nneg unitary-sound
  simp:Inf-trans-def)

```

lemma Inf-utrans-greatest:

```

 $\llbracket \bigwedge P. \text{unitary } P \implies \text{unitary } (t P); \forall u \in S. \text{le-utrans } t u \rrbracket \implies \text{le-utrans } t (\text{Inf-utrans } S)$ 
unfolding Inf-utrans-def Inf-trans-def
by(cases S={}, simp-all, (blast intro!:le-utransI Inf-exp-greatest)+)

```

end

Chapter 4

The pGCL Language

4.1 A Shallow Embedding of pGCL in HOL

theory *Embedding* imports *Misc Induction* begin

4.1.1 Core Primitives and Syntax

A pGCL program is embedded directly as its strict or liberal transformer. This is achieved with an additional parameter, specifying which semantics should be obeyed.

type-synonym $'s\ prog = bool \Rightarrow ('s \Rightarrow real) \Rightarrow ('s \Rightarrow real)$

Abort either always fails, $\lambda P s. 0::'c$, or always succeeds, $\lambda P s. 1::'c$.

definition $Abort :: 's\ prog$
where $Abort \equiv \lambda ab P s. \text{if } ab \text{ then } 0 \text{ else } 1$

Skip does nothing at all.

definition $Skip :: 's\ prog$
where $Skip \equiv \lambda ab P. P$

Apply lifts a state transformer into the space of programs.

definition $Apply :: ('s \Rightarrow 's) \Rightarrow 's\ prog$
where $Apply f \equiv \lambda ab P s. P (f s)$

Seq is sequential composition.

definition $Seq :: 's\ prog \Rightarrow 's\ prog \Rightarrow 's\ prog$
(**infixl** ;; 59)
where $Seq a b \equiv (\lambda ab. a\ ab\ o\ b\ ab)$

PC is *probabilistic* choice between programs.

definition $PC :: 's\ prog \Rightarrow ('s \Rightarrow real) \Rightarrow 's\ prog \Rightarrow 's\ prog$
(**-** \oplus **-** [58,57,57] 57)

where $PC\ a\ P\ b \equiv \lambda ab\ Q\ s. P\ s * a\ ab\ Q\ s + (1 - P\ s) * b\ ab\ Q\ s$

DC is *demonic choice* between programs.

definition $DC :: 's\ prog \Rightarrow 's\ prog \Rightarrow 's\ prog\ (-\ \sqcap\ -\ [58,57]\ 57)$

where $DC\ a\ b \equiv \lambda ab\ Q\ s. \min\ (a\ ab\ Q\ s)\ (b\ ab\ Q\ s)$

AC is *angelic choice* between programs.

definition $AC :: 's\ prog \Rightarrow 's\ prog \Rightarrow 's\ prog\ (-\ \sqcup\ -\ [58,57]\ 57)$

where $AC\ a\ b \equiv \lambda ab\ Q\ s. \max\ (a\ ab\ Q\ s)\ (b\ ab\ Q\ s)$

$Embed$ allows any expectation transformer to be treated syntactically as a program, by ignoring the failure flag.

definition $Embed :: 's\ trans \Rightarrow 's\ prog$

where $Embed\ t = (\lambda ab. t)$

Mu is the recursive primitive, and is either then least or greatest fixed point.

definition $Mu :: ('s\ prog \Rightarrow 's\ prog) \Rightarrow 's\ prog\ (\mathbf{binder}\ \mu\ 50)$

where $Mu(T) \equiv (\lambda ab. \text{if } ab \text{ then lfp-trans } (\lambda t. T\ (Embed\ t)\ ab) \\ \text{else gfp-trans } (\lambda t. T\ (Embed\ t)\ ab))$

$repeat$ expresses finite repetition

primrec

$repeat :: nat \Rightarrow 'a\ prog \Rightarrow 'a\ prog$

where

$repeat\ 0\ p = Skip\ |$

$repeat\ (Suc\ n)\ p = p\ ;;\ repeat\ n\ p$

$SetDC$ is demonic choice between a set of alternatives, which may depend on the state.

definition $SetDC :: ('a \Rightarrow 's\ prog) \Rightarrow ('s \Rightarrow 'a\ set) \Rightarrow 's\ prog$

where $SetDC\ f\ S \equiv \lambda ab\ P\ s. \text{Inf}\ ((\lambda a. f\ a\ ab\ P\ s)\ 'S\ s)$

syntax $-SetDC :: ptnrn \Rightarrow ('s \Rightarrow 'a\ set) \Rightarrow 's\ prog \Rightarrow 's\ prog$
 $(\sqcap\ -\ \in\ -\ / - 100)$

translations $\sqcap_{x \in S}. p == CONST\ SetDC\ (\%x. p)\ S$

The above syntax allows us to write $\sqcap_{x \in S}. Apply\ f$

$SetPC$ is *probabilistic choice* from a set. Note that this is only meaningful for distributions of finite support.

definition

$SetPC :: ('a \Rightarrow 's\ prog) \Rightarrow ('s \Rightarrow 'a \Rightarrow real) \Rightarrow 's\ prog$

where

$SetPC\ f\ p \equiv \lambda ab\ P\ s. \sum_{a \in \text{supp}\ (p\ s)}. p\ s\ a * f\ a\ ab\ P\ s$

$Bind$ allows us to name an expression in the current state, and re-use it later.

definition

$Bind :: ('s \Rightarrow 'a) \Rightarrow ('a \Rightarrow 's \text{ prog}) \Rightarrow 's \text{ prog}$
where
 $Bind \ g \ f \ a \ b \equiv \lambda P \ s. \text{ let } a = g \ s \text{ in } f \ a \ a \ b \ P \ s$

This gives us something like let syntax

syntax $-Bind :: p\text{trn} \Rightarrow ('s \Rightarrow 'a) \Rightarrow 's \text{ prog} \Rightarrow 's \text{ prog}$
 $(- \text{ is } - \text{ in } - [55,55,55]55)$
translations $x \text{ is } f \text{ in } a \Rightarrow CONST \ Bind \ f \ (\%x. a)$

definition $flip :: ('a \Rightarrow 'b \Rightarrow 'c) \Rightarrow 'b \Rightarrow 'a \Rightarrow 'c$
where $[simp]: flip \ f = (\lambda b \ a. f \ a \ b)$

The following pair of translations introduce let-style syntax for *SetPC* and *SetDC*, respectively.

syntax $-PBind :: p\text{trn} \Rightarrow ('s \Rightarrow \text{real}) \Rightarrow 's \text{ prog} \Rightarrow 's \text{ prog}$
 $(bind - at - in - [55,55,55]55)$
translations $bind \ x \ \text{at} \ p \ \text{in} \ a \Rightarrow CONST \ SetPC \ (\%x. a) \ (CONST \ flip \ (\%x. p))$

syntax $-DBind :: p\text{trn} \Rightarrow ('s \Rightarrow 'a \ \text{set}) \Rightarrow 's \text{ prog} \Rightarrow 's \text{ prog}$
 $(bind - from - in - [55,55,55]55)$
translations $bind \ x \ \text{from} \ S \ \text{in} \ a \Rightarrow CONST \ SetDC \ (\%x. a) \ S$

The following syntax translations are for convenience when using a record as the state type.

syntax
 $-assign :: ident \Rightarrow 'a \Rightarrow 's \text{ prog} \ (- := - [1000,900]900)$
ML <
 $fun \ assign\text{-tr} - [Const \ (name,-), \ arg] =$
 $Const \ (Embedding.Apply, \ dummyT) \ \$$
 $Abs \ (s, \ dummyT,$
 $Syntax.const \ (suffix \ Record.updateN \ name) \ \$$
 $Abs \ (Name.uu-, \ dummyT, \ arg \ \$ \ Bound \ 1) \ \$ \ Bound \ 0)$
 $| \ assign\text{-tr} - \ ts = raise \ TERM \ (assign\text{-tr}, \ ts)$
 >
parse-translation <[(@{syntax-const -assign}, assign-tr)]>

syntax
 $-SetPC :: ident \Rightarrow ('s \Rightarrow 'a \Rightarrow \text{real}) \Rightarrow 's \text{ prog}$
 $(choose - at - [66,66]66)$
ML <
 $fun \ set\text{-pc}\text{-tr} - [Const \ (f,-), \ P] =$
 $Const \ (SetPC, \ dummyT) \ \$$
 $Abs \ (v, \ dummyT,$
 $(Const \ (Embedding.Apply, \ dummyT) \ \$$
 $Abs \ (s, \ dummyT,$
 $Syntax.const \ (suffix \ Record.updateN \ f) \ \$$
 $Abs \ (Name.uu-, \ dummyT, \ Bound \ 2) \ \$ \ Bound \ 0))) \ \$$
 P

```

| set-pc-tr - ts = raise TERM (set-pc-tr, ts)
>
parse-translation <[(@{syntax-const -SetPC}, set-pc-tr)]>

syntax
-set-dc :: ident => ('s => 'a set) => 's prog (- :∈ - [66,66]66)
ML <
fun set-dc-tr - [Const (f,-), S] =
  Const (SetDC, dummyT) $
  Abs (v, dummyT,
    (Const (Embedding.Apply, dummyT) $
      Abs (s, dummyT,
        Syntax.const (suffix Record.updateNf) $
          Abs (Name.uu-, dummyT, Bound 2) $ Bound 0))) $
  S
| set-dc-tr - ts = raise TERM (set-dc-tr, ts)
>
parse-translation <[(@{syntax-const -set-dc}, set-dc-tr)]>

```

These definitions instantiate the embedding as either weakest precondition (True) or weakest liberal precondition (False).

```

syntax
-set-dc-UNIV :: ident => 's prog (any - [66]66)

```

```

translations
-set-dc-UNIV x => -set-dc x (%-. CONST UNIV)

```

```

definition
wp :: 's prog ⇒ 's trans
where
wp pr ≡ pr True

```

```

definition
wlp :: 's prog ⇒ 's trans
where
wlp pr ≡ pr False

```

If-Then-Else as a degenerate probabilistic choice.

```

abbreviation(input)
if-then-else :: ['s ⇒ bool, 's prog, 's prog] ⇒ 's prog
  (If - Then - Else - 58)
where
If P Then a Else b == a «P» ⊕ b

```

Syntax for loops

```

abbreviation
do-while :: ['s ⇒ bool, 's prog] ⇒ 's prog
  (do - → // (4 -) // od)
where

```

do-while $P a \equiv \mu x. \text{If } P \text{ Then } a ;; x \text{ Else Skip}$

4.1.2 Unfolding rules for non-recursive primitives

lemma *eval-wp-Abort*:

$wp \text{ Abort } P = (\lambda s. 0)$

unfolding *wp-def Abort-def* **by**(*simp*)

lemma *eval-wlp-Abort*:

$wlp \text{ Abort } P = (\lambda s. 1)$

unfolding *wlp-def Abort-def* **by**(*simp*)

lemma *eval-wp-Skip*:

$wp \text{ Skip } P = P$

unfolding *wp-def Skip-def* **by**(*simp*)

lemma *eval-wlp-Skip*:

$wlp \text{ Skip } P = P$

unfolding *wlp-def Skip-def* **by**(*simp*)

lemma *eval-wp-Apply*:

$wp (\text{Apply } f) P = P \circ f$

unfolding *wp-def Apply-def* **by**(*simp add:o-def*)

lemma *eval-wlp-Apply*:

$wlp (\text{Apply } f) P = P \circ f$

unfolding *wlp-def Apply-def* **by**(*simp add:o-def*)

lemma *eval-wp-Seq*:

$wp (a ;; b) P = (wp a \circ wp b) P$

unfolding *wp-def Seq-def* **by**(*simp*)

lemma *eval-wlp-Seq*:

$wlp (a ;; b) P = (wlp a \circ wlp b) P$

unfolding *wlp-def Seq-def* **by**(*simp*)

lemma *eval-wp-PC*:

$wp (a \oplus b) P = (\lambda s. Q s * wp a P s + (1 - Q s) * wp b P s)$

unfolding *wp-def PC-def* **by**(*simp*)

lemma *eval-wlp-PC*:

$wlp (a \oplus b) P = (\lambda s. Q s * wlp a P s + (1 - Q s) * wlp b P s)$

unfolding *wlp-def PC-def* **by**(*simp*)

lemma *eval-wp-DC*:

$wp (a \sqcap b) P = (\lambda s. \min (wp a P s) (wp b P s))$

unfolding *wp-def DC-def* **by**(*simp*)

lemma *eval-wlp-DC*:

$wlp (a \sqcap b) P = (\lambda s. \min (wlp a P s) (wlp b P s))$
unfolding *wlp-def DC-def by(simp)*

lemma *eval-wp-AC:*
 $wp (a \sqcup b) P = (\lambda s. \max (wp a P s) (wp b P s))$
unfolding *wp-def AC-def by(simp)*

lemma *eval-wlp-AC:*
 $wlp (a \sqcup b) P = (\lambda s. \max (wlp a P s) (wlp b P s))$
unfolding *wlp-def AC-def by(simp)*

lemma *eval-wp-Embed:*
 $wp (Embed t) = t$
unfolding *wp-def Embed-def by(simp)*

lemma *eval-wlp-Embed:*
 $wlp (Embed t) = t$
unfolding *wlp-def Embed-def by(simp)*

lemma *eval-wp-SetDC:*
 $wp (SetDC p S) R s = Inf ((\lambda a. wp (p a) R s) ` S s)$
unfolding *wp-def SetDC-def by(simp)*

lemma *eval-wlp-SetDC:*
 $wlp (SetDC p S) R s = Inf ((\lambda a. wlp (p a) R s) ` S s)$
unfolding *wlp-def SetDC-def by(simp)*

lemma *eval-wp-SetPC:*
 $wp (SetPC f p) P = (\lambda s. \sum a \in \text{supp} (p s). p s a * wp (f a) P s)$
unfolding *wp-def SetPC-def by(simp)*

lemma *eval-wlp-SetPC:*
 $wlp (SetPC f p) P = (\lambda s. \sum a \in \text{supp} (p s). p s a * wlp (f a) P s)$
unfolding *wlp-def SetPC-def by(simp)*

lemma *eval-wp-Mu:*
 $wp (\mu t. T t) = lfp\text{-trans} (\lambda t. wp (T (Embed t)))$
unfolding *wp-def Mu-def by(simp)*

lemma *eval-wlp-Mu:*
 $wlp (\mu t. T t) = gfp\text{-trans} (\lambda t. wlp (T (Embed t)))$
unfolding *wlp-def Mu-def by(simp)*

lemma *eval-wp-Bind:*
 $wp (Bind g f) = (\lambda P s. wp (f (g s)) P s)$
unfolding *Bind-def wp-def Let-def by(simp)*

lemma *eval-wlp-Bind:*
 $wlp (Bind g f) = (\lambda P s. wlp (f (g s)) P s)$

unfolding *Bind-def wlp-def Let-def* **by**(*simp*)

Use *simp add:wlp_eval* to fully unfold a program fragment

lemmas *wp-eval = eval-wp-Abort eval-wlp-Abort eval-wp-Skip eval-wlp-Skip*
eval-wp-Apply eval-wlp-Apply eval-wp-Seq eval-wlp-Seq
eval-wp-PC eval-wlp-PC eval-wp-DC eval-wlp-DC
eval-wp-AC eval-wlp-AC
eval-wp-Embed eval-wlp-Embed eval-wp-SetDC eval-wlp-SetDC
eval-wp-SetPC eval-wlp-SetPC eval-wp-Mu eval-wlp-Mu
eval-wp-Bind eval-wlp-Bind

lemma *Skip-Seq*:

Skip ;; *A = A*

unfolding *Skip-def Seq-def o-def* **by**(*rule refl*)

lemma *Seq-Skip*:

A ;; *Skip = A*

unfolding *Skip-def Seq-def o-def* **by**(*rule refl*)

Use these as *simp* rules to clear out Skips

lemmas *skip-simps = Skip-Seq Seq-Skip*

end

4.2 Healthiness

theory *Healthiness* **imports** *Embedding* **begin**

4.2.1 The Healthiness of the Embedding

Healthiness is mostly derived by structural induction using the simplifier. *Abort*, *Skip* and *Apply* form base cases.

lemma *healthy-wp-Abort*:

healthy (*wp Abort*)

proof(*rule healthy-parts*)

fix *b* **and** *P::'a ⇒ real*

assume *nP: nneg P* **and** *bP: bounded-by b P*

thus *bounded-by b (wp Abort P)*

unfolding *wp-eval* **by**(*blast*)

show *nneg (wp Abort P)*

unfolding *wp-eval* **by**(*blast*)

next

fix *P Q::'a expect*

show *wp Abort P ⊢ wp Abort Q*

unfolding *wp-eval* **by**(*blast*)

next

fix *P* **and** *c* **and** *s::'a*

show $c * wp \text{ Abort } P s = wp \text{ Abort } (\lambda s. c * P s) s$
unfolding $wp\text{-eval}$ **by**(*auto*)
qed

lemma *nearly-healthy-wlp-Abort*:
nearly-healthy ($wlp \text{ Abort}$)
proof(*rule nearly-healthyI*)
fix $P :: 's \Rightarrow real$
show *unitary* ($wlp \text{ Abort } P$)
by(*simp add:wp-eval*)
next
fix $P Q :: 's \text{ expect}$
assume $P \Vdash Q$ **and** *unitary* P **and** *unitary* Q
thus $wlp \text{ Abort } P \Vdash wlp \text{ Abort } Q$
unfolding $wp\text{-eval}$ **by**(*blast*)
qed

lemma *healthy-wp-Skip*:
healthy ($wp \text{ Skip}$)
by(*force intro!:healthy-parts simp:wp-eval*)

lemma *nearly-healthy-wlp-Skip*:
nearly-healthy ($wlp \text{ Skip}$)
by(*auto simp:wp-eval*)

lemma *healthy-wp-Seq*:
fixes $t :: 's \text{ prog}$ **and** u
assumes ht : *healthy* ($wp t$) **and** hu : *healthy* ($wp u$)
shows *healthy* ($wp (t ;; u)$)
proof(*rule healthy-parts, simp-all add:wp-eval*)
fix b **and** $P :: 's \Rightarrow real$
assume *bounded-by* $b P$ **and** *nneg* P
with hu **have** *bounded-by* $b (wp u P)$ **and** *nneg* ($wp u P$) **by**(*auto*)
with ht **show** *bounded-by* $b (wp t (wp u P))$
and *nneg* ($wp t (wp u P)$) **by**(*auto*)
next
fix $P :: 's \Rightarrow real$ **and** Q
assume *sound* P **and** *sound* Q **and** $P \Vdash Q$
with hu **have** *sound* ($wp u P$) **and** *sound* ($wp u Q$)
and $wp u P \Vdash wp u Q$ **by**(*auto*)
with ht **show** $wp t (wp u P) \Vdash wp t (wp u Q)$ **by**(*auto*)
next
fix $P :: 's \Rightarrow real$ **and** $c :: real$ **and** s
assume pos : $0 \leq c$ **and** sP : *sound* P
with ht **and** hu **have** $c * wp t (wp u P) s = wp t (\lambda s. c * wp u P s) s$
by(*auto intro!:scalingD*)
also with hu **and** pos **and** sP **have** $\dots = wp t (wp u (\lambda s. c * P s)) s$
by(*simp add:scalingD[OF healthy-scalingD]*)
finally show $c * wp t (wp u P) s = wp t (wp u (\lambda s. c * P s)) s$.

qed

lemma *nearly-healthy-wlp-Seq*:

fixes $t::'s \text{ prog}$ **and** u
assumes ht : *nearly-healthy* ($wlp\ t$) **and** hu : *nearly-healthy* ($wlp\ u$)
shows *nearly-healthy* ($wlp\ (t\ ;;\ u)$)
proof(*rule nearly-healthyI, simp-all add:wp-eval*)
fix b **and** $P::'s \Rightarrow \text{real}$
assume *unitary* P
with hu **have** *unitary* ($wlp\ u\ P$) **by**(*auto*)
with ht **show** *unitary* ($wlp\ t\ (wlp\ u\ P)$) **by**(*auto*)
next
fix $P\ Q::'s \Rightarrow \text{real}$
assume *unitary* P **and** *unitary* Q **and** $P \Vdash Q$
with hu **have** *unitary* ($wlp\ u\ P$) **and** *unitary* ($wlp\ u\ Q$)
and $wlp\ u\ P \Vdash wlp\ u\ Q$ **by**(*auto*)
with ht **show** $wlp\ t\ (wlp\ u\ P) \Vdash wlp\ t\ (wlp\ u\ Q)$ **by**(*auto*)
qed

lemma *healthy-wp-PC*:

fixes $f::'s \text{ prog}$
assumes hf : *healthy* ($wp\ f$) **and** hg : *healthy* ($wp\ g$)
and uP : *unitary* P
shows *healthy* ($wp\ (f\ p\oplus\ g)$)
proof(*intro healthy-parts bounded-byI nnegI le-funI, simp-all add:wp-eval*)
fix b **and** $Q::'s \Rightarrow \text{real}$ **and** $s::'s$
assume nQ : *nneg* Q **and** bQ : *bounded-by* $b\ Q$

Non-negative:

from nQ **and** bQ **and** hf **have** $0 \leq wp\ f\ Q\ s$ **by**(*auto*)
with uP **have** $0 \leq P\ s * \dots$ **by**(*auto intro:mult-nonneg-nonneg*)
moreover {
from uP **have** $0 \leq 1 - P\ s$
by *auto*
with nQ **and** bQ **and** hg **have** $0 \leq \dots * wp\ g\ Q\ s$
by (*metis healthy-nnegD2 mult-nonneg-nonneg nneg-def*)
}
ultimately show $0 \leq P\ s * wp\ f\ Q\ s + (1 - P\ s) * wp\ g\ Q\ s$
by(*auto intro:mult-nonneg-nonneg*)

Bounded:

from $nQ\ bQ\ hf$ **have** $wp\ f\ Q\ s \leq b$ **by**(*auto*)
with $uP\ nQ\ bQ\ hf$ **have** $P\ s * wp\ f\ Q\ s \leq P\ s * b$
by(*blast intro!:mult-mono*)
moreover {
from $nQ\ bQ\ hg\ uP$
have $wp\ g\ Q\ s \leq b$ **and** $0 \leq 1 - P\ s$
by *auto*
with $nQ\ bQ\ hg$ **have** $(1 - P\ s) * wp\ g\ Q\ s \leq (1 - P\ s) * b$

```

  by(blast intro!:mult-mono)
}
ultimately have  $P s * wp f Q s + (I - P s) * wp g Q s \leq$ 
   $P s * b + (I - P s) * b$ 
  by(blast intro:add-mono)
also have  $\dots = b$  by(auto simp:algebra-simps)
finally show  $P s * wp f Q s + (I - P s) * wp g Q s \leq b$  .
next

```

Monotonic:

```

fix  $Q R::'s \Rightarrow real$  and  $s$ 
assume  $sQ$ : sound Q and  $sR$ : sound R and  $le$ :  $Q \Vdash R$ 

with  $hf$  have  $wp f Q s \leq wp f R s$  by(blast dest:mono-transD)
with  $uP$  have  $P s * wp f Q s \leq P s * wp f R s$ 
  by(auto intro:mult-left-mono)
moreover {
  from  $sQ sR le hg$ 
  have  $wp g Q s \leq wp g R s$  by(blast dest:mono-transD)
  moreover from  $uP$  have  $0 \leq I - P s$ 
  by auto
  ultimately have  $(I - P s) * wp g Q s \leq (I - P s) * wp g R s$ 
  by(auto intro:mult-left-mono)
}
ultimately show  $P s * wp f Q s + (I - P s) * wp g Q s \leq$ 
   $P s * wp f R s + (I - P s) * wp g R s$  by(auto)
next

```

Scaling:

```

fix  $Q::'s \Rightarrow real$  and  $c::real$  and  $s::'s$ 
assume  $sQ$ : sound Q and  $pos$ :  $0 \leq c$ 
have  $c * (P s * wp f Q s + (I - P s) * wp g Q s) =$ 
   $P s * (c * wp f Q s) + (I - P s) * (c * wp g Q s)$ 
  by(simp add:distrib-left)
also have  $\dots = P s * wp f (\lambda s. c * Q s) s +$ 
   $(I - P s) * wp g (\lambda s. c * Q s) s$ 
  using  $hf hg sQ pos$ 
  by(simp add:scalingD[OF healthy-scalingD])
finally show  $c * (P s * wp f Q s + (I - P s) * wp g Q s) =$ 
   $P s * wp f (\lambda s. c * Q s) s + (I - P s) * wp g (\lambda s. c * Q s) s$  .
qed

```

lemma *nearly-healthy-wlp-PC*:

```

fixes  $f::'s prog$ 
assumes  $hf$ : nearly-healthy (wlp f)
  and  $hg$ : nearly-healthy (wlp g)
  and  $uP$ : unitary P
shows nearly-healthy (wlp (f p⊕ g))
proof(intro nearly-healthyI unitaryI2 nnegI bounded-byI le-funI,

```

```

    simp-all add:wp-eval)
  fix Q::'s expect and s::'s
  assume uQ: unitary Q
  from uQ hf hg have utQ: unitary (wlp f Q) unitary (wlp g Q) by(auto)
  from uP have nnP: 0 ≤ P s 0 ≤ 1 - P s
  by auto
  moreover from utQ have 0 ≤ wlp f Q s 0 ≤ wlp g Q s by(auto)
  ultimately show 0 ≤ P s * wlp f Q s + (1 - P s) * wlp g Q s
  by(auto intro:add-nonneg-nonneg mult-nonneg-nonneg)

  from utQ have wlp f Q s ≤ 1 wlp g Q s ≤ 1 by(auto)
  with nnP have P s * wlp f Q s + (1 - P s) * wlp g Q s ≤ P s * 1 + (1 - P s) * 1
  by(blast intro:add-mono mult-left-mono)
  thus P s * wlp f Q s + (1 - P s) * wlp g Q s ≤ 1 by(simp)

  fix R::'s expect
  assume uR: unitary R and le: Q ⊢ R
  with uQ have wlp f Q s ≤ wlp f R s
  by(auto intro:le-funD[OF nearly-healthy-monoD, OF hf])
  with nnP have P s * wlp f Q s ≤ P s * wlp f R s
  by(auto intro:mult-left-mono)
  moreover {
    from uQ uR le have wlp g Q s ≤ wlp g R s
    by(auto intro:le-funD[OF nearly-healthy-monoD, OF hg])
    with nnP have (1 - P s) * wlp g Q s ≤ (1 - P s) * wlp g R s
    by(auto intro:mult-left-mono)
  }
  ultimately show P s * wlp f Q s + (1 - P s) * wlp g Q s ≤
    P s * wlp f R s + (1 - P s) * wlp g R s
  by(auto)
qed

lemma healthy-wp-DC:
  fixes f::'s prog
  assumes hf: healthy (wp f) and hg: healthy (wp g)
  shows healthy (wp (f □ g))
  proof(intro healthy-parts bounded-byI nnegI le-funI, simp-all only:wp-eval)
    fix b and P::'s ⇒ real and s::'s
    assume nP: nneg P and bP: bounded-by b P

    with hf have bounded-by b (wp f P) by(auto)
    hence wp f P s ≤ b by(blast)
    thus min (wp f P s) (wp g P s) ≤ b by(auto)

    from nP bP assms show 0 ≤ min (wp f P s) (wp g P s) by(auto)
  next
    fix P::'s ⇒ real and Q and s::'s
    from assms have mf: mono-trans (wp f) and mg: mono-trans (wp g) by(auto)
    assume sP: sound P and sQ: sound Q and le: P ⊢ Q

```

hence $wp\ f\ P\ s \leq wp\ f\ Q\ s$ **and** $wp\ g\ P\ s \leq wp\ g\ Q\ s$
by(*auto intro:le-funD[OF mono-transD[OF mf]] le-funD[OF mono-transD[OF mg]]*)
thus $\min (wp\ f\ P\ s) (wp\ g\ P\ s) \leq \min (wp\ f\ Q\ s) (wp\ g\ Q\ s)$ **by**(*auto*)
next
fix $P::'s \Rightarrow \text{real}$ **and** $c::\text{real}$ **and** $s::'s$
assume sP : *sound P* **and** pos : $0 \leq c$
from *assms* **have** sf : *scaling (wp f)* **and** sg : *scaling (wp g)* **by**(*auto*)
from pos **have** $c * \min (wp\ f\ P\ s) (wp\ g\ P\ s) =$
 $\min (c * wp\ f\ P\ s) (c * wp\ g\ P\ s)$
by(*simp add:min-distrib*)
also from sP **and** pos
have $\dots = \min (wp\ f\ (\lambda s. c * P\ s)\ s) (wp\ g\ (\lambda s. c * P\ s)\ s)$
by(*simp add:scalingD[OF sf] scalingD[OF sg]*)
finally show $c * \min (wp\ f\ P\ s) (wp\ g\ P\ s) =$
 $\min (wp\ f\ (\lambda s. c * P\ s)\ s) (wp\ g\ (\lambda s. c * P\ s)\ s)$.
qed

lemma *nearly-healthy-wlp-DC*:

fixes $f::'s\ \text{prog}$
assumes hf : *nearly-healthy (wlp f)*
and hg : *nearly-healthy (wlp g)*
shows *nearly-healthy (wlp (f \sqcap g))*
proof(*intro nearly-healthyI bounded-byI nnegI le-funI unitaryI2,*
simp-all add:wp-eval, safe)
fix $P::'s \Rightarrow \text{real}$ **and** $s::'s$
assume uP : *unitary P*
with $hf\ hg$ **have** uP : *unitary (wlp f P)* *unitary (wlp g P)* **by**(*auto*)
thus $0 \leq wlp\ f\ P\ s$ $0 \leq wlp\ g\ P\ s$ **by**(*auto*)

have $\min (wlp\ f\ P\ s) (wlp\ g\ P\ s) \leq wlp\ f\ P\ s$ **by**(*auto*)
also from uP **have** $\dots \leq 1$ **by**(*auto*)
finally show $\min (wlp\ f\ P\ s) (wlp\ g\ P\ s) \leq 1$.

fix $Q::'s \Rightarrow \text{real}$
assume uQ : *unitary Q* **and** le : $P \vdash Q$
have $\min (wlp\ f\ P\ s) (wlp\ g\ P\ s) \leq wlp\ f\ P\ s$ **by**(*auto*)
also from $uP\ uQ\ le$ **have** $\dots \leq wlp\ f\ Q\ s$
by(*auto intro:le-funD[OF nearly-healthy-monoD, OF hf]*)
finally show $\min (wlp\ f\ P\ s) (wlp\ g\ P\ s) \leq wlp\ f\ Q\ s$.

have $\min (wlp\ f\ P\ s) (wlp\ g\ P\ s) \leq wlp\ g\ P\ s$ **by**(*auto*)
also from $uP\ uQ\ le$ **have** $\dots \leq wlp\ g\ Q\ s$
by(*auto intro:le-funD[OF nearly-healthy-monoD, OF hg]*)
finally show $\min (wlp\ f\ P\ s) (wlp\ g\ P\ s) \leq wlp\ g\ Q\ s$.
qed

lemma *healthy-wp-AC*:

fixes $f::'s\ \text{prog}$
assumes hf : *healthy (wp f)* **and** hg : *healthy (wp g)*

shows *healthy* ($wp\ f\ \sqcup\ g$)
proof(*intro healthy-parts bounded-byI nnegI le-funI, simp-all only:wp-eval*)
fix b **and** $P::'s \Rightarrow \text{real}$ **and** $s::'s$
assume $nP: \text{nneg } P$ **and** $bP: \text{bounded-by } b\ P$

with hf **have** *bounded-by* b ($wp\ f\ P$) **by**(*auto*)
hence $wp\ f\ P\ s \leq b$ **by**(*blast*)
moreover {
from $bP\ nP\ hg$ **have** *bounded-by* b ($wp\ g\ P$) **by**(*auto*)
hence $wp\ g\ P\ s \leq b$ **by**(*blast*)
}

ultimately show $\max (wp\ f\ P\ s) (wp\ g\ P\ s) \leq b$ **by**(*auto*)

from $nP\ bP\ \text{assms}$ **have** $0 \leq wp\ f\ P\ s$ **by**(*auto*)
thus $0 \leq \max (wp\ f\ P\ s) (wp\ g\ P\ s)$ **by**(*auto*)

next
fix $P::'s \Rightarrow \text{real}$ **and** Q **and** $s::'s$
from *assms* **have** $mf: \text{mono-trans } (wp\ f)$ **and** $mg: \text{mono-trans } (wp\ g)$ **by**(*auto*)
assume $sP: \text{sound } P$ **and** $sQ: \text{sound } Q$ **and** $le: P \Vdash Q$
hence $wp\ f\ P\ s \leq wp\ f\ Q\ s$ **and** $wp\ g\ P\ s \leq wp\ g\ Q\ s$
by(*auto intro:le-funD[OF mono-transD, OF mf] le-funD[OF mono-transD, OF mg]*)
thus $\max (wp\ f\ P\ s) (wp\ g\ P\ s) \leq \max (wp\ f\ Q\ s) (wp\ g\ Q\ s)$ **by**(*auto*)

next
fix $P::'s \Rightarrow \text{real}$ **and** $c::\text{real}$ **and** $s::'s$
assume $sP: \text{sound } P$ **and** $pos: 0 \leq c$
from *assms* **have** $sf: \text{scaling } (wp\ f)$ **and** $sg: \text{scaling } (wp\ g)$ **by**(*auto*)
from pos **have** $c * \max (wp\ f\ P\ s) (wp\ g\ P\ s) =$
 $\max (c * wp\ f\ P\ s) (c * wp\ g\ P\ s)$
by(*simp add:max-distrib*)
also from sP **and** pos
have $\dots = \max (wp\ f\ (\lambda s. c * P\ s)\ s) (wp\ g\ (\lambda s. c * P\ s)\ s)$
by(*simp add:scalingD[OF sf] scalingD[OF sg]*)
finally show $c * \max (wp\ f\ P\ s) (wp\ g\ P\ s) =$
 $\max (wp\ f\ (\lambda s. c * P\ s)\ s) (wp\ g\ (\lambda s. c * P\ s)\ s)$.

qed

lemma *nearly-healthy-wlp-AC*:

fixes $f::'s\ \text{prog}$
assumes $hf: \text{nearly-healthy } (wlp\ f)$
and $hg: \text{nearly-healthy } (wlp\ g)$
shows *nearly-healthy* ($wlp\ (f\ \sqcup\ g)$)
proof(*intro nearly-healthyI bounded-byI nnegI unitaryI2 le-funI, simp-all only:wp-eval*)
fix b **and** $P::'s \Rightarrow \text{real}$ **and** $s::'s$
assume $uP: \text{unitary } P$

with hf **have** $wlp\ f\ P\ s \leq 1$ **by**(*auto*)
moreover from $uP\ hg$ **have** *unitary* ($wlp\ g\ P$) **by**(*auto*)
hence $wlp\ g\ P\ s \leq 1$ **by**(*auto*)
ultimately show $\max (wlp\ f\ P\ s) (wlp\ g\ P\ s) \leq 1$ **by**(*auto*)

from uP hf **have** $unitary (wlp\ f\ P)$ **by**($auto$)
hence $0 \leq wlp\ f\ P\ s$ **by**($auto$)
thus $0 \leq \max (wlp\ f\ P\ s) (wlp\ g\ P\ s)$ **by**($auto$)
next
fix $P::'s \Rightarrow real$ **and** Q **and** $s::'s$
assume uP : $unitary\ P$ **and** uQ : $unitary\ Q$ **and** le : $P \Vdash Q$
hence $wlp\ f\ P\ s \leq wlp\ f\ Q\ s$ **and** $wlp\ g\ P\ s \leq wlp\ g\ Q\ s$
by($auto$ $intro$: le - $funD$ [OF $nearly$ - $healthy$ - $monoD$, OF hf]
 le - $funD$ [OF $nearly$ - $healthy$ - $monoD$, OF hg])
thus $\max (wlp\ f\ P\ s) (wlp\ g\ P\ s) \leq \max (wlp\ f\ Q\ s) (wlp\ g\ Q\ s)$ **by**($auto$)
qed

lemma $healthy$ - wp - $Embed$:
 $healthy\ t \Longrightarrow healthy (wp (Embed\ t))$
unfolding wp - def $Embed$ - def **by**($simp$)

lemma $nearly$ - $healthy$ - wlp - $Embed$:
 $nearly$ - $healthy\ t \Longrightarrow nearly$ - $healthy (wlp (Embed\ t))$
unfolding wlp - def $Embed$ - def **by**($simp$)

lemma $healthy$ - wp - $repeat$:
assumes h - a : $healthy (wp\ a)$
shows $healthy (wp (repeat\ n\ a))$ (**is** $?X\ n$)
proof($induct\ n$)
show $?X\ 0$ **by**($auto\ simp$: wp - $eval$)
next
fix n **assume** IH : $?X\ n$
thus $?X (Suc\ n)$ **by**($simp\ add$: $healthy$ - wp - $Seq\ h$ - a)
qed

lemma $nearly$ - $healthy$ - wlp - $repeat$:
assumes h - a : $nearly$ - $healthy (wlp\ a)$
shows $nearly$ - $healthy (wlp (repeat\ n\ a))$ (**is** $?X\ n$)
proof($induct\ n$)
show $?X\ 0$ **by**($simp\ add$: wp - $eval$)
next
fix n **assume** IH : $?X\ n$
thus $?X (Suc\ n)$ **by**($simp\ add$: $nearly$ - $healthy$ - wlp - $Seq\ h$ - a)
qed

lemma $healthy$ - wp - $SetDC$:
fixes $prog::'b \Rightarrow 'a\ prog$ **and** $S::'a \Rightarrow 'b\ set$
assumes $healthy$: $\bigwedge x\ s. x \in S\ s \Longrightarrow healthy (wp (prog\ x))$
and $nonempty$: $\bigwedge s. \exists x. x \in S\ s$
shows $healthy (wp (SetDC\ prog\ S))$ (**is** $healthy\ ?T$)
proof($intro\ healthy$ - $parts\ bounded$ - $byI\ nnegI\ le$ - $funI$, $simp$ - $all\ only$: wp - $eval$)
fix b **and** $P::'a \Rightarrow real$ **and** $s::'a$
assume bP : $bounded$ - $by\ b\ P$ **and** nP : $nneg\ P$

hence sP : *sound* P **by**(*auto*)

from *nonempty* **obtain** x **where** xin : $x \in (\lambda a. wp (prog a) P s) ' S s$ **by**(*blast*)

moreover from sP and *healthy*

have $\forall x \in (\lambda a. wp (prog a) P s) ' S s. 0 \leq x$ **by**(*auto*)

ultimately have $Inf ((\lambda a. wp (prog a) P s) ' S s) \leq x$

by(*intro cInf-lower bdd-belowI, auto*)

also from xin **and** *healthy* **and** sP **and** bP **have** $x \leq b$ **by**(*blast*)

finally show $Inf ((\lambda a. wp (prog a) P s) ' S s) \leq b$.

from xin **and** sP **and** *healthy*

show $0 \leq Inf ((\lambda a. wp (prog a) P s) ' S s)$ **by**(*blast intro:cInf-greatest*)

next

fix $P::'a \Rightarrow real$ **and** Q **and** $s::'a$

assume sP : *sound* P **and** sQ : *sound* Q **and** le : $P \Vdash Q$

from *nonempty* **obtain** x **where** xin : $x \in (\lambda a. wp (prog a) P s) ' S s$ **by**(*blast*)

moreover from sP and *healthy*

have $\forall x \in (\lambda a. wp (prog a) P s) ' S s. 0 \leq x$ **by**(*auto*)

moreover

have $\forall x \in (\lambda a. wp (prog a) Q s) ' S s. \exists y \in (\lambda a. wp (prog a) P s) ' S s. y \leq x$

proof(*rule ballI, clarify, rule bexI*)

fix x **and** a **assume** ain : $a \in S s$

with *healthy* **and** sP **and** sQ **and** le **show** $wp (prog a) P s \leq wp (prog a) Q s$

by(*auto dest:mono-transD[OF healthy-monoD]*)

from ain **show** $wp (prog a) P s \in (\lambda a. wp (prog a) P s) ' S s$ **by**(*simp*)

qed

ultimately

show $Inf ((\lambda a. wp (prog a) P s) ' S s) \leq Inf ((\lambda a. wp (prog a) Q s) ' S s)$

by(*intro cInf-mono, blast+*)

next

fix $P::'a \Rightarrow real$ **and** $c::real$ **and** $s::'a$

assume sP : *sound* P **and** pos : $0 \leq c$

from *nonempty* **obtain** x **where** xin : $x \in (\lambda a. wp (prog a) P s) ' S s$ **by**(*blast*)

have $c * Inf ((\lambda a. wp (prog a) P s) ' S s) =$

$Inf ((*) c ' ((\lambda a. wp (prog a) P s) ' S s))$ (**is** $?U = ?V$)

proof(*rule antisym*)

show $?U \leq ?V$

proof(*rule cInf-greatest*)

from *nonempty* **show** $(*) c ' (\lambda a. wp (prog a) P s) ' S s \neq \{\}$ **by**(*auto*)

fix x **assume** $x \in (*) c ' (\lambda a. wp (prog a) P s) ' S s$

then obtain y **where** yin : $y \in (\lambda a. wp (prog a) P s) ' S s$ **and** rx : $x = c * y$ **by**(*auto*)

have $Inf ((\lambda a. wp (prog a) P s) ' S s) \leq y$

proof(*intro cInf-lower[OF yin] bdd-belowI*)

fix z **assume** zin : $z \in (\lambda a. wp (prog a) P s) ' S s$

then obtain a **where** $a \in S s$ **and** $z = wp (prog a) P s$ **by**(*auto*)

with sP **show** $0 \leq z$ **by**(*auto dest:healthy*)

qed

with pos rx **show** $c * Inf ((\lambda a. wp (prog a) P s) ' S s) \leq x$ **by**(*auto intro:mult-left-mono*)

```

qed
show  $?V \leq ?U$ 
proof(cases)
  assume  $cz: c = 0$ 
  moreover {
    from nonempty obtain  $c$  where  $c \in S s$  by(auto)
    hence  $\exists x. \exists xa \in S s. x = wp (prog xa) P s$  by(auto)
  }
  ultimately show ?thesis by(simp add:image-def)
next
assume  $c \neq 0$ 
from nonempty have  $S s \neq \{\}$  by blast
then have  $inverse\ c * (INF\ x \in S\ s. c * wp (prog\ x)\ P\ s) \leq (INF\ a \in S\ s. wp (prog\ a)\ P\ s)$ 
proof (rule cINF-greatest)
  fix  $x$ 
  assume  $x \in S s$ 
  have bdd-below  $((\lambda x. c * wp (prog\ x)\ P\ s) ' S\ s)$ 
  proof (rule bdd-belowI [of - 0])
    fix  $z$ 
    assume  $z \in (\lambda x. c * wp (prog\ x)\ P\ s) ' S\ s$ 
    then obtain  $b$  where  $b \in S s$  and  $rwz: z = c * wp (prog\ b)\ P\ s$  by auto
    with sP have  $0 \leq wp (prog\ b)\ P\ s$  by (auto dest: healthy)
    with pos show  $0 \leq z$  by (auto simp: rwz intro: mult-nonneg-nonneg)
  qed
  then have  $(INF\ x \in S\ s. c * wp (prog\ x)\ P\ s) \leq c * wp (prog\ x)\ P\ s$ 
  using  $\langle x \in S s \rangle$  by (rule cINF-lower)
  with  $\langle c \neq 0 \rangle$  show  $inverse\ c * (INF\ x \in S\ s. c * wp (prog\ x)\ P\ s) \leq wp (prog\ x)\ P\ s$ 
  by (simp add: mult-div-mono-left pos)
qed
with  $\langle c \neq 0 \rangle$  have  $inverse\ c * ?V \leq inverse\ c * ?U$ 
by (simp add: mult.assoc [symmetric] image-comp)
with pos have  $c * (inverse\ c * ?V) \leq c * (inverse\ c * ?U)$ 
by(auto intro:mult-left-mono)
with  $\langle c \neq 0 \rangle$  show ?thesis by (simp add:mult.assoc [symmetric])
qed
qed
also have  $\dots = Inf ((\lambda a. c * wp (prog\ a)\ P\ s) ' S\ s)$ 
by (simp add: image-comp)
also from sP and pos have  $\dots = Inf ((\lambda a. wp (prog\ a)\ (\lambda s. c * P\ s)\ s) ' S\ s)$ 
by(simp add:scalingD[OF healthy-scalingD, OF healthy] cong:image-cong)
finally show  $c * Inf ((\lambda a. wp (prog\ a)\ P\ s) ' S\ s) =$ 
   $Inf ((\lambda a. wp (prog\ a)\ (\lambda s. c * P\ s)\ s) ' S\ s) .$ 
qed

lemma nearly-healthy-wlp-SetDC:
fixes prog:: $'b \Rightarrow 'a\ prog$  and S:: $'a \Rightarrow 'b\ set$ 
assumes healthy:  $\bigwedge x\ s. x \in S\ s \implies\ nearly\text{-healthy}\ (wlp\ (prog\ x))$ 
and nonempty:  $\bigwedge s. \exists x. x \in S\ s$ 
shows nearly-healthy  $(wlp\ (SetDC\ prog\ S))$  (is nearly-healthy ?T)

```


proof(*intro nearly-healthyI unitaryI2 bounded-byI nnegI le-funI, simp-all only:wp-eval*)
fix b **and** $P::'a \Rightarrow \text{real}$ **and** $s::'a$
assume uP : unitary P

from *nonempty* **obtain** x **where** xin : $x \in (\lambda a. \text{wlp } (prog \ a) \ P \ s) \ 'S \ s$ **by**(*blast*)
moreover {
from uP *healthy*
have $\forall x \in (\lambda a. \text{wlp } (prog \ a) \ P) \ 'S \ s. \text{unitary } x$ **by**(*auto*)
hence $\forall x \in (\lambda a. \text{wlp } (prog \ a) \ P) \ 'S \ s. 0 \leq x \ s$ **by**(*auto*)
hence $\forall y \in (\lambda a. \text{wlp } (prog \ a) \ P \ s) \ 'S \ s. 0 \leq y$ **by**(*auto*)
}
ultimately have $\text{Inf } ((\lambda a. \text{wlp } (prog \ a) \ P \ s) \ 'S \ s) \leq x$ **by**(*intro cInf-lower bdd-belowI, auto*)
also from xin *healthy* uP **have** $x \leq 1$ **by**(*blast*)
finally show $\text{Inf } ((\lambda a. \text{wlp } (prog \ a) \ P \ s) \ 'S \ s) \leq 1$.

from xin uP *healthy*
show $0 \leq \text{Inf } ((\lambda a. \text{wlp } (prog \ a) \ P \ s) \ 'S \ s)$
by(*blast dest!:unitary-sound[OF nearly-healthy-unitaryD[OF - uP]]*
intro:cInf-greatest)

next
fix $P::'a \Rightarrow \text{real}$ **and** Q **and** $s::'a$
assume uP : unitary P **and** uQ : unitary Q **and** le : $P \Vdash Q$

from *nonempty* **obtain** x **where** xin : $x \in (\lambda a. \text{wlp } (prog \ a) \ P \ s) \ 'S \ s$ **by**(*blast*)
moreover {
from uP *healthy*
have $\forall x \in (\lambda a. \text{wlp } (prog \ a) \ P) \ 'S \ s. \text{unitary } x$ **by**(*auto*)
hence $\forall x \in (\lambda a. \text{wlp } (prog \ a) \ P) \ 'S \ s. 0 \leq x \ s$ **by**(*auto*)
hence $\forall y \in (\lambda a. \text{wlp } (prog \ a) \ P \ s) \ 'S \ s. 0 \leq y$ **by**(*auto*)
}
moreover
have $\forall x \in (\lambda a. \text{wlp } (prog \ a) \ Q \ s) \ 'S \ s. \exists y \in (\lambda a. \text{wlp } (prog \ a) \ P \ s) \ 'S \ s. y \leq x$
proof(*rule ballI, clarify, rule bexI*)
fix x **and** a **assume** ain : $a \in S \ s$
from $uP \ uQ \ le$ **show** $\text{wlp } (prog \ a) \ P \ s \leq \text{wlp } (prog \ a) \ Q \ s$
by(*auto intro:le-funD[OF nearly-healthy-monoD[OF healthy, OF ain]]*)
from ain **show** $\text{wlp } (prog \ a) \ P \ s \in (\lambda a. \text{wlp } (prog \ a) \ P \ s) \ 'S \ s$ **by**(*simp*)
qed
ultimately
show $\text{Inf } ((\lambda a. \text{wlp } (prog \ a) \ P \ s) \ 'S \ s) \leq \text{Inf } ((\lambda a. \text{wlp } (prog \ a) \ Q \ s) \ 'S \ s)$
by(*intro cInf-mono, blast+*)
qed

lemma *healthy-wp-SetPC*:
fixes $p::'s \Rightarrow 'a \Rightarrow \text{real}$
and $f::'a \Rightarrow 's \ \text{prog}$
assumes *healthy*: $\bigwedge a \ s. a \in \text{supp } (p \ s) \Longrightarrow \text{healthy } (wp \ (f \ a))$
and *sound*: $\bigwedge s. \text{sound } (p \ s)$

and sub-dist: $\bigwedge s. (\sum a \in \text{supp}(p\ s). p\ s\ a) \leq 1$
shows healthy ($\text{wp}(\text{SetPC}\ f\ p)$) (**is healthy** ?X)
proof(*intro healthy-parts bounded-byI nnegI le-funI, simp-all add:wp-eval*)
fix b **and** $P::'s \Rightarrow \text{real}$ **and** $s::'s$
assume bP : *bounded-by* $b\ P$ **and** nP : *nneg* P
hence sP : *sound* P **by**(*auto*)

from sP **and** bP **and healthy** **have** $\bigwedge a\ s. a \in \text{supp}(p\ s) \implies \text{wp}(f\ a)\ P\ s \leq b$
by(*blast dest:healthy-bounded-byD*)
with sound **have** $(\sum a \in \text{supp}(p\ s). p\ s\ a * \text{wp}(f\ a)\ P\ s) \leq (\sum a \in \text{supp}(p\ s). p\ s\ a * b)$
by(*blast intro:sum-mono mult-left-mono*)
also have $\dots = (\sum a \in \text{supp}(p\ s). p\ s\ a) * b$
by(*simp add:sum-distrib-right*)
also {
from bP **and** nP **have** $0 \leq b$ **by**(*blast*)
with sub-dist **have** $(\sum a \in \text{supp}(p\ s). p\ s\ a) * b \leq 1 * b$
by(*rule mult-right-mono*)
}

also have $1 * b = b$ **by**(*simp*)
finally show $(\sum a \in \text{supp}(p\ s). p\ s\ a * \text{wp}(f\ a)\ P\ s) \leq b$.

show $0 \leq (\sum a \in \text{supp}(p\ s). p\ s\ a * \text{wp}(f\ a)\ P\ s)$
proof(*rule sum-nonneg [OF mult-nonneg-nonneg]*)
fix x
from sound **show** $0 \leq p\ s\ x$ **by**(*blast*)
assume $x \in \text{supp}(p\ s)$ **with** sP **and healthy**
show $0 \leq \text{wp}(f\ x)\ P\ s$ **by**(*blast*)
qed

next
fix $P::'s \Rightarrow \text{real}$ **and** $Q::'s \Rightarrow \text{real}$ **and** s
assume $s\text{-}P$: *sound* P **and** $s\text{-}Q$: *sound* Q **and** ent : $P \Vdash Q$
with healthy **have** $\bigwedge a. a \in \text{supp}(p\ s) \implies \text{wp}(f\ a)\ P\ s \leq \text{wp}(f\ a)\ Q\ s$
by(*blast*)
with sound **show** $(\sum a \in \text{supp}(p\ s). p\ s\ a * \text{wp}(f\ a)\ P\ s) \leq$
 $(\sum a \in \text{supp}(p\ s). p\ s\ a * \text{wp}(f\ a)\ Q\ s)$
by(*blast intro:sum-mono mult-left-mono*)

next
fix $P::'s \Rightarrow \text{real}$ **and** $c::\text{real}$ **and** $s::'s$
assume *sound*: *sound* P **and** *pos*: $0 \leq c$
have $c * (\sum a \in \text{supp}(p\ s). p\ s\ a * \text{wp}(f\ a)\ P\ s) =$
 $(\sum a \in \text{supp}(p\ s). p\ s\ a * (c * \text{wp}(f\ a)\ P\ s))$
(is ?A = ?B)
by(*simp add:sum-distrib-left ac-simps*)
also from sound **and** *pos* **and healthy**
have $\dots = (\sum a \in \text{supp}(p\ s). p\ s\ a * \text{wp}(f\ a)\ (\lambda s. c * P\ s)\ s)$
by(*auto simp:scalingD[OF healthy-scalingD]*)
finally show ?A =
qed

lemma *nearly-healthy-wlp-SetPC*:

fixes $p::'s \Rightarrow 'a \Rightarrow \text{real}$
and $f::'a \Rightarrow 's \text{ prog}$
assumes *healthy*: $\bigwedge a s. a \in \text{supp } (p s) \Longrightarrow \text{nearly-healthy } (wlp (f a))$
and *sound*: $\bigwedge s. \text{sound } (p s)$
and *sub-dist*: $\bigwedge s. (\sum a \in \text{supp } (p s). p s a) \leq 1$
shows *nearly-healthy* (wlp (SetPC f p)) (**is** *nearly-healthy ?X*)
proof(intro *nearly-healthyI unitaryI2 bounded-byI nnegI le-funI, simp-all only:wp-eval*)
fix b **and** $P::'s \Rightarrow \text{real}$ **and** $s::'s$
assume uP : *unitary P*

from uP *healthy* **have** $\bigwedge a. a \in \text{supp } (p s) \Longrightarrow \text{unitary } (wlp (f a) P)$ **by**(*auto*)
hence $\bigwedge a. a \in \text{supp } (p s) \Longrightarrow wlp (f a) P s \leq 1$ **by**(*auto*)
with *sound* **have** $(\sum a \in \text{supp } (p s). p s a * wlp (f a) P s) \leq (\sum a \in \text{supp } (p s). p s a * 1)$
by(*blast intro:sum-mono mult-left-mono*)
also **have** $\dots = (\sum a \in \text{supp } (p s). p s a)$
by(*simp add:sum-distrib-right*)
also **note** *sub-dist*
finally **show** $(\sum a \in \text{supp } (p s). p s a * wlp (f a) P s) \leq 1$.
show $0 \leq (\sum a \in \text{supp } (p s). p s a * wlp (f a) P s)$
proof(*rule sum-nonneg [OF mult-nonneg-nonneg]*)
fix x
from *sound* **show** $0 \leq p s x$ **by**(*blast*)
assume $x \in \text{supp } (p s)$ **with** uP *healthy*
show $0 \leq wlp (f x) P s$ **by**(*blast*)
qed

next
fix $P::'s \text{ expect}$ **and** $Q::'s \text{ expect}$ **and** s
assume uP : *unitary P* **and** uQ : *unitary Q* **and** le : $P \Vdash Q$
hence $\bigwedge a. a \in \text{supp } (p s) \Longrightarrow wlp (f a) P s \leq wlp (f a) Q s$
by(*blast intro:le-funD[OF nearly-healthy-monoD, OF healthy]*)
with *sound* **show** $(\sum a \in \text{supp } (p s). p s a * wlp (f a) P s) \leq$
 $(\sum a \in \text{supp } (p s). p s a * wlp (f a) Q s)$
by(*blast intro:sum-mono mult-left-mono*)
qed

lemma *healthy-wp-Apply*:

healthy (wp (Apply f))
unfolding *Apply-def wp-def* **by**(*blast*)

lemma *nearly-healthy-wlp-Apply*:

nearly-healthy (wlp (Apply f))
by(intro *nearly-healthyI unitaryI2 nnegI bounded-byI, auto simp:o-def wp-eval*)

lemma *healthy-wp-Bind*:

fixes $f::'s \Rightarrow 'a$
assumes *hsub*: $\bigwedge s. \text{healthy } (wp (p (f s)))$
shows *healthy* (wp (Bind f p))
proof(intro *healthy-parts nnegI bounded-byI le-funI, simp-all only:wp-eval*)

```

fix  $b$  and  $P::'s$  expect and  $s::'s$ 
assume  $bP$ : bounded-by  $b$   $P$  and  $nP$ : nneg  $P$ 
with  $hsub$  have bounded-by  $b$  ( $wp$  ( $p$  ( $f$   $s$ ))  $P$ ) by(auto)
thus  $wp$  ( $p$  ( $f$   $s$ ))  $P$   $s \leq b$  by(auto)
from  $bP$   $nP$   $hsub$  have nneg ( $wp$  ( $p$  ( $f$   $s$ ))  $P$ ) by(auto)
thus  $0 \leq wp$  ( $p$  ( $f$   $s$ ))  $P$   $s$  by(auto)
next
fix  $P$   $Q::'s$  expect and  $s::'s$ 
assume sound  $P$  sound  $Q$   $P \Vdash Q$ 
thus  $wp$  ( $p$  ( $f$   $s$ ))  $P$   $s \leq wp$  ( $p$  ( $f$   $s$ ))  $Q$   $s$ 
by(rule le-funD[OF mono-transD, OF healthy-monoD, OF hsub])
next
fix  $P::'s$  expect and  $c::real$  and  $s::'s$ 
assume sound  $P$  and  $0 \leq c$ 
thus  $c * wp$  ( $p$  ( $f$   $s$ ))  $P$   $s = wp$  ( $p$  ( $f$   $s$ )) ( $\lambda s. c * P$   $s$ )  $s$ 
by(simp add:scalingD[OF healthy-scalingD, OF hsub])
qed

lemma nearly-healthy-wlp-Bind:
fixes  $f::'s \Rightarrow 'a$ 
assumes  $hsub$ :  $\bigwedge s. \text{nearly-healthy}$  ( $wlp$  ( $p$  ( $f$   $s$ )))
shows nearly-healthy ( $wlp$  ( $Bind$   $f$   $p$ ))
proof(intro nearly-healthyI unitaryI2 nnegI bounded-byI le-funI, simp-all only:wp-eval)
fix  $P::'s$  expect and  $s::'s$  assume  $uP$ : unitary  $P$ 
with  $hsub$  have unitary ( $wlp$  ( $p$  ( $f$   $s$ ))  $P$ ) by(auto)
thus  $0 \leq wlp$  ( $p$  ( $f$   $s$ ))  $P$   $s$   $wlp$  ( $p$  ( $f$   $s$ ))  $P$   $s \leq 1$  by(auto)

fix  $Q::'s$  expect
assume unitary  $Q$   $P \Vdash Q$ 
with  $uP$  show  $wlp$  ( $p$  ( $f$   $s$ ))  $P$   $s \leq wlp$  ( $p$  ( $f$   $s$ ))  $Q$   $s$ 
by(blast intro:le-funD[OF nearly-healthy-monoD, OF hsub])
qed

```

4.2.2 Healthiness for Loops

```

lemma wp-loop-step-mono:
fixes  $t$   $u::'s$  trans
assumes  $hb$ : healthy ( $wp$  body)
and  $le$ : le-trans  $t$   $u$ 
and  $ht$ :  $\bigwedge P. \text{sound } P \implies \text{sound } (t P)$ 
and  $hu$ :  $\bigwedge P. \text{sound } P \implies \text{sound } (u P)$ 
shows le-trans ( $wp$  ( $body$  ;;  $Embed$   $t$   $\ll G \gg \oplus Skip$ ))
( $wp$  ( $body$  ;;  $Embed$   $u$   $\ll G \gg \oplus Skip$ ))
proof(intro le-transI le-funI, simp add:wp-eval)
fix  $P::'s$  expect and  $s::'s$ 
assume  $sP$ : sound  $P$ 
with  $le$  have  $t P \Vdash u P$  by(auto)
moreover from  $sP$   $ht$   $hu$  have sound ( $t P$ ) sound ( $u P$ ) by(auto)
ultimately have  $wp$  body ( $t P$ )  $s \leq wp$  body ( $u P$ )  $s$ 

```

by(*auto intro:le-funD*[*OF mono-transD*, *OF healthy-monoD*, *OF hb*])
thus $\langle\langle G \rangle\rangle s * wp \text{ body } (t P) s \leq \langle\langle G \rangle\rangle s * wp \text{ body } (u P) s$
by(*auto intro:mult-left-mono*)
qed

lemma *wlp-loop-step-mono*:

fixes $t u :: 's \text{ trans}$
assumes *mb: nearly-healthy* (*wlp body*)
and *le: le-utrans* $t u$
and *ht*: $\bigwedge P. \text{unitary } P \implies \text{unitary } (t P)$
and *hu*: $\bigwedge P. \text{unitary } P \implies \text{unitary } (u P)$
shows *le-utrans* (*wlp* (*body* ;; *Embed* $t \langle\langle G \rangle\rangle \oplus \text{Skip}$))
 (*wlp* (*body* ;; *Embed* $u \langle\langle G \rangle\rangle \oplus \text{Skip}$))
proof(*intro le-utransI le-funI, simp add:wp-eval*)
fix $P :: 's \text{ expect}$ **and** $s :: 's$
assume $uP: \text{unitary } P$
with *le* **have** $t P \Vdash u P$ **by**(*auto*)
moreover from uP *ht hu* **have** $\text{unitary } (t P) \text{unitary } (u P)$ **by**(*auto*)
ultimately have $wlp \text{ body } (t P) s \leq wlp \text{ body } (u P) s$
by(*rule le-funD*[*OF nearly-healthy-monoD*[*OF mb*]])
thus $\langle\langle G \rangle\rangle s * wlp \text{ body } (t P) s \leq \langle\langle G \rangle\rangle s * wlp \text{ body } (u P) s$
by(*auto intro:mult-left-mono*)
qed

For each sound expectation, we have a pre fixed point of the loop body. This lets us use the relevant fixed-point lemmas.

lemma *lfp-loop-fp*:

assumes *hb: healthy* (*wp body*)
and *sP: sound* P
shows $\lambda s. \langle\langle G \rangle\rangle s * wp \text{ body } (\lambda s. \text{bound-of } P) s + \langle\langle \mathcal{N} G \rangle\rangle s * P s \Vdash \lambda s. \text{bound-of } P$
proof(*rule le-funI*)
fix s
from *sP* **have** $\text{sound } (\lambda s. \text{bound-of } P)$ **by**(*auto*)
moreover hence *bounded-by* (*bound-of* P) ($\lambda s. \text{bound-of } P$) **by**(*auto*)
ultimately have *bounded-by* (*bound-of* P) (*wp body* ($\lambda s. \text{bound-of } P$))
using *hb* **by**(*auto*)
hence $wp \text{ body } (\lambda s. \text{bound-of } P) s \leq \text{bound-of } P$ **by**(*auto*)
moreover from *sP* **have** $P s \leq \text{bound-of } P$ **by**(*auto*)
ultimately have $\langle\langle G \rangle\rangle s * wp \text{ body } (\lambda s. \text{bound-of } P) s + (1 - \langle\langle G \rangle\rangle s) * P s \leq$
 $\langle\langle G \rangle\rangle s * \text{bound-of } P + (1 - \langle\langle G \rangle\rangle s) * \text{bound-of } P$
by(*blast intro:add-mono mult-left-mono*)
thus $\langle\langle G \rangle\rangle s * wp \text{ body } (\lambda s. \text{bound-of } P) s + \langle\langle \mathcal{N} G \rangle\rangle s * P s \leq \text{bound-of } P$
by(*simp add:algebra-simps negate-embed*)
qed

lemma *lfp-loop-greatest*:

fixes $P :: 's \text{ expect}$
assumes *lb*: $\bigwedge R. \lambda s. \langle\langle G \rangle\rangle s * wp \text{ body } R s + \langle\langle \mathcal{N} G \rangle\rangle s * P s \Vdash R \implies \text{sound } R \implies Q \Vdash R$
and *hb: healthy* (*wp body*)

and sP : *sound* P
and sQ : *sound* Q
shows $Q \Vdash \text{lf}p\text{-exp} (\lambda Q s. \langle G \rangle s * \text{wp body } Q s + \langle \mathcal{N} G \rangle s * P s)$
using sP **by** (*auto intro!*:*lf}p-exp-greatest*[*OF lb sQ*] sP *lf}p-loop-fp* *hb*)

lemma *lf}p-loop-sound*:

fixes P ::'s *expect*
assumes hb : *healthy* (*wp body*)
and sP : *sound* P
shows *sound* (*lf}p-exp* ($\lambda Q s. \langle G \rangle s * \text{wp body } Q s + \langle \mathcal{N} G \rangle s * P s$))
using *assms* **by** (*auto intro!*:*lf}p-exp-sound* *lf}p-loop-fp*)

lemma *wlp-loop-step-unitary*:

fixes $t u$::'s *trans*
assumes hb : *nearly-healthy* (*wlp body*)
and ht : $\bigwedge P. \text{unitary } P \implies \text{unitary } (t P)$
and uP : *unitary* P
shows *unitary* (*wlp* (*body* ;; *Embed* $t \langle G \rangle \oplus \text{Skip}$) P)
proof(*intro unitaryI2 nnegI bounded-byI, simp-all add:wp-eval*)
fix s ::'s
from ht uP **have** utP : *unitary* ($t P$) **by**(*auto*)
with hb **have** *unitary* (*wlp body* ($t P$)) **by**(*auto*)
hence $0 \leq \text{wlp body } (t P) s$ **by**(*auto*)
with uP **show** $0 \leq \langle G \rangle s * \text{wlp body } (t P) s + (1 - \langle G \rangle s) * P s$
by(*auto intro!*:*add-nonneg-nonneg mult-nonneg-nonneg*)
from ht uP **have** *bounded-by 1* ($t P$) **by**(*auto*)
with utP hb **have** *bounded-by 1* (*wlp body* ($t P$)) **by**(*auto*)
hence *wlp body* ($t P$) $s \leq 1$ **by**(*auto*)
with uP **have** $\langle G \rangle s * \text{wlp body } (t P) s + (1 - \langle G \rangle s) * P s \leq \langle G \rangle s * 1 + (1 - \langle G \rangle s)$
 $* 1$
by(*blast intro:add-mono mult-left-mono*)
also have $\dots = 1$ **by**(*simp*)
finally show $\langle G \rangle s * \text{wlp body } (t P) s + (1 - \langle G \rangle s) * P s \leq 1$.
qed

lemma *wp-loop-step-sound*:

fixes $t u$::'s *trans*
assumes hb : *healthy* (*wp body*)
and ht : $\bigwedge P. \text{sound } P \implies \text{sound } (t P)$
and sP : *sound* P
shows *sound* (*wp* (*body* ;; *Embed* $t \langle G \rangle \oplus \text{Skip}$) P)
proof(*intro soundI2 nnegI bounded-byI, simp-all add:wp-eval*)
fix s ::'s
from ht sP **have** stP : *sound* ($t P$) **by**(*auto*)
with hb **have** $0 \leq \text{wp body } (t P) s$ **by**(*auto*)
with sP **show** $0 \leq \langle G \rangle s * \text{wp body } (t P) s + (1 - \langle G \rangle s) * P s$
by(*auto intro!*:*add-nonneg-nonneg mult-nonneg-nonneg*)
from ht sP **have** *sound* ($t P$) **by**(*auto*)

moreover hence *bounded-by* (*bound-of* (*t P*)) (*t P*) **by**(*auto*)
ultimately have *wp body* (*t P*) $s \leq \text{bound-of } (t P)$ **using** *hb* **by**(*auto*)
hence *wp body* (*t P*) $s \leq \max (\text{bound-of } P) (\text{bound-of } (t P))$ **by**(*auto*)
moreover {
 from *sP* **have** $P s \leq \text{bound-of } P$ **by**(*auto*)
 hence $P s \leq \max (\text{bound-of } P) (\text{bound-of } (t P))$ **by**(*auto*)
}

ultimately
have $\langle G \rangle s * \text{wp body } (t P) s + (1 - \langle G \rangle s) * P s \leq$
 $\langle G \rangle s * \max (\text{bound-of } P) (\text{bound-of } (t P)) +$
 $(1 - \langle G \rangle s) * \max (\text{bound-of } P) (\text{bound-of } (t P))$
by(*blast intro:add-mono mult-left-mono*)
also have $\dots = \max (\text{bound-of } P) (\text{bound-of } (t P))$ **by**(*simp add:algebra-simps*)
finally show $\langle G \rangle s * \text{wp body } (t P) s + (1 - \langle G \rangle s) * P s \leq$
 $\max (\text{bound-of } P) (\text{bound-of } (t P))$.

qed

This gives the equivalence with the alternative definition for loops [McIver and Morgan, 2004, §7, p. 198, footnote 23].

lemma *wlp-Loop1*:

fixes *body* :: '*s prog*
assumes *unitary*: *unitary P*
 and *healthy*: *nearly-healthy (wlp body)*
shows *wlp (do G \longrightarrow body od) P =*
 $\text{gfp-exp } (\lambda Q s. \langle G \rangle s * \text{wlp body } Q s + \langle \mathcal{N} G \rangle s * P s)$
 (*is ?X = gfp-exp (?Y P)*)
proof –
 let $?Z u = (\text{body} ;; \text{Embed } u \langle G \rangle \oplus \text{Skip})$
 show *?thesis*
 proof(*simp only: wp-eval, intro gfp-pulldown assms le-funI*)
 fix *u P*
 show *wlp (?Z u) P = ?Y P (u P)* **by**(*simp add:wp-eval negate-embed*)
next
 fix *t::'s trans* **and** *P::'s expect*
 assume *ut*: $\bigwedge Q. \text{unitary } Q \implies \text{unitary } (t Q)$ **and** *uP*: *unitary P*
 thus *unitary (wlp (?Z t) P)*
 by(*rule wlp-loop-step-unitary[OF healthy]*)
next
 fix *P Q::'s expect*
 assume *uP*: *unitary P* **and** *uQ*: *unitary Q*
 show *unitary* ($\lambda a. \langle G \rangle a * \text{wlp body } Q a + \langle \mathcal{N} G \rangle a * P a$)
 proof(*intro unitaryI2 nnegI bounded-byI*)
 fix *s::'s*
 from *healthy uQ*
 have *unitary (wlp body Q)* **by**(*auto*)
 hence $0 \leq \text{wlp body } Q s$ **by**(*auto*)
 with *uP* **show** $0 \leq \langle G \rangle s * \text{wlp body } Q s + \langle \mathcal{N} G \rangle s * P s$
 by(*auto intro!:add-nonneg-nonneg mult-nonneg-nonneg*)

```

from healthy  $uQ$  have bounded-by  $I$  (wlp body  $Q$ ) by(auto)
with  $uP$  have  $\langle\langle G \rangle\rangle s * \text{wlp body } Q s + (I - \langle\langle G \rangle\rangle s) * P s \leq \langle\langle G \rangle\rangle s * I + (I - \langle\langle G \rangle\rangle s)$ 
*  $I$ 
  by(blast intro:add-mono mult-left-mono)
  also have  $\dots = I$  by(simp)
  finally show  $\langle\langle G \rangle\rangle s * \text{wlp body } Q s + \langle\langle \mathcal{N} G \rangle\rangle s * P s \leq I$ 
  by(simp add:negate-embed)
qed
next
fix  $P Q R :: 's \text{ expect and } s :: 's$ 
assume  $uP$ : unitary  $P$  and  $uQ$ : unitary  $Q$  and  $uR$ : unitary  $R$ 
and  $le$ :  $Q \Vdash R$ 
hence wlp body  $Q s \leq \text{wlp body } R s$ 
by(blast intro:le-funD[OF nearly-healthy-monoD, OF healthy])
thus  $\langle\langle G \rangle\rangle s * \text{wlp body } Q s + \langle\langle \mathcal{N} G \rangle\rangle s * P s \leq$ 
 $\langle\langle G \rangle\rangle s * \text{wlp body } R s + \langle\langle \mathcal{N} G \rangle\rangle s * P s$ 
by(auto intro:mult-left-mono)
next
fix  $t u :: 's \text{ trans}$ 
assume le-utrans  $t u$ 
 $\bigwedge P. \text{unitary } P \implies \text{unitary } (t P)$ 
 $\bigwedge P. \text{unitary } P \implies \text{unitary } (u P)$ 
thus le-utrans (wlp  $(?Z t)$ ) (wlp  $(?Z u)$ )
by(blast intro!:wlp-loop-step-mono[OF healthy])
qed
qed

lemma wp-loop-sound:
assumes  $sP$ : sound  $P$ 
and  $hb$ : healthy (wp body)
shows sound (wp do  $G \longrightarrow \text{body od } P$ )
proof(simp only: wp-eval, intro lfp-trans-sound sP)
let  $?v = \lambda P s. \text{bound-of } P$ 
show le-trans (wp (body ;; Embed  $?v \langle\langle G \rangle\rangle \oplus \text{Skip}$ ))  $?v$ 
by(intro le-transI, simp add:wp-eval lfp-loop-fp[unfolded negate-embed] hb)
show  $\bigwedge P. \text{sound } P \implies \text{sound } (?v P)$  by(auto)
qed

```

Likewise, we can rewrite strict loops.

```

lemma wp-LoopI:
fixes body :: 's prog
assumes  $sP$ : sound  $P$ 
and healthy: healthy (wp body)
shows wp (do  $G \longrightarrow \text{body od } P =$ 
lfp-exp  $(\lambda Q s. \langle\langle G \rangle\rangle s * \text{wp body } Q s + \langle\langle \mathcal{N} G \rangle\rangle s * P s)$ 
(is  $?X = \text{lfp-exp } (?Y P)$ )
proof –
let  $?Z u = (\text{body} ;; \text{Embed } u \langle\langle G \rangle\rangle \oplus \text{Skip})$ 
show thesis

```



```

proof(simp only: wp-eval, intro lfp-pulldown assms le-funI sP mono-transI)
  fix u P
  show wp (?Z u) P = ?Y P (u P) by(simp add:wp-eval negate-embed)
next
  fix t::'s trans and P::'s expect
  assume ut:  $\bigwedge Q. \text{sound } Q \implies \text{sound } (t \ Q)$  and uP: sound P
  with healthy show sound (wp (?Z t) P) by(rule wp-loop-step-sound)
next
  fix P Q::'s expect
  assume sP: sound P and sQ: sound Q
  show sound ( $\lambda a. \ll G \gg a * \text{wp body } Q \ a + \ll \mathcal{N} \ G \gg a * P \ a$ )
  proof(intro soundI2 nnegI bounded-byI)
    fix s::'s
    from sQ have nneg Q bounded-by (bound-of Q) Q by(auto)
    with healthy have bounded-by (bound-of Q) (wp body Q) by(auto)
    hence wp body Q s  $\leq$  bound-of Q by(auto)
    hence wp body Q s  $\leq$  max (bound-of P) (bound-of Q) by(auto)
    moreover {
      from sP have P s  $\leq$  bound-of P by(auto)
      hence P s  $\leq$  max (bound-of P) (bound-of Q) by(auto)
    }
    ultimately have  $\ll G \gg s * \text{wp body } Q \ s + \ll \mathcal{N} \ G \gg s * P \ s \leq$ 
       $\ll G \gg s * \max (\text{bound-of } P) (\text{bound-of } Q) +$ 
       $\ll \mathcal{N} \ G \gg s * \max (\text{bound-of } P) (\text{bound-of } Q)$ 
      by(auto intro!:add-mono mult-left-mono)
    also have ... = max (bound-of P) (bound-of Q) by(simp add:algebra-simps negate-embed)
    finally show  $\ll G \gg s * \text{wp body } Q \ s + \ll \mathcal{N} \ G \gg s * P \ s \leq$  max (bound-of P) (bound-of Q)
  .

  from sP have  $0 \leq P \ s$  by(auto)
  moreover from sQ healthy have  $0 \leq \text{wp body } Q \ s$  by(auto)
  ultimately show  $0 \leq \ll G \gg s * \text{wp body } Q \ s + \ll \mathcal{N} \ G \gg s * P \ s$ 
    by(auto intro: add-nonneg-nonneg mult-nonneg-nonneg)
qed
next
  fix P Q R::'s expect and s::'s
  assume sQ: sound Q and sR: sound R
  and le: Q  $\Vdash$  R
  hence wp body Q s  $\leq$  wp body R s
  by(blast intro:le-funD[OF mono-transD, OF healthy-monoD, OF healthy])
  thus  $\ll G \gg s * \text{wp body } Q \ s + \ll \mathcal{N} \ G \gg s * P \ s \leq$ 
     $\ll G \gg s * \text{wp body } R \ s + \ll \mathcal{N} \ G \gg s * P \ s$ 
    by(auto intro:mult-left-mono)
next
  fix t u::'s trans
  assume le: le-trans t u
  and st:  $\bigwedge P. \text{sound } P \implies \text{sound } (t \ P)$ 
  and su:  $\bigwedge P. \text{sound } P \implies \text{sound } (u \ P)$ 
  with healthy show le-trans (wp (?Z t)) (wp (?Z u))

```

```

  by(rule wp-loop-step-mono)
next
from healthy show le-trans (wp (?Z (λP s. bound-of P))) (λP s. bound-of P)
  by(intro le-transI, simp add:wp-eval lfp-loop-fp[unfolded negate-embed])
next
fix P::'s expect and s::'s
assume sound P
thus sound (λs. bound-of P) by(auto)
qed
qed

lemma nearly-healthy-wlp-loop:
fixes body::'s prog
assumes hb: nearly-healthy (wlp body)
shows nearly-healthy (wlp (do G → body od))
proof(intro nearly-healthyI unitaryI2 nnegI2 bounded-byI2, simp-all add:wlp-Loop1 hb)
fix P::'s expect
assume uP: unitary P
let ?X R = λQ s. « G » s * wlp body Q s + « N G » s * R s

show λs. 0 ⊨ gfp-exp (?X P)
proof(rule gfp-exp-upperbound)
show unitary (λs. 0::real) by(auto)
with hb have unitary (wlp body (λs. 0)) by(auto)
with uP show λs. 0 ⊨ (?X P (λs. 0))
  by(blast intro!:le-funI add-nonneg-nonneg mult-nonneg-nonneg)
qed

show gfp-exp (?X P) ⊨ λs. 1
proof(rule gfp-exp-least)
show unitary (λs. 1::real) by(auto)
fix Q::'s expect
assume unitary Q
thus Q ⊨ λs. 1 by(auto)
qed

fix Q::'s expect
assume uQ: unitary Q and le: P ⊨ Q
show gfp-exp (?X P) ⊨ gfp-exp (?X Q)
proof(rule gfp-exp-least)
fix R::'s expect assume uR: unitary R
assume fp: R ⊨ ?X P R
also from le have ... ⊨ ?X Q R
  by(blast intro:add-mono mult-left-mono le-funI)
finally show R ⊨ gfp-exp (?X Q)
  using uR by(auto intro:gfp-exp-upperbound)
next
show unitary (gfp-exp (?X Q))
proof(rule gfp-exp-unitary, intro unitaryI2 nnegI bounded-byI)

```

```

fix  $R::'s$  expect and  $s::'s$  assume  $uR$ : unitary  $R$ 
with  $hb$  have  $ubP$ : unitary (wlp body  $R$ ) by(auto)
with  $uQ$  show  $0 \leq \llbracket G \rrbracket s * \text{wlp body } R s + \llbracket \mathcal{N} G \rrbracket s * Q s$ 
  by(blast intro:add-nonneg-nonneg mult-nonneg-nonneg)

from  $ubP$   $uQ$  have wlp body  $R s \leq 1$   $Q s \leq 1$  by(auto)
hence  $\llbracket G \rrbracket s * \text{wlp body } R s + \llbracket \mathcal{N} G \rrbracket s * Q s \leq \llbracket G \rrbracket s * 1 + \llbracket \mathcal{N} G \rrbracket s * 1$ 
  by(blast intro:add-mono mult-left-mono)
thus  $\llbracket G \rrbracket s * \text{wlp body } R s + \llbracket \mathcal{N} G \rrbracket s * Q s \leq 1$ 
  by(simp add:negate-embed)
qed
qed
qed

```

We show healthiness by appealing to the properties of expectation fixed points, applied to the alternative loop definition.

lemma *healthy-wp-loop*:

```

fixes  $body::'s$  prog
assumes  $hb$ : healthy (wp body)
shows healthy (wp (do  $G \longrightarrow body$  od))

```

proof –

```

let  $?X P = (\lambda Q s. \llbracket G \rrbracket s * \text{wp body } Q s + \llbracket \mathcal{N} G \rrbracket s * P s)$ 

```

```

show ?thesis

```

proof(*intro healthy-parts bounded-byI2 nnegI2, simp-all add:wp-Loop1 hb soundI2 sound-intros*)

```

fix  $P::'s$  expect and  $c::real$  and  $s::'s$ 
assume  $sP$ : sound  $P$  and  $nnc$ :  $0 \leq c$ 
show  $c * (\text{lfp-exp } (?X P)) s = \text{lfp-exp } (?X (\lambda s. c * P s)) s$ 

```

proof(*cases*)

```

assume  $c = 0$  thus ?thesis

```

proof(*simp, intro antisym*)

```

from  $hb$  have  $fp$ :  $\lambda s. \llbracket G \rrbracket s * \text{wp body } (\lambda-. 0) s \Vdash \lambda s. 0$  by(simp)

```

```

hence  $\text{lfp-exp } (\lambda P s. \llbracket G \rrbracket s * \text{wp body } P s) \Vdash \lambda s. 0$ 

```

```

  by(auto intro:lfp-exp-lowerbound)

```

```

thus  $\text{lfp-exp } (\lambda P s. \llbracket G \rrbracket s * \text{wp body } P s) s \leq 0$  by(auto)

```

```

have  $\lambda s. 0 \Vdash \text{lfp-exp } (\lambda P s. \llbracket G \rrbracket s * \text{wp body } P s)$ 

```

```

  by(auto intro:lfp-exp-greatest fp)

```

```

thus  $0 \leq \text{lfp-exp } (\lambda P s. \llbracket G \rrbracket s * \text{wp body } P s) s$  by(auto)

```

qed

next

```

have onesided:  $\bigwedge P c. c \neq 0 \implies 0 \leq c \implies \text{sound } P \implies$ 

```

```

   $\lambda a. c * \text{lfp-exp } (\lambda a b. \llbracket G \rrbracket b * \text{wp body } a b + \llbracket \mathcal{N} G \rrbracket b * P b) a \Vdash$ 

```

```

   $\text{lfp-exp } (\lambda a b. \llbracket G \rrbracket b * \text{wp body } a b + \llbracket \mathcal{N} G \rrbracket b * (c * P b))$ 

```

proof –

```

fix  $P::'s$  expect and  $c::real$ 

```

```

assume  $cnz$ :  $c \neq 0$  and  $nnc$ :  $0 \leq c$  and  $sP$ : sound  $P$ 

```

```

with  $nnc$  have  $cpos$ :  $0 < c$  by(auto)

```

```

hence  $nnic$ :  $0 \leq \text{inverse } c$  by(auto)

```

```

show  $\lambda a. c * \text{lfp-exp } (\lambda a b. \llbracket G \rrbracket b * \text{wp body } a b + \llbracket \mathcal{N} G \rrbracket b * P b) a \Vdash$ 

```

```

   $\text{lfp-exp } (\lambda a b. \llbracket G \rrbracket b * \text{wp body } a b + \llbracket \mathcal{N} G \rrbracket b * (c * P b))$ 

```

proof(*rule lfp-exp-greatest*)
fix Q : 's *expect*
assume sQ : *sound Q*
and fp : $\lambda b. \llbracket G \rrbracket b * wp \text{ body } Q b + \llbracket \mathcal{N} G \rrbracket b * (c * P b) \Vdash Q$
hence $\bigwedge s. \llbracket G \rrbracket s * wp \text{ body } Q s + \llbracket \mathcal{N} G \rrbracket s * (c * P s) \leq Q s$ **by**(*auto*)
with *nnic*
have $\bigwedge s. \text{inverse } c * (\llbracket G \rrbracket s * wp \text{ body } Q s + \llbracket \mathcal{N} G \rrbracket s * (c * P s)) \leq$
 $\text{inverse } c * Q s$
by(*auto intro:mult-left-mono*)
hence $\bigwedge s. \llbracket G \rrbracket s * (\text{inverse } c * wp \text{ body } Q s) + (\text{inverse } c * c) * \llbracket \mathcal{N} G \rrbracket s * P s \leq$
 $\text{inverse } c * Q s$
by(*simp add:algebra-simps*)
hence $\bigwedge s. \llbracket G \rrbracket s * wp \text{ body } (\lambda s. \text{inverse } c * Q s) s + \llbracket \mathcal{N} G \rrbracket s * P s \leq$
 $\text{inverse } c * Q s$
by(*simp add:cnz scalingD[OF healthy-scalingD, OF hb sQ nnic]*)
hence $\lambda s. \llbracket G \rrbracket s * wp \text{ body } (\lambda s. \text{inverse } c * Q s) s + \llbracket \mathcal{N} G \rrbracket s * P s \Vdash$
 $\lambda s. \text{inverse } c * Q s$ **by**(*rule le-funI*)
moreover from *nnic sQ* **have** *sound* ($\lambda s. \text{inverse } c * Q s$)
by(*iprover intro:sound-intros*)
ultimately have *lfp-exp* ($\lambda a b. \llbracket G \rrbracket b * wp \text{ body } a b + \llbracket \mathcal{N} G \rrbracket b * P b$) \Vdash
 $\lambda s. \text{inverse } c * Q s$
by(*rule lfp-exp-lowerbound*)
hence $\bigwedge s. \text{lfp-exp } (\lambda a b. \llbracket G \rrbracket b * wp \text{ body } a b + \llbracket \mathcal{N} G \rrbracket b * P b) s \leq \text{inverse } c * Q s$
by(*rule le-funD*)
with *nnc*
have $\bigwedge s. c * \text{lfp-exp } (\lambda a b. \llbracket G \rrbracket b * wp \text{ body } a b + \llbracket \mathcal{N} G \rrbracket b * P b) s \leq$
 $c * (\text{inverse } c * Q s)$
by(*auto intro:mult-left-mono*)
also from *cnz* **have** $\bigwedge s. \dots s = Q s$ **by**(*simp*)
finally show $\lambda a. c * \text{lfp-exp } (\lambda a b. \llbracket G \rrbracket b * wp \text{ body } a b + \llbracket \mathcal{N} G \rrbracket b * P b) a \Vdash Q$
by(*rule le-funI*)
next
from *sP* **have** *sound* ($\lambda s. \text{bound-of } P$) **by**(*auto*)
with *hb sP* **have** *sound* (*lfp-exp* (?X P))
by(*blast intro:lfp-exp-sound lfp-loop-fp*)
with *nnc* **show** *sound* ($\lambda s. c * \text{lfp-exp } (?X P) s$)
by(*auto intro!:sound-intros*)

from *hb sP nnc*
show $\lambda s. \llbracket G \rrbracket s * wp \text{ body } (\lambda s. \text{bound-of } (\lambda s. c * P s)) s +$
 $\llbracket \mathcal{N} G \rrbracket s * (c * P s) \Vdash \lambda s. \text{bound-of } (\lambda s. c * P s)$
by(*iprover intro:lfp-loop-fp sound-intros*)

from *sP nnc* **show** *sound* ($\lambda s. \text{bound-of } (\lambda s. c * P s)$)
by(*auto intro!:sound-intros*)
qed
qed
assume *nzc*: $c \neq 0$

```

show ?thesis (is ?X P c s = ?Y P c s)
proof(rule fun-cong[where x=s], rule antisym)
from nzc nnc sP show ?X P c  $\Vdash$  ?Y P c by(rule onesided)

from nzc have nzc: inverse c  $\neq$  0 by(auto)
moreover with nnc have nnc: 0  $\leq$  inverse c by(auto)
moreover from nnc sP have scP: sound ( $\lambda s. c * P s$ ) by(auto intro!:sound-intros)
ultimately have ?X ( $\lambda s. c * P s$ ) (inverse c)  $\Vdash$  ?Y ( $\lambda s. c * P s$ ) (inverse c)
by(rule onesided)
with nnc have  $\lambda s. c * ?X (\lambda s. c * P s) (inverse c) s \Vdash$ 
 $\lambda s. c * ?Y (\lambda s. c * P s) (inverse c) s$ 
by(blast intro:mult-left-mono)
with nzc show ?Y P c  $\Vdash$  ?X P c by(simp add:mult.assoc[symmetric])
qed
qed
next
fix P::'s expect and b::real
assume bP: bounded-by b P and nP: nneg P
show lfp-exp ( $\lambda Q s. \langle\langle G \rangle\rangle s * wp \text{ body } Q s + \langle\langle \mathcal{N} G \rangle\rangle s * P s$ )  $\Vdash$   $\lambda s. b$ 
proof(intro lfp-exp-lowerbound le-funI)
fix s::'s
from bP nP hb have bounded-by b (wp body ( $\lambda s. b$ )) by(auto)
hence wp body ( $\lambda s. b$ ) s  $\leq$  b by(auto)
moreover from bP have P s  $\leq$  b by(auto)
ultimately have  $\langle\langle G \rangle\rangle s * wp \text{ body } (\lambda s. b) s + \langle\langle \mathcal{N} G \rangle\rangle s * P s \leq \langle\langle G \rangle\rangle s * b + \langle\langle \mathcal{N} G \rangle\rangle s$ 
* b
by(auto intro!:add-mono mult-left-mono)
also have ... = b by(simp add:negate-embed field-simps)
finally show  $\langle\langle G \rangle\rangle s * wp \text{ body } (\lambda s. b) s + \langle\langle \mathcal{N} G \rangle\rangle s * P s \leq b$  .
from bP nP have 0  $\leq$  b by(auto)
thus sound ( $\lambda s. b$ ) by(auto)
qed
from hb bP nP show  $\lambda s. 0 \Vdash$  lfp-exp ( $\lambda Q s. \langle\langle G \rangle\rangle s * wp \text{ body } Q s + \langle\langle \mathcal{N} G \rangle\rangle s * P s$ )
by(auto dest!:sound-nneg intro!:lfp-loop-greatest)
next
fix P Q::'s expect
assume sP: sound P and sQ: sound Q and le: P  $\Vdash$  Q
show lfp-exp (?X P)  $\Vdash$  lfp-exp (?X Q)
proof(rule lfp-exp-greatest)
fix R::'s expect
assume sR: sound R
and fp:  $\lambda s. \langle\langle G \rangle\rangle s * wp \text{ body } R s + \langle\langle \mathcal{N} G \rangle\rangle s * Q s \Vdash R$ 
from le have  $\lambda s. \langle\langle G \rangle\rangle s * wp \text{ body } R s + \langle\langle \mathcal{N} G \rangle\rangle s * P s \Vdash$ 
 $\lambda s. \langle\langle G \rangle\rangle s * wp \text{ body } R s + \langle\langle \mathcal{N} G \rangle\rangle s * Q s$ 
by(auto intro:le-funI add-left-mono mult-left-mono)
also note fp
finally show lfp-exp ( $\lambda R s. \langle\langle G \rangle\rangle s * wp \text{ body } R s + \langle\langle \mathcal{N} G \rangle\rangle s * P s$ )  $\Vdash$  R
using sR by(auto intro:lfp-exp-lowerbound)
next

```

```

from hb sP show sound (lfp-exp ( $\lambda R s. \langle\langle G \rangle\rangle s * wp \text{ body } R s + \langle\langle \mathcal{N} G \rangle\rangle s * P s$ ))
by(rule lfp-loop-sound)
from hb sQ show  $\lambda s. \langle\langle G \rangle\rangle s * wp \text{ body } (\lambda s. \text{bound-of } Q) s + \langle\langle \mathcal{N} G \rangle\rangle s * Q s \Vdash \lambda s. \text{bound-of } Q$ 
by(rule lfp-loop-fp)
from sQ show sound ( $\lambda s. \text{bound-of } Q$ ) by(auto)
qed
qed
qed

```

Use 'simp add:healthy_intros' or 'blast intro:healthy_intros' as appropriate to discharge healthiness side-conditions for primitive programs automatically.

```

lemmas healthy-intros =
  healthy-wp-Abort nearly-healthy-wlp-Abort healthy-wp-Skip nearly-healthy-wlp-Skip
  healthy-wp-Seq nearly-healthy-wlp-Seq healthy-wp-PC nearly-healthy-wlp-PC
  healthy-wp-DC nearly-healthy-wlp-DC healthy-wp-AC nearly-healthy-wlp-AC
  healthy-wp-Embed nearly-healthy-wlp-Embed healthy-wp-Apply nearly-healthy-wlp-Apply
  healthy-wp-SetDC nearly-healthy-wlp-SetDC healthy-wp-SetPC nearly-healthy-wlp-SetPC
  healthy-wp-Bind nearly-healthy-wlp-Bind healthy-wp-repeat nearly-healthy-wlp-repeat
  healthy-wp-loop nearly-healthy-wlp-loop
end

```

4.3 Continuity

theory *Continuity imports Healthiness begin*

We rely on one additional healthiness property, continuity, which is shown here separately, as its proof relies, in general, on healthiness. It is only relevant when a program appears in an inductive context i.e. inside a loop.

A continuous transformer preserves limits (or the suprema of ascending chains).

```

definition bd-cts :: 's trans  $\Rightarrow$  bool'
where bd-cts t = ( $\forall M. (\forall i. M i \Vdash M (\text{Suc } i)) \wedge \text{sound } (M i) \longrightarrow$ 
  ( $\exists b. \forall i. \text{bounded-by } b (M i) \longrightarrow$ 
   $t (\text{Sup-exp } (\text{range } M)) = \text{Sup-exp } (\text{range } (t o M))$ ))

```

lemma *bd-ctsD*:

```

[[ bd-cts t;  $\bigwedge i. M i \Vdash M (\text{Suc } i)$ ;  $\bigwedge i. \text{sound } (M i)$ ;  $\bigwedge i. \text{bounded-by } b (M i)$  ]]  $\Longrightarrow$ 
   $t (\text{Sup-exp } (\text{range } M)) = \text{Sup-exp } (\text{range } (t o M))$ 
unfolding bd-cts-def by(auto)

```

lemma *bd-ctsI*:

```

( $\bigwedge b M. (\bigwedge i. M i \Vdash M (\text{Suc } i)) \Longrightarrow (\bigwedge i. \text{sound } (M i)) \Longrightarrow (\bigwedge i. \text{bounded-by } b (M i)) \Longrightarrow$ 
   $t (\text{Sup-exp } (\text{range } M)) = \text{Sup-exp } (\text{range } (t o M))$ )  $\Longrightarrow$  bd-cts t
unfolding bd-cts-def by(auto)

```

A generalised property for transformers of transformers.

definition $bd\text{-cts}\text{-tr} :: ('s\ trans \Rightarrow 's\ trans) \Rightarrow bool$
where $bd\text{-cts}\text{-tr}\ T = (\forall M. (\forall i. le\text{-trans}\ (M\ i)\ (M\ (Suc\ i)) \wedge feasible\ (M\ i)) \longrightarrow$
 $equiv\text{-trans}\ (T\ (Sup\text{-trans}\ (M\ 'UNIV)))\ (Sup\text{-trans}\ ((T\ o\ M)\ 'UNIV)))$

lemma $bd\text{-cts}\text{-trD}$:
 $\llbracket bd\text{-cts}\text{-tr}\ T; \bigwedge i. le\text{-trans}\ (M\ i)\ (M\ (Suc\ i)); \bigwedge i. feasible\ (M\ i) \rrbracket \Longrightarrow$
 $equiv\text{-trans}\ (T\ (Sup\text{-trans}\ (M\ 'UNIV)))\ (Sup\text{-trans}\ ((T\ o\ M)\ 'UNIV))$
by($simp\ add:bd\text{-cts}\text{-tr}\text{-def}$)

lemma $bd\text{-cts}\text{-trI}$:
 $(\bigwedge M. (\bigwedge i. le\text{-trans}\ (M\ i)\ (M\ (Suc\ i))) \Longrightarrow (\bigwedge i. feasible\ (M\ i)) \Longrightarrow$
 $equiv\text{-trans}\ (T\ (Sup\text{-trans}\ (M\ 'UNIV)))\ (Sup\text{-trans}\ ((T\ o\ M)\ 'UNIV))) \Longrightarrow bd\text{-cts}\text{-tr}$
 T
by($simp\ add:bd\text{-cts}\text{-tr}\text{-def}$)

4.3.1 Continuity of Primitives

lemma $cts\text{-wp}\text{-Abort}$:
 $bd\text{-cts}\ (wp\ (Abort::'s\ prog))$
proof –
have $X: range\ (\lambda(i::nat)\ (s::'s). 0) = \{\lambda s. 0\}$ **by**($auto$)
show $?thesis$ **by**($intro\ bd\text{-cts}I, simp\ add:wp\text{-eval}\ o\text{-def}\ Sup\text{-exp}\text{-def}\ X$)
qed

lemma $cts\text{-wp}\text{-Skip}$:
 $bd\text{-cts}\ (wp\ Skip)$
by($rule\ bd\text{-cts}I, simp\ add:wp\text{-def}\ Skip\text{-def}\ o\text{-def}$)

lemma $cts\text{-wp}\text{-Apply}$:
 $bd\text{-cts}\ (wp\ (Apply\ f))$
proof –
have $X: \bigwedge M\ s. \{P\ (f\ s) \mid P. P \in range\ M\} = \{P\ s \mid P. P \in range\ (\lambda i\ s. M\ i\ (f\ s))\}$ **by**($auto$)
show $?thesis$ **by**($intro\ bd\text{-cts}I\ ext, simp\ add:wp\text{-eval}\ o\text{-def}\ Sup\text{-exp}\text{-def}\ X$)
qed

lemma $cts\text{-wp}\text{-Bind}$:
fixes $a::'a \Rightarrow 's\ prog$
assumes $ca: \bigwedge s. bd\text{-cts}\ (wp\ (a\ (f\ s)))$
shows $bd\text{-cts}\ (wp\ (Bind\ f\ a))$
proof($rule\ bd\text{-cts}I$)
fix $M::nat \Rightarrow 's\ expect$ **and** $c::real$
assume $chain: \bigwedge i. M\ i \Vdash M\ (Suc\ i)$ **and** $sM: \bigwedge i. sound\ (M\ i)$
and $bM: \bigwedge i. bounded\text{-by}\ c\ (M\ i)$
with $bd\text{-cts}D[OF\ ca]$
have $\bigwedge s. wp\ (a\ (f\ s))\ (Sup\text{-exp}\ (range\ M)) =$
 $Sup\text{-exp}\ (range\ (wp\ (a\ (f\ s))\ o\ M))$
by($auto$)
moreover **have** $\bigwedge s. \{fa\ s \mid fa \in range\ (\lambda x. wp\ (a\ (f\ s))\ (M\ x))\} =$
 $\{fa\ s \mid fa \in range\ (\lambda x\ s. wp\ (a\ (f\ s))\ (M\ x)\ s)\}$

```

by(auto)
ultimately show  $wp\ (Bind\ f\ a)\ (Sup\text{-}exp\ (range\ M)) =$ 
 $Sup\text{-}exp\ (range\ (wp\ (Bind\ f\ a)\ \circ\ M))$ 
by(simp add:wp-eval o-def Sup-exp-def)
qed

```

The first nontrivial proof. We transform the suprema into limits, and appeal to the continuity of the underlying operation (here infimum). This is typical of the remainder of the nonrecursive elements.

lemma *cts-wp-DC*:

```

fixes  $a\ b::'s\ prog$ 
assumes  $ca: bd\text{-}cts\ (wp\ a)$ 
 $\quad$  and  $cb: bd\text{-}cts\ (wp\ b)$ 
 $\quad$  and  $ha: healthy\ (wp\ a)$ 
 $\quad$  and  $hb: healthy\ (wp\ b)$ 
shows  $bd\text{-}cts\ (wp\ (a\ \sqcap\ b))$ 
proof(rule bd-ctsI, rule antisym)
fix  $M::nat \Rightarrow 's\ expect$  and  $c::real$ 
assume  $chain: \bigwedge i. M\ i \Vdash M\ (Suc\ i)$  and  $sM: \bigwedge i. sound\ (M\ i)$ 
 $\quad$  and  $bM: \bigwedge i. bounded\text{-}by\ c\ (M\ i)$ 

from  $ha\ hb$  have  $hab: healthy\ (wp\ (a\ \sqcap\ b))$  by(rule healthy-intros)
from  $bM$  have  $leSup: \bigwedge i. M\ i \Vdash Sup\text{-}exp\ (range\ M)$  by(auto intro:Sup-exp-upper)
from  $sM\ bM$  have  $sSup: sound\ (Sup\text{-}exp\ (range\ M))$  by(auto intro:Sup-exp-sound)

show  $Sup\text{-}exp\ (range\ (wp\ (a\ \sqcap\ b)\ \circ\ M)) \Vdash wp\ (a\ \sqcap\ b)\ (Sup\text{-}exp\ (range\ M))$ 
proof(rule Sup-exp-least, clarsimp, rule le-funI)
fix  $i\ s$ 
from  $mono\text{-}transD[OF\ healthy\text{-}monoD[OF\ hab]]\ leSup\ sM\ sSup$ 
have  $wp\ (a\ \sqcap\ b)\ (M\ i) \Vdash wp\ (a\ \sqcap\ b)\ (Sup\text{-}exp\ (range\ M))$  by(auto)
thus  $wp\ (a\ \sqcap\ b)\ (M\ i)\ s \leq wp\ (a\ \sqcap\ b)\ (Sup\text{-}exp\ (range\ M))\ s$  by(auto)

from  $hab\ sSup$  have  $sound\ (wp\ (a\ \sqcap\ b)\ (Sup\text{-}exp\ (range\ M)))$  by(auto)
thus  $nneg\ (wp\ (a\ \sqcap\ b)\ (Sup\text{-}exp\ (range\ M)))$  by(auto)
qed

from  $sM\ bM\ ha$  have  $\bigwedge i. bounded\text{-}by\ c\ (wp\ a\ (M\ i))$  by(auto)
hence  $baM: \bigwedge i\ s. wp\ a\ (M\ i)\ s \leq c$  by(auto)
from  $sM\ bM\ hb$  have  $\bigwedge i. bounded\text{-}by\ c\ (wp\ b\ (M\ i))$  by(auto)
hence  $bbM: \bigwedge i\ s. wp\ b\ (M\ i)\ s \leq c$  by(auto)

show  $wp\ (a\ \sqcap\ b)\ (Sup\text{-}exp\ (range\ M)) \Vdash Sup\text{-}exp\ (range\ (wp\ (a\ \sqcap\ b)\ \circ\ M))$ 
proof(simp add:wp-eval o-def, rule le-funI)
fix  $s::'s$ 
from  $bd\text{-}ctsD[OF\ ca, of\ M, OF\ chain\ sM\ bM]\ bd\text{-}ctsD[OF\ cb, of\ M, OF\ chain\ sM\ bM]$ 
have  $min\ (wp\ a\ (Sup\text{-}exp\ (range\ M))\ s)\ (wp\ b\ (Sup\text{-}exp\ (range\ M))\ s) =$ 
 $min\ (Sup\text{-}exp\ (range\ (wp\ a\ \circ\ M))\ s)\ (Sup\text{-}exp\ (range\ (wp\ b\ \circ\ M))\ s)$  by(simp)
also {
have  $\{f\ s\ | f \in range\ (\lambda x. wp\ a\ (M\ x))\} = range\ (\lambda i. wp\ a\ (M\ i)\ s)$ 

```



```

    {f s | f.f ∈ range (λx. wp b (M x))} = range (λi. wp b (M i) s)
  by(auto)
  hence min (Sup-exp (range (wp a o M)) s) (Sup-exp (range (wp b o M)) s) =
    min (Sup (range (λi. wp a (M i) s))) (Sup (range (λi. wp b (M i) s)))
  by(simp add:Sup-exp-def o-def)
}
also {
  have (λi. wp a (M i) s) ⟶ Sup (range (λi. wp a (M i) s))
  proof(rule increasing-LIMSEQ)
    fix n
    from mono-transD[OF healthy-monoD, OF ha] sM chain
    show wp a (M n) s ≤ wp a (M (Suc n)) s by(auto intro:le-funD)
    from baM show wp a (M n) s ≤ Sup (range (λi. wp a (M i) s))
    by(intro cSup-upper bdd-aboveI, auto)

    fix e::real assume pe: 0 < e
    from baM have cSup: Sup (range (λi. wp a (M i) s)) ∈ closure (range (λi. wp a (M
i) s))
    by(blast intro:closure-contains-Sup)
    with pe obtain y where yin: y ∈ (range (λi. wp a (M i) s))
      and dy: dist y (Sup (range (λi. wp a (M i) s))) < e
    by(blast dest:iffDI[OF closure-approachable])
    from yin obtain i where y = wp a (M i) s by(auto)
    with dy have dist (wp a (M i) s) (Sup (range (λi. wp a (M i) s))) < e
    by(simp)
    moreover from baM have wp a (M i) s ≤ Sup (range (λi. wp a (M i) s))
    by(intro cSup-upper bdd-aboveI, auto)
    ultimately have Sup (range (λi. wp a (M i) s)) ≤ wp a (M i) s + e
    by(simp add:dist-real-def)
    thus ∃ i. Sup (range (λi. wp a (M i) s)) ≤ wp a (M i) s + e by(auto)
  qed
  moreover
  have (λi. wp b (M i) s) ⟶ Sup (range (λi. wp b (M i) s))
  proof(rule increasing-LIMSEQ)
    fix n
    from mono-transD[OF healthy-monoD, OF hb] sM chain
    show wp b (M n) s ≤ wp b (M (Suc n)) s by(auto intro:le-funD)
    from bbM show wp b (M n) s ≤ Sup (range (λi. wp b (M i) s))
    by(intro cSup-upper bdd-aboveI, auto)

    fix e::real assume pe: 0 < e
    from bbM have cSup: Sup (range (λi. wp b (M i) s)) ∈ closure (range (λi. wp b (M
i) s))
    by(blast intro:closure-contains-Sup)
    with pe obtain y where yin: y ∈ (range (λi. wp b (M i) s))
      and dy: dist y (Sup (range (λi. wp b (M i) s))) < e
    by(blast dest:iffDI[OF closure-approachable])
    from yin obtain i where y = wp b (M i) s by(auto)
    with dy have dist (wp b (M i) s) (Sup (range (λi. wp b (M i) s))) < e

```

by(*simp*)
moreover from *bbM* **have** $wp\ b\ (M\ i)\ s \leq Sup\ (range\ (\lambda i.\ wp\ b\ (M\ i)\ s))$
by(*intro cSup-upper bdd-aboveI, auto*)
ultimately have $Sup\ (range\ (\lambda i.\ wp\ b\ (M\ i)\ s)) \leq wp\ b\ (M\ i)\ s + e$
by(*simp add:dist-real-def*)
thus $\exists i.\ Sup\ (range\ (\lambda i.\ wp\ b\ (M\ i)\ s)) \leq wp\ b\ (M\ i)\ s + e$ **by**(*auto*)
qed
ultimately have $(\lambda i.\ min\ (wp\ a\ (M\ i)\ s)\ (wp\ b\ (M\ i)\ s)) \longrightarrow$
 $min\ (Sup\ (range\ (\lambda i.\ wp\ a\ (M\ i)\ s)))\ (Sup\ (range\ (\lambda i.\ wp\ b\ (M\ i)\ s)))$
by(*rule tendsto-min*)
moreover have *bdd-above* $(range\ (\lambda i.\ min\ (wp\ a\ (M\ i)\ s)\ (wp\ b\ (M\ i)\ s)))$
proof(*intro bdd-aboveI, clarsimp*)
fix *i*
have $min\ (wp\ a\ (M\ i)\ s)\ (wp\ b\ (M\ i)\ s) \leq wp\ a\ (M\ i)\ s$ **by**(*auto*)
also {
from *ha sM bM* **have** *bounded-by c* $(wp\ a\ (M\ i))$ **by**(*auto*)
hence $wp\ a\ (M\ i)\ s \leq c$ **by**(*auto*)
}
finally show $min\ (wp\ a\ (M\ i)\ s)\ (wp\ b\ (M\ i)\ s) \leq c$.
qed
ultimately
have $min\ (Sup\ (range\ (\lambda i.\ wp\ a\ (M\ i)\ s)))\ (Sup\ (range\ (\lambda i.\ wp\ b\ (M\ i)\ s))) \leq$
 $Sup\ (range\ (\lambda i.\ min\ (wp\ a\ (M\ i)\ s)\ (wp\ b\ (M\ i)\ s)))$
by(*blast intro:LIMSEQ-le-const2 cSup-upper min.mono[OF baM bbM]*)
}
also {
have $range\ (\lambda i.\ min\ (wp\ a\ (M\ i)\ s)\ (wp\ b\ (M\ i)\ s)) =$
 $\{f\ s\ |\ f.\ f \in range\ (\lambda i\ s.\ min\ (wp\ a\ (M\ i)\ s)\ (wp\ b\ (M\ i)\ s))\}$
by(*auto*)
hence $Sup\ (range\ (\lambda i.\ min\ (wp\ a\ (M\ i)\ s)\ (wp\ b\ (M\ i)\ s))) =$
 $Sup\text{-exp}\ (range\ (\lambda i\ s.\ min\ (wp\ a\ (M\ i)\ s)\ (wp\ b\ (M\ i)\ s)))\ s$
by (*simp add: Sup-exp-def cong del: SUP-cong-simp*)
}
finally show $min\ (wp\ a\ (Sup\text{-exp}\ (range\ M))\ s)\ (wp\ b\ (Sup\text{-exp}\ (range\ M))\ s) \leq$
 $Sup\text{-exp}\ (range\ (\lambda i\ s.\ min\ (wp\ a\ (M\ i)\ s)\ (wp\ b\ (M\ i)\ s)))\ s$.
qed
qed

lemma *cts-wp-Seq*:
fixes *a b::'s prog*
assumes *ca: bd-cts* $(wp\ a)$
and *cb: bd-cts* $(wp\ b)$
and *hb: healthy* $(wp\ b)$
shows *bd-cts* $(wp\ (a\ ;;\ b))$
proof(*rule bd-ctsI, simp add:o-def wp-eval*)
fix *M::nat* \Rightarrow 's *expect* **and** *c::real*
assume *chain*: $\bigwedge i.\ M\ i \Vdash M\ (Suc\ i)$ **and** *sM*: $\bigwedge i.\ sound\ (M\ i)$
and *bM*: $\bigwedge i.\ bounded\text{-by}\ c\ (M\ i)$
hence $wp\ a\ (wp\ b\ (Sup\text{-exp}\ (range\ M))) = wp\ a\ (Sup\text{-exp}\ (range\ (wp\ b\ o\ M)))$

```

by(subst bd-ctsD[OF cb], auto)
also {
  from sM hb have  $\bigwedge i. \text{sound } ((\text{wp } b \text{ o } M) i) \text{ by}(auto)$ 
  moreover from chain sM
  have  $\bigwedge i. (\text{wp } b \text{ o } M) i \Vdash (\text{wp } b \text{ o } M) (\text{Suc } i)$ 
    by(auto intro:mono-transD[OF healthy-monoD, OF hb])
  moreover from sM bM hb have  $\bigwedge i. \text{bounded-by } c ((\text{wp } b \text{ o } M) i) \text{ by}(auto)$ 
  ultimately have  $\text{wp } a (\text{Sup-exp } (\text{range } (\text{wp } b \text{ o } M))) =$ 
     $\text{Sup-exp } (\text{range } (\text{wp } a \text{ o } (\text{wp } b \text{ o } M)))$ 
    by(subst bd-ctsD[OF ca], auto)
}
also have  $\text{Sup-exp } (\text{range } (\text{wp } a \text{ o } (\text{wp } b \text{ o } M))) =$ 
   $\text{Sup-exp } (\text{range } (\lambda i. \text{wp } a (\text{wp } b (M i))))$ 
  by(simp add:o-def)
finally show  $\text{wp } a (\text{wp } b (\text{Sup-exp } (\text{range } M))) =$ 
   $\text{Sup-exp } (\text{range } (\lambda i. \text{wp } a (\text{wp } b (M i)))) .$ 
qed

lemma cts-wp-PC:
fixes a b::'s prog
assumes ca: bd-cts (wp a)
  and cb: bd-cts (wp b)
  and ha: healthy (wp a)
  and hb: healthy (wp b)
  and up: unitary p
shows bd-cts (wp (PC a p b))
proof(rule bd-ctsI, rule ext, simp add:o-def wp-eval)
fix M::nat  $\Rightarrow$  's expect and c::real and s::'s
assume chain:  $\bigwedge i. M i \Vdash M (\text{Suc } i)$  and sM:  $\bigwedge i. \text{sound } (M i)$ 
  and bM:  $\bigwedge i. \text{bounded-by } c (M i)$ 

from sM have  $\bigwedge i. \text{nneg } (M i) \text{ by}(auto)$ 
with bM have  $nc: 0 \leq c \text{ by}(auto)$ 

from chain sM bM have  $\text{wp } a (\text{Sup-exp } (\text{range } M)) = \text{Sup-exp } (\text{range } (\text{wp } a \text{ o } M))$ 
  by(rule bd-ctsD[OF ca])
hence  $\text{wp } a (\text{Sup-exp } (\text{range } M)) s = \text{Sup-exp } (\text{range } (\text{wp } a \text{ o } M)) s$ 
  by(simp)
also {
  have  $\{f s \mid f. f \in \text{range } (\lambda x. \text{wp } a (M x))\} = \text{range } (\lambda i. \text{wp } a (M i) s)$ 
    by(auto)
  hence  $\text{Sup-exp } (\text{range } (\text{wp } a \text{ o } M)) s = \text{Sup } (\text{range } (\lambda i. \text{wp } a (M i) s))$ 
    by(simp add:Sup-exp-def o-def)
}
finally have  $p s * \text{wp } a (\text{Sup-exp } (\text{range } M)) s =$ 
   $p s * \text{Sup } (\text{range } (\lambda i. \text{wp } a (M i) s)) \text{ by}(simp)$ 
also have  $\dots = \text{Sup } \{p s * x \mid x. x \in \text{range } (\lambda i. \text{wp } a (M i) s)\}$ 
proof(rule cSup-mult, blast, clarsimp)
from up show  $0 \leq p s \text{ by}(auto)$ 

```

```

fix  $i$ 
from  $sM$   $bM$   $ha$  have bounded-by  $c$  ( $wp$   $a$  ( $M$   $i$ )) by(auto)
thus  $wp$   $a$  ( $M$   $i$ )  $s \leq c$  by(auto)
qed
also {
  have  $\{p$   $s * x \mid x. x \in range$  ( $\lambda i. wp$   $a$  ( $M$   $i$ )  $s$ ) $\} = range$  ( $\lambda i. p$   $s * wp$   $a$  ( $M$   $i$ )  $s$ )
    by(auto)
  hence  $Sup$   $\{p$   $s * x \mid x. x \in range$  ( $\lambda i. wp$   $a$  ( $M$   $i$ )  $s$ ) $\} =$ 
     $Sup$  ( $range$  ( $\lambda i. p$   $s * wp$   $a$  ( $M$   $i$ )  $s$ )) by(simp)
}
finally have  $p$   $s * wp$   $a$  ( $Sup$ -exp ( $range$   $M$ ))  $s = Sup$  ( $range$  ( $\lambda i. p$   $s * wp$   $a$  ( $M$   $i$ )  $s$ )) .
moreover {
  from chain  $sM$   $bM$  have  $wp$   $b$  ( $Sup$ -exp ( $range$   $M$ )) =  $Sup$ -exp ( $range$  ( $wp$   $b$   $o$   $M$ ))
    by(rule bd-ctsD[OF cb])
  hence  $wp$   $b$  ( $Sup$ -exp ( $range$   $M$ ))  $s = Sup$ -exp ( $range$  ( $wp$   $b$   $o$   $M$ ))  $s$ 
    by(simp)
  also {
    have  $\{f$   $s \mid f. f \in range$  ( $\lambda x. wp$   $b$  ( $M$   $x$ )) $\} = range$  ( $\lambda i. wp$   $b$  ( $M$   $i$ )  $s$ )
      by(auto)
    hence  $Sup$ -exp ( $range$  ( $wp$   $b$   $o$   $M$ ))  $s = Sup$  ( $range$  ( $\lambda i. wp$   $b$  ( $M$   $i$ )  $s$ ))
      by(simp add:Sup-exp-def o-def)
  }
  finally have  $(1 - p$   $s) * wp$   $b$  ( $Sup$ -exp ( $range$   $M$ ))  $s =$ 
     $(1 - p$   $s) * Sup$  ( $range$  ( $\lambda i. wp$   $b$  ( $M$   $i$ )  $s$ )) by(simp)
  also have  $\dots = Sup$   $\{(1 - p$   $s) * x \mid x. x \in range$  ( $\lambda i. wp$   $b$  ( $M$   $i$ )  $s$ ) $\}$ 
  proof(rule cSup-mult, blast, clarsimp)
  from up show  $0 \leq 1 - p$   $s$ 
    by auto
  fix  $i$ 
  from  $sM$   $bM$   $hb$  have bounded-by  $c$  ( $wp$   $b$  ( $M$   $i$ )) by(auto)
  thus  $wp$   $b$  ( $M$   $i$ )  $s \leq c$  by(auto)
qed
also {
  have  $\{(1 - p$   $s) * x \mid x. x \in range$  ( $\lambda i. wp$   $b$  ( $M$   $i$ )  $s$ ) $\} =$ 
     $range$  ( $\lambda i. (1 - p$   $s) * wp$   $b$  ( $M$   $i$ )  $s$ )
    by(auto)
  hence  $Sup$   $\{(1 - p$   $s) * x \mid x. x \in range$  ( $\lambda i. wp$   $b$  ( $M$   $i$ )  $s$ ) $\} =$ 
     $Sup$  ( $range$  ( $\lambda i. (1 - p$   $s) * wp$   $b$  ( $M$   $i$ )  $s$ )) by(simp)
}
finally have  $(1 - p$   $s) * wp$   $b$  ( $Sup$ -exp ( $range$   $M$ ))  $s =$ 
   $Sup$  ( $range$  ( $\lambda i. (1 - p$   $s) * wp$   $b$  ( $M$   $i$ )  $s$ )) .
}
ultimately
have  $p$   $s * wp$   $a$  ( $Sup$ -exp ( $range$   $M$ ))  $s + (1 - p$   $s) * wp$   $b$  ( $Sup$ -exp ( $range$   $M$ ))  $s =$ 
   $Sup$  ( $range$  ( $\lambda i. p$   $s * wp$   $a$  ( $M$   $i$ )  $s$ )) +  $Sup$  ( $range$  ( $\lambda i. (1 - p$   $s) * wp$   $b$  ( $M$   $i$ )  $s$ ))
  by(simp)
also {
  from  $bM$   $sM$   $ha$  have  $\bigwedge i. bounded$ -by  $c$  ( $wp$   $a$  ( $M$   $i$ )) by(auto)
  hence  $\bigwedge i. wp$   $a$  ( $M$   $i$ )  $s \leq c$  by(auto)
}

```

moreover from up have $0 \leq p s$ by(auto)
ultimately have $\bigwedge i. p s * wp a (M i) s \leq p s * c$ by(auto intro:mult-left-mono)
also from $up nc$ have $p s * c \leq 1 * c$ by(blast intro:mult-right-mono)
also have $\dots = c$ by(simp)
finally have baM : $\bigwedge i. p s * wp a (M i) s \leq c$.

have $lima$: $(\lambda i. p s * wp a (M i) s) \longrightarrow Sup (range (\lambda i. p s * wp a (M i) s))$
proof(rule increasing-LIMSEQ)
fix n
from sM chain healthy-monoD[OF ha] have $wp a (M n) \Vdash wp a (M (Suc n))$
by(auto)
with up show $p s * wp a (M n) s \leq p s * wp a (M (Suc n)) s$
by(blast intro:mult-left-mono)
from baM show $p s * wp a (M n) s \leq Sup (range (\lambda i. p s * wp a (M i) s))$
by(intro cSup-upper bdd-aboveI, auto)

next
fix $e::real$
assume pe : $0 < e$
from baM have $Sup (range (\lambda i. p s * wp a (M i) s)) \in$
 $closure (range (\lambda i. p s * wp a (M i) s))$
by(blast intro:closure-contains-Sup)
thm closure-approachable
with pe obtain y where yin : $y \in range (\lambda i. p s * wp a (M i) s)$
and dy : $dist y (Sup (range (\lambda i. p s * wp a (M i) s))) < e$
by(blast dest:iffD1[OF closure-approachable])
from yin obtain i where $y = p s * wp a (M i) s$ by(auto)
with dy have $dist (p s * wp a (M i) s) (Sup (range (\lambda i. p s * wp a (M i) s))) < e$
by(simp)
moreover from baM have $p s * wp a (M i) s \leq Sup (range (\lambda i. p s * wp a (M i) s))$
by(intro cSup-upper bdd-aboveI, auto)
ultimately have $Sup (range (\lambda i. p s * wp a (M i) s)) \leq p s * wp a (M i) s + e$
by(simp add:dist-real-def)
thus $\exists i. Sup (range (\lambda i. p s * wp a (M i) s)) \leq p s * wp a (M i) s + e$ by(auto)
qed

from bM sM hb have $\bigwedge i. bounded-by c (wp b (M i))$ by(auto)
hence $\bigwedge i. wp b (M i) s \leq c$ by(auto)
moreover from up have $0 \leq (1 - p s)$
by auto
ultimately have $\bigwedge i. (1 - p s) * wp b (M i) s \leq (1 - p s) * c$ by(auto intro:mult-left-mono)
also {
from up have $1 - p s \leq 1$ by(auto)
with nc have $(1 - p s) * c \leq 1 * c$ by(blast intro:mult-right-mono)
}
also have $1 * c = c$ by(simp)
finally have bbM : $\bigwedge i. (1 - p s) * wp b (M i) s \leq c$.

have $limb$: $(\lambda i. (1 - p s) * wp b (M i) s) \longrightarrow Sup (range (\lambda i. (1 - p s) * wp b (M i) s))$

```

proof(rule increasing-LIMSEQ)
  fix n
  from sM chain healthy-monoD[OF hb] have wp b (M n)  $\Vdash$  wp b (M (Suc n))
    by(auto)
  moreover from up have  $0 \leq 1 - p s$ 
    by auto
  ultimately show  $(1 - p s) * wp b (M n) s \leq (1 - p s) * wp b (M (Suc n)) s$ 
    by(blast intro:mult-left-mono)
  from bbM show  $(1 - p s) * wp b (M n) s \leq \text{Sup} (\text{range} (\lambda i. (1 - p s) * wp b (M i) s))$ 
    by(intro cSup-upper bdd-aboveI, auto)
next
  fix e::real
  assume pe:  $0 < e$ 
  from bbM have  $\text{Sup} (\text{range} (\lambda i. (1 - p s) * wp b (M i) s)) \in$ 
     $\text{closure} (\text{range} (\lambda i. (1 - p s) * wp b (M i) s))$ 
    by(blast intro:closure-contains-Sup)
  with pe obtain y where yin:  $y \in \text{range} (\lambda i. (1 - p s) * wp b (M i) s)$ 
    and dy:  $\text{dist } y (\text{Sup} (\text{range} (\lambda i. (1 - p s) * wp b (M i) s))) < e$ 
    by(blast dest:iffD1[OF closure-approachable])
  from yin obtain i where  $y = (1 - p s) * wp b (M i) s$  by(auto)
  with dy have  $\text{dist} ((1 - p s) * wp b (M i) s)$ 
     $(\text{Sup} (\text{range} (\lambda i. (1 - p s) * wp b (M i) s))) < e$ 
    by(simp)
  moreover from bbM
  have  $(1 - p s) * wp b (M i) s \leq \text{Sup} (\text{range} (\lambda i. (1 - p s) * wp b (M i) s))$ 
    by(intro cSup-upper bdd-aboveI, auto)
  ultimately have  $\text{Sup} (\text{range} (\lambda i. (1 - p s) * wp b (M i) s)) \leq (1 - p s) * wp b (M i) s$ 
+ e
    by(simp add:dist-real-def)
  thus  $\exists i. \text{Sup} (\text{range} (\lambda i. (1 - p s) * wp b (M i) s)) \leq (1 - p s) * wp b (M i) s + e$ 
by(auto)
qed

from lima limb have  $(\lambda i. p s * wp a (M i) s + (1 - p s) * wp b (M i) s) \longrightarrow$ 
   $\text{Sup} (\text{range} (\lambda i. p s * wp a (M i) s)) + \text{Sup} (\text{range} (\lambda i. (1 - p s) * wp b (M i) s))$ 
  by(rule tendsto-add)
moreover from add-mono[OF baM bbM]
have  $\bigwedge i. p s * wp a (M i) s + (1 - p s) * wp b (M i) s \leq$ 
   $\text{Sup} (\text{range} (\lambda i. p s * wp a (M i) s + (1 - p s) * wp b (M i) s))$ 
  by(intro cSup-upper bdd-aboveI, auto)
ultimately have  $\text{Sup} (\text{range} (\lambda i. p s * wp a (M i) s)) +$ 
   $\text{Sup} (\text{range} (\lambda i. (1 - p s) * wp b (M i) s)) \leq$ 
   $\text{Sup} (\text{range} (\lambda i. p s * wp a (M i) s + (1 - p s) * wp b (M i) s))$ 
  by(blast intro: LIMSEQ-le-const2)
}
also {
have  $\text{range} (\lambda i. p s * wp a (M i) s + (1 - p s) * wp b (M i) s) =$ 
   $\{f s \mid f \in \text{range} (\lambda x s. p s * wp a (M x) s + (1 - p s) * wp b (M x) s)\}$ 
by(auto)

```

```

hence  $Sup (\text{range } (\lambda i. p s * wp a (M i) s + (1 - p s) * wp b (M i) s)) =$ 
 $Sup\text{-exp } (\text{range } (\lambda x s. p s * wp a (M x) s + (1 - p s) * wp b (M x) s)) s$ 
by (simp add: Sup-exp-def cong del: SUP-cong-simp)
}
finally
have  $p s * wp a (Sup\text{-exp } (\text{range } M)) s + (1 - p s) * wp b (Sup\text{-exp } (\text{range } M)) s \leq$ 
 $Sup\text{-exp } (\text{range } (\lambda i s. p s * wp a (M i) s + (1 - p s) * wp b (M i) s)) s .$ 
moreover
have  $Sup\text{-exp } (\text{range } (\lambda i s. p s * wp a (M i) s + (1 - p s) * wp b (M i) s)) s \leq$ 
 $p s * wp a (Sup\text{-exp } (\text{range } M)) s + (1 - p s) * wp b (Sup\text{-exp } (\text{range } M)) s$ 
proof (rule le-funD[OF Sup-exp-least], clarsimp, rule le-funI)
fix  $i::nat$  and  $s::s$ 
from  $bM$  have  $leSup: M i \Vdash Sup\text{-exp } (\text{range } M)$ 
by (blast intro: Sup-exp-upper)
moreover from  $sM$   $bM$  have  $sSup: \text{sound } (Sup\text{-exp } (\text{range } M))$ 
by (auto intro: Sup-exp-sound)
moreover note healthy-monoD[OF ha] sM
ultimately have  $wp a (M i) \Vdash wp a (Sup\text{-exp } (\text{range } M))$  by (auto)
hence  $wp a (M i) s \leq wp a (Sup\text{-exp } (\text{range } M)) s$  by (auto)
moreover {
from  $leSup$   $sSup$  healthy-monoD[OF hb] sM
have  $wp b (M i) \Vdash wp b (Sup\text{-exp } (\text{range } M))$  by (auto)
hence  $wp b (M i) s \leq wp b (Sup\text{-exp } (\text{range } M)) s$  by (auto)
}
moreover from  $up$  have  $0 \leq p s$   $0 \leq 1 - p s$ 
by auto
ultimately
show  $p s * wp a (M i) s + (1 - p s) * wp b (M i) s \leq$ 
 $p s * wp a (Sup\text{-exp } (\text{range } M)) s + (1 - p s) * wp b (Sup\text{-exp } (\text{range } M)) s$ 
by (blast intro: add-mono mult-left-mono)

from  $sSup$   $ha$   $hb$  have  $\text{sound } (wp a (Sup\text{-exp } (\text{range } M)))$ 
 $\text{sound } (wp b (Sup\text{-exp } (\text{range } M)))$ 
by (auto)
hence  $\bigwedge s. 0 \leq wp a (Sup\text{-exp } (\text{range } M)) s \wedge s. 0 \leq wp b (Sup\text{-exp } (\text{range } M)) s$ 
by (auto)
moreover from  $up$  have  $\bigwedge s. 0 \leq p s \wedge s. 0 \leq 1 - p s$ 
by auto
ultimately show  $nneg (\lambda c. p c * wp a (Sup\text{-exp } (\text{range } M)) c +$ 
 $(1 - p c) * wp b (Sup\text{-exp } (\text{range } M)) c)$ 
by (blast intro: add-nonneg-nonneg mult-nonneg-nonneg)
qed
ultimately show  $p s * wp a (Sup\text{-exp } (\text{range } M)) s + (1 - p s) * wp b (Sup\text{-exp } (\text{range } M)) s =$ 
 $Sup\text{-exp } (\text{range } (\lambda x s. p s * wp a (M x) s + (1 - p s) * wp b (M x) s)) s$ 
by (auto)
qed

```

Both set-based choice operators are only continuous for finite sets (probabilistic

choice *can* be extended infinitely, but we have not done so). The proofs for both are inductive, and rely on the above results on binary operators.

lemma *SetPC-Bind*:

$SetPC\ a\ p = Bind\ p\ (\lambda p. SetPC\ a\ (\lambda-. p))$

by(*intro ext, simp add:SetPC-def Bind-def Let-def*)

lemma *SetPC-remove*:

assumes $nz: p\ x \neq 0$ **and** $n1: p\ x \neq 1$

and $fsupp: finite\ (supp\ p)$

shows $SetPC\ a\ (\lambda-. p) = PC\ (a\ x)\ (\lambda-. p\ x)\ (SetPC\ a\ (\lambda-. dist-remove\ p\ x))$

proof(*intro ext, simp add:SetPC-def PC-def*)

fix $ab\ P\ s$

from nz **have** $x \in supp\ p$ **by**(*simp add:supp-def*)

hence $supp\ p = insert\ x\ (supp\ p - \{x\})$ **by**(*auto*)

hence $(\sum_{x \in supp\ p}. p\ x * a\ x\ ab\ P\ s) =$

$(\sum_{x \in insert\ x\ (supp\ p - \{x\})}. p\ x * a\ x\ ab\ P\ s)$

by(*simp*)

also from $fsupp$

have $\dots = p\ x * a\ x\ ab\ P\ s + (\sum_{x \in supp\ p - \{x\}}. p\ x * a\ x\ ab\ P\ s)$

by(*blast intro:sum.insert*)

also from $n1$

have $\dots = p\ x * a\ x\ ab\ P\ s + (1 - p\ x) * ((\sum_{x \in supp\ p - \{x\}}. p\ x * a\ x\ ab\ P\ s) / (1 - p\ x))$

by(*simp add:field-simps*)

also have $\dots = p\ x * a\ x\ ab\ P\ s +$

$(1 - p\ x) * ((\sum_{y \in supp\ p - \{x\}}. (p\ y / (1 - p\ x)) * a\ y\ ab\ P\ s))$

by(*simp add:sum-divide-distrib*)

also have $\dots = p\ x * a\ x\ ab\ P\ s +$

$(1 - p\ x) * ((\sum_{y \in supp\ p - \{x\}}. dist-remove\ p\ x\ y * a\ y\ ab\ P\ s))$

by(*simp add:dist-remove-def*)

also from $nz\ n1$

have $\dots = p\ x * a\ x\ ab\ P\ s +$

$(1 - p\ x) * ((\sum_{y \in supp\ (dist-remove\ p\ x)}. dist-remove\ p\ x\ y * a\ y\ ab\ P\ s))$

by(*simp add:supp-dist-remove*)

finally show $(\sum_{x \in supp\ p}. p\ x * a\ x\ ab\ P\ s) =$

$p\ x * a\ x\ ab\ P\ s +$

$(1 - p\ x) * (\sum_{y \in supp\ (dist-remove\ p\ x)}. dist-remove\ p\ x\ y * a\ y\ ab\ P\ s).$

qed

lemma *cts-bot*:

$bd-cts\ (\lambda(P::'s\ expect)\ (s::'s). 0::real)$

proof –

have $X: \bigwedge s::'s. \{(P::'s\ expect)\ s \mid P. P \in range\ (\lambda P\ s. 0)\} = \{0\}$ **by**(*auto*)

show *?thesis* **by**(*intro bd-ctsI, simp add:Sup-exp-def o-def X*)

qed

lemma *wp-SetPC-nil*:

$wp\ (SetPC\ a\ (\lambda s\ a. 0)) = (\lambda P\ s. 0)$

by(*intro ext, simp add:wp-eval*)

lemma *SetPC-sgl*:

$\text{supp } p = \{x\} \implies \text{SetPC } a (\lambda \cdot. p) = (\lambda ab P s. p x * a x ab P s)$
by(*simp add:SetPC-def*)

lemma *bd-cts-scale*:

fixes $a::'s \text{ trans}$

assumes $ca: \text{bd-cts } a$

and $ha: \text{healthy } a$

and $nnc: 0 \leq c$

shows $\text{bd-cts } (\lambda P s. c * a P s)$

proof(*intro bd-ctsI ext, simp add:o-def*)

fix $M::\text{nat} \implies 's \text{ expect}$ **and** $d::\text{real}$ **and** $s::'s$

assume $\text{chain}: \bigwedge i. M i \Vdash M (\text{Suc } i)$ **and** $sM: \bigwedge i. \text{sound } (M i)$

and $bM: \bigwedge i. \text{bounded-by } d (M i)$

from sM **have** $\bigwedge i. \text{nneg } (M i)$ **by**(*auto*)

with bM **have** $\text{nnd}: 0 \leq d$ **by**(*auto*)

from $sM bM$ **have** $s\text{Sup}: \text{sound } (\text{Sup-exp } (\text{range } M))$ **by**(*auto intro:Sup-exp-sound*)

with $\text{healthy-scalingD}[OF ha]$ nnc

have $c * a (\text{Sup-exp } (\text{range } M)) s = a (\lambda s. c * \text{Sup-exp } (\text{range } M) s) s$

by(*auto intro:scalingD*)

also {

have $\bigwedge s. \{f s \mid f. f \in \text{range } M\} = \text{range } (\lambda i. M i s)$ **by**(*auto*)

hence $a (\lambda s. c * \text{Sup-exp } (\text{range } M) s) s =$

$a (\lambda s. c * \text{Sup } (\text{range } (\lambda i. M i s))) s$

by(*simp add:Sup-exp-def*)

}

also {

from bM **have** $\bigwedge x s. x \in \text{range } (\lambda i. M i s) \implies x \leq d$ **by**(*auto*)

with nnc **have** $a (\lambda s. c * \text{Sup } (\text{range } (\lambda i. M i s))) s =$

$a (\lambda s. \text{Sup } \{c * x \mid x. x \in \text{range } (\lambda i. M i s)\}) s$

by(*subst cSup-mult, blast+*)

}

also {

have $X: \bigwedge s. \{c * x \mid x. x \in \text{range } (\lambda i. M i s)\} = \text{range } (\lambda i. c * M i s)$ **by**(*auto*)

have $a (\lambda s. \text{Sup } \{c * x \mid x. x \in \text{range } (\lambda i. M i s)\}) s =$

$a (\lambda s. \text{Sup } (\text{range } (\lambda i. c * M i s))) s$ **by**(*simp add:X*)

}

also {

have $\bigwedge s. \text{range } (\lambda i. c * M i s) = \{f s \mid f. f \in \text{range } (\lambda i s. c * M i s)\}$

by(*auto*)

hence $(\lambda s. \text{Sup } (\text{range } (\lambda i. c * M i s))) = \text{Sup-exp } (\text{range } (\lambda i s. c * M i s))$

by (*simp add: Sup-exp-def cong del: SUP-cong-simp*)

hence $a (\lambda s. \text{Sup } (\text{range } (\lambda i. c * M i s))) s =$

$a (\text{Sup-exp } (\text{range } (\lambda i s. c * M i s))) s$ **by**(*simp*)

}

also {

```

from le-funD[OF chain] nnc
have  $\bigwedge i. (\lambda s. c * M i s) \Vdash (\lambda s. c * M (Suc i) s)$ 
  by(auto intro:le-funI[OF mult-left-mono])
moreover from sM nnc
have  $\bigwedge i. sound (\lambda s. c * M i s)$ 
  by(auto intro:sound-intros)
moreover from bM nnc
have  $\bigwedge i. bounded-by (c * d) (\lambda s. c * M i s)$ 
  by(auto intro:mult-left-mono)
ultimately
have  $a (Sup-exp (range (\lambda i s. c * M i s))) =$ 
   $Sup-exp (range (a o (\lambda i s. c * M i s)))$ 
  by(rule bd-ctsD[OF ca])
hence  $a (Sup-exp (range (\lambda i s. c * M i s))) s =$ 
   $Sup-exp (range (a o (\lambda i s. c * M i s))) s$ 
  by(auto)
}
also have  $Sup-exp (range (a o (\lambda i s. c * M i s))) s =$ 
   $Sup-exp (range (\lambda x. a (\lambda s. c * M x s))) s$ 
  by(simp add:o-def)
also {
  from nnc sM
  have  $\bigwedge x. a (\lambda s. c * M x s) = (\lambda s. c * a (M x) s)$ 
    by(auto intro:scalingD[OF healthy-scalingD, OF ha, symmetric])
  hence  $Sup-exp (range (\lambda x. a (\lambda s. c * M x s))) s =$ 
     $Sup-exp (range (\lambda x s. c * a (M x) s)) s$ 
    by(simp)
}
finally show  $c * a (Sup-exp (range M)) s = Sup-exp (range (\lambda x s. c * a (M x) s)) s .$ 
qed

```

lemma *cts-wp-SetPC-const*:

```

fixes a::'a  $\Rightarrow$  's prog
assumes ca:  $\bigwedge x. x \in (supp p) \Longrightarrow bd-cts (wp (a x))$ 
  and ha:  $\bigwedge x. x \in (supp p) \Longrightarrow healthy (wp (a x))$ 
  and up: unitary p
  and sump:  $sum p (supp p) \leq 1$ 
  and fsupp: finite (supp p)
shows bd-cts (wp (SetPC a (\lambda-. p)))
proof(cases supp p = {}, simp add:supp-empty SetPC-def wp-def cts-bot)
assume nesupp:  $supp p \neq \{\}$ 
from fsupp have unitary p  $\longrightarrow sum p (supp p) \leq 1 \longrightarrow$ 
   $(\forall x \in supp p. bd-cts (wp (a x))) \longrightarrow$ 
   $(\forall x \in supp p. healthy (wp (a x))) \longrightarrow$ 
  bd-cts (wp (SetPC a (\lambda-. p)))
proof(induct supp p arbitrary:p, simp add:supp-empty wp-SetPC-nil cts-bot, clarify)
fix x::'a and F::'a set and p::'a  $\Rightarrow$  real
assume fF: finite F
assume insert x F = supp p

```

hence $pstep$: $supp\ p = insert\ x\ F$ **by**($simp$)
hence xin : $x \in supp\ p$ **by**($auto$)
assume up : $unitary\ p$ **and** ca : $\forall x \in supp\ p. bd\text{-}cts\ (wp\ (a\ x))$
and ha : $\forall x \in supp\ p. healthy\ (wp\ (a\ x))$
and $sump$: $sum\ p\ (supp\ p) \leq 1$
and xni : $x \notin F$
assume IH : $\bigwedge p. F = supp\ p \implies$
 $unitary\ p \implies sum\ p\ (supp\ p) \leq 1 \implies$
 $(\forall x \in supp\ p. bd\text{-}cts\ (wp\ (a\ x))) \implies$
 $(\forall x \in supp\ p. healthy\ (wp\ (a\ x))) \implies$
 $bd\text{-}cts\ (wp\ (SetPC\ a\ (\lambda\cdot. p)))$

from $fF\ pstep$ **have** $fsupp$: $finite\ (supp\ p)$ **by**($auto$)

from xin **have** nzp : $p\ x \neq 0$ **by**($simp\ add$: $supp\text{-}def$)

have $xy\text{-}le\text{-}sum$:
 $\bigwedge y. y \in supp\ p \implies y \neq x \implies p\ x + p\ y \leq sum\ p\ (supp\ p)$
proof –
fix y **assume** yin : $y \in supp\ p$ **and** yne : $y \neq x$
from up **have** $0 \leq sum\ p\ (supp\ p - \{x, y\})$
by($auto\ intro$: $sum\text{-}nonneg$)
hence $p\ x + p\ y \leq p\ x + p\ y + sum\ p\ (supp\ p - \{x, y\})$
by($auto$)
also {
from $yin\ yne\ fsupp$
have $p\ y + sum\ p\ (supp\ p - \{x, y\}) = sum\ p\ (supp\ p - \{x\})$
by($subst\ sum.insert[symmetric]$, ($blast\ intro!$: $sum.cong$)+)
moreover
from $xin\ fsupp$
have $p\ x + sum\ p\ (supp\ p - \{x\}) = sum\ p\ (supp\ p)$
by($subst\ sum.insert[symmetric]$, ($blast\ intro!$: $sum.cong$)+)
ultimately
have $p\ x + p\ y + sum\ p\ (supp\ p - \{x, y\}) = sum\ p\ (supp\ p)$ **by**($simp$)
}

finally show $p\ x + p\ y \leq sum\ p\ (supp\ p)$.
qed

have $n1p$: $\bigwedge y. y \in supp\ p \implies y \neq x \implies p\ x \neq 1$
proof($rule\ ccontr, simp$)
assume $px1$: $p\ x = 1$
fix y **assume** yin : $y \in supp\ p$ **and** yne : $y \neq x$
from up **have** $0 \leq p\ y$ **by**($auto$)
with yin **have** $0 < p\ y$ **by**($auto\ simp$: $supp\text{-}def$)
hence $0 + p\ x < p\ y + p\ x$ **by**($rule\ add\text{-}strict\text{-}right\text{-}mono$)
with $px1$ **have** $1 < p\ x + p\ y$ **by**($simp$)
also from $yin\ yne$ **have** $p\ x + p\ y \leq sum\ p\ (supp\ p)$
by($rule\ xy\text{-}le\text{-}sum$)
finally show $False$ **using** $sump$ **by**($simp$)

qed

```

show bd-cts (wp (SetPC a ( $\lambda\cdot.p$ )))
proof(cases  $F = \{\}$ )
  case True with pstep have  $\text{supp } p = \{x\}$  by(simp)
  hence  $\text{wp } (\text{SetPC } a (\lambda\cdot.p)) = (\lambda P s. p x * \text{wp } (a x) P s)$ 
    by(simp add:SetPC-sgl wp-def)
  moreover {
    from up ca ha xin have  $\text{bd-cts } (\text{wp } (a x)) \text{ healthy } (\text{wp } (a x)) 0 \leq p x$ 
      by(auto)
    hence  $\text{bd-cts } (\lambda P s. p x * \text{wp } (a x) P s)$ 
      by(rule bd-cts-scale)
  }
  ultimately show ?thesis by(simp)
next
  assume  $\text{neF}: F \neq \{\}$ 
  then obtain y where  $\text{yinF}: y \in F$  by(auto)
  with xni have  $\text{yne}: y \neq x$  by(auto)
  from yinF pstep have  $\text{yin}: y \in \text{supp } p$  by(auto)

  from supp-dist-remove[of p x, OF nzp nlp, OF yin yne]
  have  $\text{supp-sub}: \text{supp } (\text{dist-remove } p x) \subseteq \text{supp } p$  by(auto)

  from xin ca have  $\text{cax}: \text{bd-cts } (\text{wp } (a x))$  by(auto)
  from xin ha have  $\text{hax}: \text{healthy } (\text{wp } (a x))$  by(auto)

  from supp-sub ha have  $\text{hra}: \forall x \in \text{supp } (\text{dist-remove } p x). \text{healthy } (\text{wp } (a x))$ 
    by(auto)
  from supp-sub ca have  $\text{cra}: \forall x \in \text{supp } (\text{dist-remove } p x). \text{bd-cts } (\text{wp } (a x))$ 
    by(auto)

  from supp-dist-remove[of p x, OF nzp nlp, OF yin yne] pstep xni
  have  $\text{Fsupp}: F = \text{supp } (\text{dist-remove } p x)$ 
    by(simp)

  have  $\text{udp}: \text{unitary } (\text{dist-remove } p x)$ 
  proof(intro unitaryI2 nnegI bounded-byI)
    fix y
    show  $0 \leq \text{dist-remove } p x y$ 
    proof(cases y=x, simp-all add:dist-remove-def)
      from up have  $0 \leq p y \ 0 \leq 1 - p x$ 
        by auto
      thus  $0 \leq p y / (1 - p x)$ 
        by(rule divide-nonneg-nonneg)
    qed
  show  $\text{dist-remove } p x y \leq 1$ 
  proof(cases y=x, simp-all add:dist-remove-def,
    cases y ∈ supp p, simp-all add:nsupp-zero)
    assume  $\text{yne}: y \neq x$  and  $\text{yin}: y \in \text{supp } p$ 

```

```

hence  $p x + p y \leq \text{sum } p \text{ (supp } p)$ 
  by(auto intro:xy-le-sum)
also note sump
finally have  $p y \leq 1 - p x$  by(auto)
moreover from up have  $p x \leq 1$  by(auto)
moreover from yin yne have  $p x \neq 1$  by(rule n1p)
ultimately show  $p y / (1 - p x) \leq 1$  by(auto)
qed
qed

from xin have  $pxn0: p x \neq 0$  by(auto simp:supp-def)
from yin yne have  $pxn1: p x \neq 1$  by(rule n1p)

from  $pxn0$   $pxn1$  have  $\text{sum (dist-remove } p x) \text{ (supp (dist-remove } p x)) =$ 
   $\text{sum (dist-remove } p x) \text{ (supp } p - \{x\})$ 
  by(simp add:supp-dist-remove)
also have  $\dots = (\sum_{y \in \text{supp } p - \{x\}} p y / (1 - p x))$ 
  by(simp add:dist-remove-def)
also have  $\dots = (\sum_{y \in \text{supp } p - \{x\}} p y) / (1 - p x)$ 
  by(simp add:sum-divide-distrib)
also {
  from xin have  $\text{insert } x \text{ (supp } p) = \text{supp } p$  by(auto)
  with fsupp have  $p x + (\sum_{y \in \text{supp } p - \{x\}} p y) = \text{sum } p \text{ (supp } p)$ 
    by(simp add:sum.insert[symmetric])
  also note sump
  finally have  $\text{sum } p \text{ (supp } p - \{x\}) \leq 1 - p x$  by(auto)
  moreover {
    from up have  $p x \leq 1$  by(auto)
    with  $pxn1$  have  $p x < 1$  by(auto)
    hence  $0 < 1 - p x$  by(auto)
  }
  ultimately have  $\text{sum } p \text{ (supp } p - \{x\}) / (1 - p x) \leq 1$ 
    by(auto)
  }
finally have  $\text{sdp: sum (dist-remove } p x) \text{ (supp (dist-remove } p x)) \leq 1$  .

from Fsupp udp sdp hra cra IH
have cts-dr: bd-cts (wp (SetPC a (λ-. dist-remove p x)))
  by(auto)

from up have  $\text{upx: unitary (λ-. } p x)$  by(auto)

from  $pxn0$   $pxn1$  fsupp hra show ?thesis
  by(simp add:SetPC-remove,
    blast intro:cts-wp-PC cax cts-dr hax healthy-intros
    unitary-sound[OF udp] sdp upx)
qed
qed
with assms show ?thesis by(auto)

```

qed

lemma *cts-wp-SetPC*:

fixes $a::'a \Rightarrow 's$ prog

assumes $ca: \bigwedge x s. x \in (\text{supp } (p s)) \Longrightarrow \text{bd-cts } (\text{wp } (a x))$

and $ha: \bigwedge x s. x \in (\text{supp } (p s)) \Longrightarrow \text{healthy } (\text{wp } (a x))$

and $up: \bigwedge s. \text{unitary } (p s)$

and $\text{sump}: \bigwedge s. \text{sum } (p s) (\text{supp } (p s)) \leq 1$

and $\text{fsupp}: \bigwedge s. \text{finite } (\text{supp } (p s))$

shows $\text{bd-cts } (\text{wp } (\text{SetPC } a p))$

proof –

from *assms* **have** $\text{bd-cts } (\text{wp } (\text{Bind } p (\lambda p. \text{SetPC } a (\lambda-. p))))$

by (*iprover intro!*:*cts-wp-Bind cts-wp-SetPC-const*)

thus ?*thesis* **by** (*simp add:SetPC-Bind[symmetric]*)

qed

lemma *wp-SetDC-Bind*:

$\text{SetDC } a S = \text{Bind } S (\lambda S. \text{SetDC } a (\lambda-. S))$

by (*intro ext, simp add:SetDC-def Bind-def*)

lemma *SetDC-finite-insert*:

assumes $fS: \text{finite } S$

and $neS: S \neq \{\}$

shows $\text{SetDC } a (\lambda-. \text{insert } x S) = a x \sqcap \text{SetDC } a (\lambda-. S)$

proof (*intro ext, simp add: SetDC-def DC-def cong del: image-cong-simp cong add: INF-cong-simp*)

fix $ab P s$

from fS **have** $A: \text{finite } (\text{insert } (a x ab P s) ((\lambda x. a x ab P s) ' S))$

and $B: \text{finite } (((\lambda x. a x ab P s) ' S))$ **by** (*auto*)

from neS **have** $C: \text{insert } (a x ab P s) ((\lambda x. a x ab P s) ' S) \neq \{\}$

and $D: (\lambda x. a x ab P s) ' S \neq \{\}$ **by** (*auto*)

from $A C$ **have** $\text{Inf } (\text{insert } (a x ab P s) ((\lambda x. a x ab P s) ' S)) =$

$\text{Min } (\text{insert } (a x ab P s) ((\lambda x. a x ab P s) ' S))$

by (*auto intro:cInf-eq-Min*)

also from $B D$ **have** $\dots = \text{min } (a x ab P s) (\text{Min } ((\lambda x. a x ab P s) ' S))$

by (*auto intro:Min-insert*)

also from $B D$ **have** $\dots = \text{min } (a x ab P s) (\text{Inf } ((\lambda x. a x ab P s) ' S))$

by (*simp add:cInf-eq-Min*)

finally show $(\text{INF } x \in \text{insert } x S. a x ab P s) =$

$\text{min } (a x ab P s) (\text{INF } x \in S. a x ab P s)$

by (*simp cong del: INF-cong-simp*)

qed

lemma *SetDC-singleton*:

$\text{SetDC } a (\lambda-. \{x\}) = a x$

by (*simp add: SetDC-def cong del: INF-cong-simp*)

lemma *cts-wp-SetDC-const*:

fixes $a::'a \Rightarrow 's$ prog

assumes $ca: \bigwedge x. x \in S \Longrightarrow \text{bd-cts } (\text{wp } (a x))$

```

and ha:  $\bigwedge x. x \in S \implies \text{healthy } (\text{wp } (a \ x))$ 
and fS: finite S
and neS:  $S \neq \{\}$ 
shows bd-cts (wp (SetDC a ( $\lambda\cdot$ . S)))
proof –
have finite S  $\implies S \neq \{\} \implies$ 
  ( $\forall x \in S. \text{bd-cts } (\text{wp } (a \ x))$ )  $\longrightarrow$ 
  ( $\forall x \in S. \text{healthy } (\text{wp } (a \ x))$ )  $\longrightarrow$ 
  bd-cts (wp (SetDC a ( $\lambda\cdot$ . S)))
proof(induct S rule:finite-induct, simp, clarsimp)
fix x::'a and F::'a set
assume fF: finite F
and IH:  $F \neq \{\} \implies \text{bd-cts } (\text{wp } (\text{SetDC } a \ (\lambda\cdot$ . F)))
and cax: bd-cts (wp (a x))
and hax: healthy (wp (a x))
and haF:  $\forall x \in F. \text{healthy } (\text{wp } (a \ x))$ 
show bd-cts (wp (SetDC a ( $\lambda\cdot$ . insert x F)))
proof(cases F = \{\}, simp add:SetDC-singleton cax)
assume  $F \neq \{\}$ 
with fF cax hax haF IH show bd-cts (wp (SetDC a ( $\lambda\cdot$ . insert x F)))
by(auto intro!:cts-wp-DC healthy-intros simp:SetDC-finite-insert)
qed
qed
with assms show ?thesis by(auto)
qed

```

```

lemma cts-wp-SetDC:
fixes a::'a  $\Rightarrow$  's prog
assumes ca:  $\bigwedge x s. x \in S \ s \implies \text{bd-cts } (\text{wp } (a \ x))$ 
and ha:  $\bigwedge x s. x \in S \ s \implies \text{healthy } (\text{wp } (a \ x))$ 
and fS:  $\bigwedge s. \text{finite } (S \ s)$ 
and neS:  $\bigwedge s. S \ s \neq \{\}$ 
shows bd-cts (wp (SetDC a S))
proof –
from assms have bd-cts (wp (Bind S ( $\lambda S. \text{SetDC } a \ (\lambda\cdot$ . S))))
by(iprover intro!:cts-wp-Bind cts-wp-SetDC-const)
thus ?thesis by(simp add:wp-SetDC-Bind[symmetric])
qed

```

```

lemma cts-wp-repeat:
bd-cts (wp a)  $\implies$  healthy (wp a)  $\implies$  bd-cts (wp (repeat n a))
by(induct n, auto intro:cts-wp-Skip cts-wp-Seq healthy-intros)

```

```

lemma cts-wp-Embed:
bd-cts t  $\implies$  bd-cts (wp (Embed t))
by(simp add:wp-eval)

```

4.3.2 Continuity of a Single Loop Step

A single loop iteration is continuous, in the more general sense defined above for transformer transformers.

lemma *cts-wp-loopstep*:

fixes *body*::'s prog

assumes *hb*: healthy (wp *body*)

and *cb*: bd-cts (wp *body*)

shows *bd-cts-tr* ($\lambda x. \text{wp } (\text{body} ;; \text{Embed } x \ll G \gg \oplus \text{Skip}))$) (is *bd-cts-tr* ?*F*)

proof(rule *bd-cts-trI*, rule *le-trans-antisym*)

fix *M*::nat \Rightarrow 's trans **and** *b*::real

assume *chain*: $\bigwedge i. \text{le-trans } (M \ i) \ (M \ (\text{Suc } i))$

and *fM*: $\bigwedge i. \text{feasible } (M \ i)$

show *fw*: *le-trans* (*Sup-trans* (range (?*F* o *M*))) (?*F* (*Sup-trans* (range *M*)))

proof(rule *le-transI*[*OF* *Sup-trans-least2*], *clarsimp*)

fix *P Q*::'s expect **and** *t*

assume *sP*: sound *P*

assume *nQ*: nneg *Q* **and** *bP*: bounded-by (bound-of *P*) *Q*

hence *sQ*: sound *Q* **by**(*auto*)

from *fM* **have** *fSup*: feasible (*Sup-trans* (range *M*))

by(*auto* intro:*feasible-Sup-trans*)

from *sQ fM* **have** *M t Q* \Vdash *Sup-trans* (range *M*) *Q*

by(*auto* intro:*Sup-trans-upper2*)

moreover **from** *sQ fM fSup*

have *sMtP*: sound (*M t Q*) sound (*Sup-trans* (range *M*) *Q*) **by**(*auto*)

ultimately **have** wp *body* (*M t Q*) \Vdash wp *body* (*Sup-trans* (range *M*) *Q*)

using *healthy-monoD*[*OF* *hb*] **by**(*auto*)

hence $\bigwedge s. \text{wp } \text{body} \ (M \ t \ Q) \ s \leq \text{wp } \text{body} \ (\text{Sup-trans} \ (\text{range } M) \ Q) \ s$

by(rule *le-funD*)

thus ?*F* (*M t*) *Q* \Vdash ?*F* (*Sup-trans* (range *M*)) *Q*

by(intro *le-funI*, *simp* add:*wp-eval* *mult-left-mono*)

show nneg (wp (*body* ;; *Embed* (*Sup-trans* (range *M*)) $\ll G \gg \oplus$ *Skip*) *Q*)

proof(rule *nnegI*, *simp* add:*wp-eval*)

fix *s*::'s

from *fSup sQ* **have** sound (*Sup-trans* (range *M*) *Q*) **by**(*auto*)

with *hb* **have** sound (wp *body* (*Sup-trans* (range *M*) *Q*)) **by**(*auto*)

hence $0 \leq \text{wp } \text{body} \ (\text{Sup-trans} \ (\text{range } M) \ Q) \ s$ **by**(*auto*)

moreover **from** *sQ* **have** $0 \leq Q \ s$ **by**(*auto*)

ultimately **show** $0 \leq \ll G \gg \ s * \text{wp } \text{body} \ (\text{Sup-trans} \ (\text{range } M) \ Q) \ s + (1 - \ll G \gg \ s) *$

Q s

by(*auto* intro:*add-nonneg-nonneg* *mult-nonneg-nonneg*)

qed

next

fix *P*::'s expect **assume** *sP*: sound *P*

thus nneg *P* bounded-by (bound-of *P*) *P* **by**(*auto*)

show $\forall u \in \text{range} \ ((\lambda x. \text{wp } (\text{body} ;; \text{Embed } x \ll G \gg \oplus \text{Skip})) \circ M)$.

$\forall R. \text{nneg } R \wedge \text{bounded-by (bound-of } P) R \longrightarrow$
 $\text{nneg (} u R) \wedge \text{bounded-by (bound-of } P) (u R)$
proof(*clarsimp, intro conjI nnegI bounded-byI, simp-all add:wp-eval*)
fix $u::\text{nat}$ **and** $R::\text{'s}$ **expect** **and** $s::\text{'s}$
assume $nR: \text{nneg } R$ **and** $bR: \text{bounded-by (bound-of } P) R$
hence $sR: \text{sound } R$ **by**(*auto*)
with fM **have** $sMuR: \text{sound (} M u R)$ **by**(*auto*)
with hb **have** $\text{sound (wp body (} M u R))$ **by**(*auto*)
hence $0 \leq \text{wp body (} M u R) s$ **by**(*auto*)
moreover from nR **have** $0 \leq R s$ **by**(*auto*)
ultimately show $0 \leq \llbracket G \rrbracket s * \text{wp body (} M u R) s + (1 - \llbracket G \rrbracket s) * R s$
by(*auto intro:add-nonneg-nonneg mult-nonneg-nonneg*)

from sR bR fM **have** $\text{bounded-by (bound-of } P) (M u R)$ **by**(*auto*)
with $sMuR$ hb **have** $\text{bounded-by (bound-of } P) (\text{wp body (} M u R))$ **by**(*auto*)
hence $\text{wp body (} M u R) s \leq \text{bound-of } P$ **by**(*auto*)
moreover from bR **have** $R s \leq \text{bound-of } P$ **by**(*auto*)
ultimately have $\llbracket G \rrbracket s * \text{wp body (} M u R) s + (1 - \llbracket G \rrbracket s) * R s \leq$
 $\llbracket G \rrbracket s * \text{bound-of } P + (1 - \llbracket G \rrbracket s) * \text{bound-of } P$
by(*auto intro:add-mono mult-left-mono*)
also have $\dots = \text{bound-of } P$ **by**(*simp add:algebra-simps*)
finally show $\llbracket G \rrbracket s * \text{wp body (} M u R) s + (1 - \llbracket G \rrbracket s) * R s \leq \text{bound-of } P$.
qed
qed

show $\text{le-trans (?F (Sup-trans (range } M))) (Sup-trans (range (?F o M)))$
proof(*rule le-transI, rule le-funI, simp add: wp-eval cong del: image-cong-simp*)
fix $P::\text{'s}$ **expect** **and** $s::\text{'s}$
assume $sP: \text{sound } P$
have $\{t P \mid t \in \text{range } M\} = \text{range } (\lambda i. M i P)$
by(*blast*)
hence $\text{wp body (Sup-trans (range } M) P) s = \text{wp body (Sup-exp (range } (\lambda i. M i P))) s$
by(*simp add:Sup-trans-def*)
also {
from sP fM **have** $\bigwedge i. \text{sound (} M i P)$ **by**(*auto*)
moreover from sP **chain** **have** $\bigwedge i. M i P \Vdash M (Suc i) P$ **by**(*auto*)
moreover {
from sP **have** $\text{bounded-by (bound-of } P) P$ **by**(*auto*)
with sP fM **have** $\bigwedge i. \text{bounded-by (bound-of } P) (M i P)$ **by**(*auto*)
}
ultimately have $\text{wp body (Sup-exp (range } (\lambda i. M i P))) s =$
 $\text{Sup-exp (range } (\lambda i. \text{wp body (} M i P))) s$
by(*subst bd-ctsD[OF cb], auto simp:o-def*)
}
also have $\text{Sup-exp (range } (\lambda i. \text{wp body (} M i P))) s =$
 $\text{Sup } \{f s \mid f \in \text{range } (\lambda i. \text{wp body (} M i P))\}$
by(*simp add:Sup-exp-def*)
finally have $\llbracket G \rrbracket s * \text{wp body (Sup-trans (range } M) P) s + (1 - \llbracket G \rrbracket s) * P s =$
 $\llbracket G \rrbracket s * \text{Sup } \{f s \mid f \in \text{range } (\lambda i. \text{wp body (} M i P))\} + (1 - \llbracket G \rrbracket s) * P s$

```

by(simp)
also {
  from  $sP fM$  have  $\bigwedge i. \text{sound } (M i P)$  by(auto)
  moreover from  $sP fM$  have  $\bigwedge i. \text{bounded-by } (\text{bound-of } P) (M i P)$  by(auto)
  ultimately have  $\bigwedge i. \text{bounded-by } (\text{bound-of } P) (\text{wp body } (M i P))$  using  $hb$  by(auto)
  hence bound:  $\bigwedge i. \text{wp body } (M i P) s \leq \text{bound-of } P$  by(auto)
  moreover
  have  $\{\langle\langle G \rangle\rangle s * x \mid x. x \in \{f s \mid f. f \in \text{range } (\lambda i. \text{wp body } (M i P))\}\} =$ 
     $\{\langle\langle G \rangle\rangle s * f s \mid f. f \in \text{range } (\lambda i. \text{wp body } (M i P))\}$ 
    by(blast)
  ultimately
  have  $\langle\langle G \rangle\rangle s * \text{Sup } \{f s \mid f. f \in \text{range } (\lambda i. \text{wp body } (M i P))\} =$ 
     $\text{Sup } \{\langle\langle G \rangle\rangle s * f s \mid f. f \in \text{range } (\lambda i. \text{wp body } (M i P))\}$ 
    by(subst cSup-mult, auto)
  moreover {
    have  $\{x + (I - \langle\langle G \rangle\rangle s) * P s \mid x.$ 
       $x \in \{\langle\langle G \rangle\rangle s * f s \mid f. f \in \text{range } (\lambda i. \text{wp body } (M i P))\}\} =$ 
       $\{\langle\langle G \rangle\rangle s * f s + (I - \langle\langle G \rangle\rangle s) * P s \mid f. f \in \text{range } (\lambda i. \text{wp body } (M i P))\}$ 
      by(blast)
    moreover from  $\text{bound } sP$  have  $\bigwedge i. \langle\langle G \rangle\rangle s * \text{wp body } (M i P) s \leq \text{bound-of } P$ 
      by(cases G s, auto)
    ultimately
    have  $\text{Sup } \{\langle\langle G \rangle\rangle s * f s \mid f. f \in \text{range } (\lambda i. \text{wp body } (M i P))\} + (I - \langle\langle G \rangle\rangle s) * P s =$ 
       $\text{Sup } \{\langle\langle G \rangle\rangle s * f s + (I - \langle\langle G \rangle\rangle s) * P s \mid f. f \in \text{range } (\lambda i. \text{wp body } (M i P))\}$ 
      by(subst cSup-add, auto)
  }
  ultimately
  have  $\langle\langle G \rangle\rangle s * \text{Sup } \{f s \mid f. f \in \text{range } (\lambda i. \text{wp body } (M i P))\} + (I - \langle\langle G \rangle\rangle s) * P s =$ 
     $\text{Sup } \{\langle\langle G \rangle\rangle s * f s + (I - \langle\langle G \rangle\rangle s) * P s \mid f. f \in \text{range } (\lambda i. \text{wp body } (M i P))\}$ 
    by(simp)
}
also {
  have  $\bigwedge i. \langle\langle G \rangle\rangle s * \text{wp body } (M i P) s + (I - \langle\langle G \rangle\rangle s) * P s =$ 
     $((\lambda x. \text{wp } (\text{body} ;; \text{Embed } x \langle\langle G \rangle\rangle \oplus \text{Skip})) \circ M) i P s$ 
    by(simp add:wp-eval)
  also have  $\bigwedge i. \dots i \leq$ 
     $\text{Sup } \{f s \mid f. f \in \{t P \mid t. t \in \text{range } ((\lambda x. \text{wp } (\text{body} ;; \text{Embed } x \langle\langle G \rangle\rangle \oplus \text{Skip})) \circ$ 
     $M)\}\}$ 
}
proof(intro cSup-upper bdd-aboveI, blast, clarsimp simp:wp-eval)
fix  $i$ 
from  $sP$  have  $bP: \text{bounded-by } (\text{bound-of } P) P$  by(auto)
with  $sP fM$  have  $\text{sound } (M i P) \text{ bounded-by } (\text{bound-of } P) (M i P)$  by(auto)
with  $hb$  have  $\text{bounded-by } (\text{bound-of } P) (\text{wp body } (M i P))$  by(auto)
with  $bP$  have  $\text{wp body } (M i P) s \leq \text{bound-of } P P s \leq \text{bound-of } P$  by(auto)
hence  $\langle\langle G \rangle\rangle s * \text{wp body } (M i P) s + (I - \langle\langle G \rangle\rangle s) * P s \leq$ 
   $\langle\langle G \rangle\rangle s * (\text{bound-of } P) + (I - \langle\langle G \rangle\rangle s) * (\text{bound-of } P)$ 
  by(auto intro:add-mono mult-left-mono)
also have  $\dots = \text{bound-of } P$  by(simp add:algebra-simps)
finally show  $\langle\langle G \rangle\rangle s * \text{wp body } (M i P) s + (I - \langle\langle G \rangle\rangle s) * P s \leq \text{bound-of } P.$ 

```

```

qed
finally
have  $Sup \{ \llbracket G \rrbracket s * f s + (I - \llbracket G \rrbracket s) * P s \mid f. f \in range (\lambda i. wp \ body \ (M \ i \ P)) \} \leq$ 
 $Sup \{ f s \mid f. f \in \{ t P \mid t. t \in range ((\lambda x. wp \ (body ;; Embed \ x \ \llbracket G \rrbracket \oplus Skip)) \circ M) \} \}$ 
by (blast intro:cSup-least)
}
also have  $Sup \{ f s \mid f. f \in \{ t P \mid t. t \in range ((\lambda x. wp \ (body ;; Embed \ x \ \llbracket G \rrbracket \oplus Skip)) \circ M) \} \} =$ 
 $Sup-trans \ (range \ ((\lambda x. wp \ (body ;; Embed \ x \ \llbracket G \rrbracket \oplus Skip)) \circ M)) \ P \ s$ 
by (simp add:Sup-trans-def Sup-exp-def)
finally show  $\llbracket G \rrbracket s * wp \ body \ (Sup-trans \ (range \ M) \ P) \ s + (I - \llbracket G \rrbracket s) * P \ s \leq$ 
 $Sup-trans \ (range \ ((\lambda x. wp \ (body ;; Embed \ x \ \llbracket G \rrbracket \oplus Skip)) \circ M)) \ P \ s .$ 
qed
qed
end

```

4.4 Continuity and Induction for Loops

theory *LoopInduction* **imports** *Healthiness Continuity* **begin**

Showing continuity for loops requires a stronger induction principle than we have used so far, which in turn relies on the continuity of loops (inductively). Thus, the proofs are intertwined, and broken off from the main set of continuity proofs. This result is also essential in showing the sublinearity of loops.

A loop step is monotonic.

lemma *wp-loop-step-mono-trans*:

fixes *body::'s prog*

assumes *sP: sound P*

and *hb: healthy (wp body)*

shows *mono-trans* $(\lambda Q \ s. \llbracket G \rrbracket s * wp \ body \ Q \ s + \llbracket \mathcal{N} \ G \rrbracket s * P \ s)$

proof (*intro mono-transI le-funI, simp*)

fix *Q R::'s expect* **and** *s::'s*

assume *sQ: sound Q* **and** *sR: sound R* **and** *le: Q \vdash R*

hence *wp body Q \vdash wp body R*

by (*rule mono-transD[OF healthy-monoD, OF hb]*)

thus $\llbracket G \rrbracket s * wp \ body \ Q \ s \leq \llbracket G \rrbracket s * wp \ body \ R \ s$

by (*auto dest:le-funD intro:mult-left-mono*)

qed

We can therefore apply the standard fixed-point lemmas to unfold it:

lemma *lfp-wp-loop-unfold*:

fixes *body::'s prog*

assumes *hb: healthy (wp body)*

and *sP: sound P*

shows *lfp-exp* $(\lambda Q \ s. \llbracket G \rrbracket s * wp \ body \ Q \ s + \llbracket \mathcal{N} \ G \rrbracket s * P \ s) =$

$(\lambda s. \llbracket G \rrbracket s * wp \ body \ (lfp-exp \ (\lambda Q \ s. \llbracket G \rrbracket s * wp \ body \ Q \ s + \llbracket \mathcal{N} \ G \rrbracket s * P \ s)) \ s +$

```

      « $\mathcal{N} G$ »  $s * P s$ )
proof(rule lfp-exp-unfold)
  from assms show mono-trans ( $\lambda Q s. \llbracket G \rrbracket s * wp \text{ body } Q s + \llbracket \mathcal{N} G \rrbracket s * P s$ )
    by(blast intro:wp-loop-step-mono-trans)
  from assms show  $\lambda s. \llbracket G \rrbracket s * wp \text{ body } (\lambda s. \text{bound-of } P) s + \llbracket \mathcal{N} G \rrbracket s * P s \Vdash \lambda s.$ 
bound-of P
    by(blast intro:lfp-loop-fp)
  from sP show sound ( $\lambda s. \text{bound-of } P$ )
    by(auto)
  fix  $Q::'s \text{ expect}$ 
  assume sound Q
  with assms show sound ( $\lambda s. \llbracket G \rrbracket s * wp \text{ body } Q s + \llbracket \mathcal{N} G \rrbracket s * P s$ )
    by(intro wp-loop-step-sound[unfolded wp-eval, simplified, folded negate-embed], auto)
qed

```

```

lemma wp-loop-step-unitary:
  fixes  $body::'s \text{ prog}$ 
  assumes hb: healthy ( $wp \text{ body}$ )
    and  $uP: \text{unitary } P$  and  $uQ: \text{unitary } Q$ 
  shows unitary ( $\lambda s. \llbracket G \rrbracket s * wp \text{ body } Q s + \llbracket \mathcal{N} G \rrbracket s * P s$ )
proof(intro unitaryI2 nnegI bounded-byI)
  fix  $s::'s$ 
  from  $uQ \text{ hb}$  have  $uwQ: \text{unitary } (wp \text{ body } Q)$  by(auto)
  with  $uP$  have  $0 \leq wp \text{ body } Q s \leq P s$  by(auto)
  thus  $0 \leq \llbracket G \rrbracket s * wp \text{ body } Q s + \llbracket \mathcal{N} G \rrbracket s * P s$ 
    by(auto intro:add-nonneg-nonneg mult-nonneg-nonneg)

  from  $uP \text{ uwQ}$  have  $wp \text{ body } Q s \leq I P s \leq I$  by(auto)
  hence  $\llbracket G \rrbracket s * wp \text{ body } Q s + \llbracket \mathcal{N} G \rrbracket s * P s \leq \llbracket G \rrbracket s * I + \llbracket \mathcal{N} G \rrbracket s * I$ 
    by(blast intro:add-mono mult-left-mono)
  also have  $\dots = I$  by(simp add:negate-embed)
  finally show  $\llbracket G \rrbracket s * wp \text{ body } Q s + \llbracket \mathcal{N} G \rrbracket s * P s \leq I$  .
qed

```

```

lemma lfp-loop-unitary:
  fixes  $body::'s \text{ prog}$ 
  assumes hb: healthy ( $wp \text{ body}$ )
    and  $uP: \text{unitary } P$ 
  shows unitary ( $lfp\text{-exp } (\lambda Q s. \llbracket G \rrbracket s * wp \text{ body } Q s + \llbracket \mathcal{N} G \rrbracket s * P s)$ )
  using assms by(blast intro:lfp-exp-unitary wp-loop-step-unitary)

```

From the lattice structure on transformers, we establish a transfinite induction principle for loops. We use this to show a number of properties, particularly subdistributivity, for loops. This proof follows the pattern of lemma `lfp_ordinal_induct` in HOL/Inductive.

```

lemma loop-induct:
  fixes  $body::'s \text{ prog}$ 
  assumes hwp: healthy ( $wp \text{ body}$ )
    and hwlp: nearly-healthy ( $wlp \text{ body}$ )

```

— The body must be healthy, both in strict and liberal semantics.
and *Limit*: $\bigwedge S. \llbracket \forall x \in S. P (fst\ x) (snd\ x); \forall x \in S. feasible\ (fst\ x); \forall x \in S. \forall Q. unitary\ Q \longrightarrow unitary\ (snd\ x\ Q) \rrbracket \Longrightarrow P (Sup-trans\ (fst\ 'S)) (Inf-utrans\ (snd\ 'S))$

— The property holds at limit points.
and *IH*: $\bigwedge t\ u. \llbracket P\ t\ u; feasible\ t; \bigwedge Q. unitary\ Q \Longrightarrow unitary\ (u\ Q) \rrbracket \Longrightarrow P (wp\ (body\ ;;\ Embed\ t\ \llcorner G \gg \oplus\ Skip)) (wlp\ (body\ ;;\ Embed\ u\ \llcorner G \gg \oplus\ Skip))$

— The inductive step. The property is preserved by a single loop iteration.
and *P-equiv*: $\bigwedge t\ t'\ u\ u'. \llbracket P\ t\ u; equiv-trans\ t\ t'; equiv-utrans\ u\ u' \rrbracket \Longrightarrow P\ t'\ u'$

— The property must be preserved by equivalence
shows $P (wp\ (do\ G \longrightarrow body\ od)) (wlp\ (do\ G \longrightarrow body\ od))$

— The property can refer to both interpretations simultaneously. The unifier will happily apply the rule to just one or the other, however.
proof(*simp add:wp-eval*)
let $?X\ t = wp\ (body\ ;;\ Embed\ t\ \llcorner G \gg \oplus\ Skip)$
let $?Y\ t = wlp\ (body\ ;;\ Embed\ t\ \llcorner G \gg \oplus\ Skip)$

let $?M = \{x. P (fst\ x) (snd\ x) \wedge feasible\ (fst\ x) \wedge (\forall Q. unitary\ Q \longrightarrow unitary\ (snd\ x\ Q)) \wedge le-trans\ (fst\ x) (lfp-trans\ ?X) \wedge le-utrans\ (gfp-trans\ ?Y) (snd\ x)\}$

have $fSup: feasible\ (Sup-trans\ (fst\ ' ?M))$
proof(*intro feasibleI bounded-byI2 nnegI2*)
fix $Q::'s\ expect$ **and** $b::real$
assume $nQ: nneg\ Q$ **and** $bQ: bounded-by\ b\ Q$
show $Sup-trans\ (fst\ ' ?M)\ Q \Vdash \lambda s. b$
unfolding *Sup-trans-def*
using $nQ\ bQ$ **by**(*auto intro!:Sup-exp-least*)
show $\lambda s. 0 \Vdash Sup-trans\ (fst\ ' ?M)\ Q$
proof(*cases*)
assume *empty*: $?M = \{\}$
show *thesis* **by**(*simp add:Sup-trans-def Sup-exp-def empty*)

next
assume *ne*: $?M \neq \{\}$
then obtain x **where** $xin: x \in ?M$ **by** *auto*
hence $ffx: feasible\ (fst\ x)$ **by**(*simp*)
with $nQ\ bQ$ **have** $\lambda s. 0 \Vdash fst\ x\ Q$ **by**(*auto*)
also from xin **have** $fst\ x\ Q \Vdash Sup-trans\ (fst\ ' ?M)\ Q$
apply(*intro Sup-trans-upper2[OF imageI - nQ bQ], assumption*)
apply(*clarsimp, blast intro: sound-nneg[OF feasible-sound] feasible-boundedD*)
done
finally show $\lambda s. 0 \Vdash Sup-trans\ (fst\ ' ?M)\ Q$.

qed
qed

have $uInf: \bigwedge P. unitary\ P \Longrightarrow unitary\ (Inf-utrans\ (snd\ ' ?M)\ P)$

```

proof(cases ?M = {})
  fix P
  assume empty: ?M = {}
  show ?thesis P by(simp only:empty, simp add:Inf-utrans-def)
next
  fix P::'s expect
  assume uP: unitary P
  and ne: ?M ≠ {}
  show ?thesis P
  proof(intro unitaryI2 nnegI2 bounded-byI2)
  from ne obtain x where xin: x ∈ ?M by auto
  hence sxin: snd x ∈ snd ' ?M by(simp)
  hence le-utrans (Inf-utrans (snd ' ?M)) (snd x)
  by(intro Inf-utrans-lower, auto)
  with uP
  have Inf-utrans (snd ' ?M) P ⊨ snd x P by(auto)
  also {
    from xin uP have unitary (snd x P) by(simp)
    hence snd x P ⊨ λs. I by(auto)
  }
  finally show Inf-utrans (snd ' ?M) P ⊨ λs. I .

  have λs. 0 ⊨ Inf-trans (snd ' ?M) P
  unfolding Inf-trans-def
  proof(rule Inf-exp-greatest)
  from sxin show {t P |t. t ∈ snd ' ?M} ≠ {} by(auto)
  show ∀ P∈{t P |t. t ∈ snd ' ?M}. λs. 0 ⊨ P
  proof(clarsimp)
  fix t::'s trans
  assume ∀ Q. unitary Q ⟶ unitary (t Q)
  with uP have unitary (t P) by(auto)
  thus λs. 0 ⊨ t P by(auto)
  qed
  qed
  also {
    from ne have X: (snd ' ?M = {}) = False by(simp)
    have Inf-trans (snd ' ?M) P = Inf-utrans (snd ' ?M) P
    unfolding Inf-utrans-def by(subst X, simp)
  }
  finally show λs. 0 ⊨ Inf-utrans (snd ' ?M) P .
  qed
qed

have wp-loop-mono: ∧t u. [ le-trans t u; ∧P. sound P ⟹ sound (t P);
  ∧P. sound P ⟹ sound (u P) ] ⟹ le-trans (?X t) (?X u)
proof(intro le-transI le-funI, simp add:wp-eval)
  fix t u::'s trans and P::'s expect and s::'s
  assume le: le-trans t u
  and st: ∧P. sound P ⟹ sound (t P)

```

and $su: \bigwedge P. \text{sound } P \implies \text{sound } (u P)$
and $sP: \text{sound } P$
hence $\text{sound } (t P) \text{ sound } (u P) \mathbf{by}(auto)$
with $healthy\text{-monoD}[OF \text{ hwp}] \text{ le } sP \mathbf{have} \text{ wp } body (t P) \Vdash \text{ wp } body (u P) \mathbf{by}(auto)$
hence $\text{wp } body (t P) s \leq \text{wp } body (u P) s \mathbf{by}(auto)$
thus $\ll G \gg s * \text{wp } body (t P) s \leq \ll G \gg s * \text{wp } body (u P) s \mathbf{by}(auto \text{ intro:mult-left-mono})$
qed

have $wlp\text{-loop-mono}: \bigwedge t u. \llbracket \text{le-utrans } t u; \bigwedge P. \text{unitary } P \implies \text{unitary } (t P);$
 $\bigwedge P. \text{unitary } P \implies \text{unitary } (u P) \rrbracket \implies \text{le-utrans } (?Y t) (?Y u)$

proof(*intro le-utransI le-funI, simp add:wp-eval*)

fix $t u::'s \text{ trans}$ **and** $P::'s \text{ expect}$ **and** $s::'s$

assume $le: \text{le-utrans } t u$

and $ut: \bigwedge P. \text{unitary } P \implies \text{unitary } (t P)$

and $uu: \bigwedge P. \text{unitary } P \implies \text{unitary } (u P)$

and $uP: \text{unitary } P$

hence $\text{unitary } (t P) \text{ unitary } (u P) \mathbf{by}(auto)$

with $le \ uP \mathbf{have} \text{ wlp } body (t P) \Vdash \text{ wlp } body (u P)$

by(*auto intro:nearly-healthy-monoD[OF hwlp]*)

hence $\text{wlp } body (t P) s \leq \text{wlp } body (u P) s \mathbf{by}(auto)$

thus $\ll G \gg s * \text{wlp } body (t P) s \leq \ll G \gg s * \text{wlp } body (u P) s$

by(*auto intro:mult-left-mono*)

qed

from $hwp \mathbf{have} \text{ hX}: \bigwedge t. \text{healthy } t \implies \text{healthy } (?X t)$

by(*auto intro:healthy-intros*)

from $hwlp \mathbf{have} \text{ hY}: \bigwedge t. \text{nearly-healthy } t \implies \text{nearly-healthy } (?Y t)$

by(*auto intro!:healthy-intros*)

have $PLimit: P (\text{Sup-trans } (fst \text{ ' } ?M)) (\text{Inf-utrans } (snd \text{ ' } ?M))$

by(*auto intro:Limit*)

have $feasible\text{-lfp-loop}:$

$feasible (\text{lfp-trans } ?X)$

proof(*intro feasibleI bounded-byI2 nnegI2,*

simp-all add:wp-LoopI[simplified wp-eval] soundI2 hwp)

fix $P::'s \text{ expect}$ **and** $b::\text{real}$

assume $bP: \text{bounded-by } b P$ **and** $nP: \text{nneg } P$

hence $sP: \text{sound } P \mathbf{by}(auto)$

show $\text{lfp-exp } (\lambda Q s. \ll G \gg s * \text{wp } body Q s + \ll \mathcal{N} G \gg s * P s) \Vdash \lambda s. b$

proof(*intro lfp-exp-lowerbound le-funI*)

fix $s::'s$

from $bP \ nP \mathbf{have} \text{ nnb}: 0 \leq b \mathbf{by}(auto)$

hence $\text{sound } (\lambda s. b) \text{ bounded-by } b (\lambda s. b) \mathbf{by}(auto)$

with $hwp \mathbf{have} \text{ bounded-by } b (\text{wp } body (\lambda s. b)) \mathbf{by}(auto)$

with $bP \mathbf{have} \text{ wp } body (\lambda s. b) s \leq b P s \leq b \mathbf{by}(auto)$

hence $\ll G \gg s * \text{wp } body (\lambda s. b) s + \ll \mathcal{N} G \gg s * P s \leq \ll G \gg s * b + \ll \mathcal{N} G \gg s * b$

by(*auto intro:add-mono mult-left-mono*)

thus $\langle G \rangle s * wp \text{ body } (\lambda s. b) s + \langle \mathcal{N} G \rangle s * P s \leq b$
by(*simp add:negate-embed algebra-simps*)
from *nmb show sound* ($\lambda s. b$) **by**(*auto*)
qed
from *hwp sP show* $\lambda s. 0 \Vdash lfp\text{-exp } (\lambda Q s. \langle G \rangle s * wp \text{ body } Q s + \langle \mathcal{N} G \rangle s * P s)$
by(*blast intro!:lfp-exp-greatest lfp-loop-fp*)
qed

have *unitary-gfp*:
 $\bigwedge P. \text{unitary } P \implies \text{unitary } (\text{gfp-trans } ?Y P)$
proof(*intro unitaryI2 nnegI2 bounded-byI2,*
simp-all add:wlp-LoopI[simplified wp-eval] hwp)
fix $P::'s \text{ expect}$
assume $uP: \text{unitary } P$
show $0 \Vdash \text{gfp-exp } (\lambda Q s. \langle G \rangle s * wlp \text{ body } Q s + \langle \mathcal{N} G \rangle s * P s)$
proof(*rule gfp-exp-upperbound[OF le-funI]*)
fix $s::'s$
from *hwp uP have* $0 \leq wlp \text{ body } (\lambda s. 0) s 0 \leq P s$ **by**(*auto dest!:unitary-sound*)
thus $0 \leq \langle G \rangle s * wlp \text{ body } (\lambda s. 0) s + \langle \mathcal{N} G \rangle s * P s$
by(*auto intro:add-nonneg-nonneg mult-nonneg-nonneg*)
show *unitary* ($\lambda s. 0$) **by**(*auto*)
qed
show *gfp-exp* ($\lambda Q s. \langle G \rangle s * wlp \text{ body } Q s + \langle \mathcal{N} G \rangle s * P s$) $\Vdash \lambda s. 1$
by(*auto intro:gfp-exp-least*)
qed

have *fX*:
 $\bigwedge t. \text{feasible } t \implies \text{feasible } (?X t)$
proof(*intro feasibleI nnegI bounded-byI, simp-all add:wp-eval*)
fix $t::'s \text{ trans}$ **and** $Q::'s \text{ expect}$ **and** $b::\text{real}$ **and** $s::'s$
assume $ft: \text{feasible } t$ **and** $bQ: \text{bounded-by } b Q$ **and** $nQ: \text{nneg } Q$
hence $\text{nneg } (t Q)$ *bounded-by* $b (t Q)$ **by**(*auto*)
moreover **hence** $stQ: \text{sound } (t Q)$ **by**(*auto*)
ultimately **have** $wp \text{ body } (t Q) s \leq b$ **using** *hwp* **by**(*auto*)
moreover **from** bQ **have** $Q s \leq b$ **by**(*auto*)
ultimately **have** $\langle G \rangle s * wp \text{ body } (t Q) s + (1 - \langle G \rangle s) * Q s \leq$
 $\langle G \rangle s * b + (1 - \langle G \rangle s) * b$
by(*auto intro:add-mono mult-left-mono*)
thus $\langle G \rangle s * wp \text{ body } (t Q) s + (1 - \langle G \rangle s) * Q s \leq b$
by(*simp add:algebra-simps*)

from $nQ stQ hwp$ **have** $0 \leq wp \text{ body } (t Q) s 0 \leq Q s$ **by**(*auto*)
thus $0 \leq \langle G \rangle s * wp \text{ body } (t Q) s + (1 - \langle G \rangle s) * Q s$
by(*auto intro:add-nonneg-nonneg mult-nonneg-nonneg*)
qed

have *uY*:
 $\bigwedge t P. (\bigwedge P. \text{unitary } P \implies \text{unitary } (t P)) \implies \text{unitary } P \implies \text{unitary } (?Y t P)$
proof(*intro unitaryI2 nnegI bounded-byI, simp-all add:wp-eval*)


```

fix  $t::'s$  trans and  $P::'s$  expect and  $s::'s$ 
assume  $ut: \bigwedge P. \text{unitary } P \implies \text{unitary } (t P)$ 
and  $uP: \text{unitary } P$ 
hence  $utP: \text{unitary } (t P)$  by(auto)
with  $hwlp$  have  $ubtP: \text{unitary } (wlp \text{ body } (t P))$  by(auto)
with  $uP$  have  $0 \leq P s \ 0 \leq wlp \text{ body } (t P) s$  by(auto)
thus  $0 \leq \langle G \rangle s * wlp \text{ body } (t P) s + (I - \langle G \rangle s) * P s$ 
by(auto intro:add-nonneg-nonneg mult-nonneg-nonneg)

from  $uP \ ubtP$  have  $P s \leq I \ wlp \text{ body } (t P) s \leq I$  by(auto)
hence  $\langle G \rangle s * wlp \text{ body } (t P) s + (I - \langle G \rangle s) * P s \leq \langle G \rangle s * I + (I - \langle G \rangle s) * I$ 
by(blast intro:add-mono mult-left-mono)
also have  $\dots = I$  by(simp add:algebra-simps)
finally show  $\langle G \rangle s * wlp \text{ body } (t P) s + (I - \langle G \rangle s) * P s \leq I$  .
qed

```

```

have  $fw\text{-lfp}: \text{le-trans } (Sup\text{-trans } (fst \ ' ?M)) (lfp\text{-trans } ?X)$ 
using feasible-nnegD[OF feasible-lfp-loop]
by(intro le-transI[OF Sup-trans-least2], blast+)
hence  $\text{le-trans } (?X (Sup\text{-trans } (fst \ ' ?M))) (?X (lfp\text{-trans } ?X))$ 
by(auto intro:wp-loop-mono feasible-sound[OF fSup]
feasible-sound[OF feasible-lfp-loop])
also have equiv-trans ... (lfp-trans ?X)
proof(rule iffD1[OF equiv-trans-comm, OF lfp-trans-unfold], iprover intro:wp-loop-mono)
fix  $t::'s$  trans and  $P::'s$  expect
assume  $st: \bigwedge Q. \text{sound } Q \implies \text{sound } (t Q)$ 
and  $sP: \text{sound } P$ 
show  $\text{sound } (?X t P)$ 
proof(intro soundI2 bounded-byI nnegI, simp-all add:wp-eval)
fix  $s::'s$ 
from  $sP \ st \ hwp$  have  $0 \leq P s \ 0 \leq wp \text{ body } (t P) s$  by(auto)
thus  $0 \leq \langle G \rangle s * wp \text{ body } (t P) s + (I - \langle G \rangle s) * P s$ 
by(blast intro:add-nonneg-nonneg mult-nonneg-nonneg)
from  $sP \ st$  have bounded-by (bound-of (t P)) (t P) by(auto)
with  $sP \ st \ hwp$  have bounded-by (bound-of (t P)) (wp body (t P)) by(auto)
hence  $wp \text{ body } (t P) s \leq \text{bound-of } (t P)$  by(auto)
moreover from  $sP \ st \ hwp$  have  $P s \leq \text{bound-of } P$  by(auto)
moreover have  $\langle G \rangle s \leq I \ I - \langle G \rangle s \leq I$  by(auto)
moreover from  $sP \ st \ hwp$  have  $0 \leq wp \text{ body } (t P) s \ 0 \leq P s$  by(auto)
moreover have  $(0::\text{real}) \leq I$  by(simp)
ultimately show  $\langle G \rangle s * wp \text{ body } (t P) s + (I - \langle G \rangle s) * P s \leq$ 
 $I * \text{bound-of } (t P) + I * \text{bound-of } P$ 
by(blast intro:add-mono mult-mono)
qed
next
let  $?fp = \lambda R s. \text{bound-of } R$ 
show  $\text{le-trans } (?X ?fp) ?fp$  by(auto intro:healthy-intros hwp)
fix  $P::'s$  expect assume  $\text{sound } P$ 
thus  $\text{sound } (?fp P)$  by(auto)

```

qed

finally have $le\text{-lfp}$: $le\text{-trans } (?X \text{ (Sup-trans (fst ' ?M))) (lfp-trans } ?X) .$

have $fw\text{-gfp}$: $le\text{-utrans } (gfp\text{-trans } ?Y) \text{ (Inf-utrans (snd ' ?M))}$
by($auto \text{ intro:Inf-utrans-greatest unitary-gfp}$)

have $equiv\text{-utrans } (gfp\text{-trans } ?Y) \text{ (?Y (gfp-trans } ?Y))$

by($auto \text{ intro!:gfp-trans-unfold wlp-loop-mono uY}$)

also from $fw\text{-gfp}$ **have** $le\text{-utrans } (?Y \text{ (gfp-trans } ?Y)) \text{ (?Y (Inf-utrans (snd ' ?M)))}$

by($auto \text{ intro:wlp-loop-mono uInf unitary-gfp}$)

finally have $ge\text{-gfp}$: $le\text{-utrans } (gfp\text{-trans } ?Y) \text{ (?Y (Inf-utrans (snd ' ?M)))} .$

from $PLimit \text{ fX uY fSup uInf}$ **have** $P \text{ (?X (Sup-trans (fst ' ?M))) } \text{ (?Y (Inf-utrans (snd ' ?M)))}$

by($iprover \text{ intro:IH}$)

moreover from $fSup$ **have** $feasible \text{ (?X (Sup-trans (fst ' ?M)))}$ **by**($rule \text{ fX}$)

moreover have $\bigwedge P. \text{ unitary } P \implies \text{unitary } (?Y \text{ (Inf-utrans (snd ' ?M))) } P$

by($auto \text{ intro:uY uInf}$)

moreover note $le\text{-lfp } ge\text{-gfp}$

ultimately have $pair\text{-in}$: $(?X \text{ (Sup-trans (fst ' ?M)), } ?Y \text{ (Inf-utrans (snd ' ?M)))} \in ?M$

by($simp$)

have $?X \text{ (Sup-trans (fst ' ?M))} \in \text{fst ' ?M}$

by($rule \text{ imageI[OF pair-in, of fst, simplified]}$)

hence $le\text{-trans } (?X \text{ (Sup-trans (fst ' ?M)))} \text{ (Sup-trans (fst ' ?M))}$

proof($rule \text{ le-transI[OF Sup-trans-upper2[where } t=?X \text{ (Sup-trans (fst ' ?M))}$
 $\text{and } S=\text{fst ' ?M}]])$)

fix P :: 's $expect$

assume sP : $sound \text{ } P$

thus $nneg \text{ } P$ **by**($auto$)

from sP **show** $bounded\text{-by } (bound\text{-of } P) \text{ } P$ **by**($auto$)

from sP **show** $\forall u \in \text{fst ' ?M}. \forall Q. \text{ nneg } Q \wedge \text{bounded-by } (bound\text{-of } P) \text{ } Q \implies$
 $\text{nneg } (u \text{ } Q) \wedge \text{bounded-by } (bound\text{-of } P) \text{ } (u \text{ } Q)$

by($auto$)

qed

hence $le\text{-trans } (lfp\text{-trans } ?X) \text{ (Sup-trans (fst ' ?M))}$

by($auto \text{ intro:lfp-trans-lowerbound feasible-sound[OF fSup]}$)

with $fw\text{-lfp}$ **have** eqt : $equiv\text{-trans } (Sup\text{-trans (fst ' ?M))} \text{ (lfp-trans } ?X)$

by($rule \text{ le-trans-antisym}$)

have $?Y \text{ (Inf-utrans (snd ' ?M))} \in \text{snd ' ?M}$

by($rule \text{ imageI[OF pair-in, of snd, simplified]}$)

hence $le\text{-utrans } (Inf\text{-utrans (snd ' ?M))} \text{ (?Y (Inf-utrans (snd ' ?M)))}$

by($intro \text{ Inf-utrans-lower, auto}$)

hence $le\text{-utrans } (Inf\text{-utrans (snd ' ?M))} \text{ (gfp-trans } ?Y)$

by($blast \text{ intro:gfp-trans-upperbound uInf}$)

with $fw\text{-gfp}$ **have** equ : $equiv\text{-utrans } (Inf\text{-utrans (snd ' ?M))} \text{ (gfp-trans } ?Y)$

by($auto \text{ intro:le-utrans-antisym}$)

from $PLimit \text{ eqt equ}$ **show** $P \text{ (lfp-trans } ?X) \text{ (gfp-trans } ?Y)$ **by**($rule \text{ } P\text{-equiv}$)

qed

4.4.1 The Limit of Iterates

The iterates of a loop are its sequence of finite unrollings. We show shortly that this converges on the least fixed point. This is enormously useful, as we can appeal to various properties of the finite iterates (which will follow by finite induction), which we can then transfer to the limit.

definition *iterates* :: 's prog \Rightarrow ('s \Rightarrow bool) \Rightarrow nat \Rightarrow 's trans
where *iterates body G i* = ((λx . wp (body ;; Embed x $\ll G \gg \oplus$ Skip)) \wedge^{\wedge} i) (λP s. 0)

lemma *iterates-0[simp]*:
iterates body G 0 = (λP s. 0)
by(*simp add:iterates-def*)

lemma *iterates-Suc[simp]*:
iterates body G (Suc i) = wp (body ;; Embed (iterates body G i) $\ll G \gg \oplus$ Skip)
by(*simp add:iterates-def*)

All iterates are healthy.

lemma *iterates-healthy*:
healthy (wp body) \implies *healthy (iterates body G i)*
by(*induct i, auto intro:healthy-intros*)

The iterates are an ascending chain.

lemma *iterates-increasing*:
fixes *body*::'s prog
assumes *hb*: *healthy (wp body)*
shows *le-trans (iterates body G i) (iterates body G (Suc i))*
proof(*induct i*)
show *le-trans (iterates body G 0) (iterates body G (Suc 0))*
proof(*simp add:iterates-def, rule le-transI*)
fix *P*::'s expect
assume *sound P*
with *hb* **have** *sound (wp (body ;; Embed (λP s. 0) $\ll G \gg \oplus$ Skip) P)*
by(*auto intro!:wp-loop-step-sound*)
thus λs . 0 \Vdash wp (body ;; Embed (λP s. 0) $\ll G \gg \oplus$ Skip) P
by(*auto*)
qed

fix *i*
assume *IH*: *le-trans (iterates body G i) (iterates body G (Suc i))*
have *equiv-trans (iterates body G (Suc i))*
 (*wp (body ;; Embed (iterates body G i) $\ll G \gg \oplus$ Skip)*)
by(*simp*)
also from *iterates-healthy[OF hb]*
have *le-trans ... (wp (body ;; Embed (iterates body G (Suc i)) $\ll G \gg \oplus$ Skip))*
by(*blast intro:wp-loop-step-mono[OF hb IH]*)
also have *equiv-trans ... (iterates body G (Suc (Suc i)))*
by(*simp*)

finally show $le\text{-}trans$ ($iterates$ $body$ G (Suc i)) ($iterates$ $body$ G (Suc (Suc i))) .
qed

lemma $wp\text{-}loop\text{-}step\text{-}bounded$:
fixes $t::'s$ $trans$ **and** $Q::'s$ $expect$
assumes nQ : $nneg$ Q
and bQ : $bounded\text{-}by$ b Q
and ht : $healthy$ t
and hb : $healthy$ (wp $body$)
shows $bounded\text{-}by$ b (wp ($body$;; $Embed$ t $\ll G \gg \oplus Skip$) Q)
proof($rule$ $bounded\text{-}byI$, $simp$ $add:wp\text{-}eval$)
fix $s::'s$
from nQ bQ **have** sQ : $sound$ Q **by**($auto$)
with bQ ht **have** $sound$ (t Q) $bounded\text{-}by$ b (t Q) **by**($auto$)
with hb **have** $bounded\text{-}by$ b (wp $body$ (t Q)) **by**($auto$)
with bQ **have** wp $body$ (t Q) $s \leq b$ Q $s \leq b$ **by**($auto$)
hence $\ll G \gg s * wp$ $body$ (t Q) $s + (1 - \ll G \gg s) * Q$ $s \leq$
 $\ll G \gg s * b + (1 - \ll G \gg s) * b$
by($auto$ $intro:add\text{-}mono$ $mult\text{-}left\text{-}mono$)
also have $\dots = b$ **by**($simp$ $add:algebra\text{-}simps$)
finally show $\ll G \gg s * wp$ $body$ (t Q) $s + (1 - \ll G \gg s) * Q$ $s \leq b$.
qed

This is the key result: The loop is equivalent to the supremum of its iterates. This proof follows the pattern of lemma $continuous_lfp$ in HOL/Library/Continuity.

lemma $lfp\text{-}iterates$:
fixes $body::'s$ $prog$
assumes hb : $healthy$ (wp $body$)
and cb : $bd\text{-}cts$ (wp $body$)
shows $equiv\text{-}trans$ (wp (do $G \longrightarrow body$ od)) ($Sup\text{-}trans$ ($range$ ($iterates$ $body$ G)))
(is $equiv\text{-}trans$ $?X$ $?Y$)
proof($rule$ $le\text{-}trans\text{-}antisym$)
let $?F = \lambda x. wp$ ($body$;; $Embed$ x $\ll G \gg \oplus Skip$)
let $?bot = \lambda(P::'s \Rightarrow real)$ $s::'s. 0::real$

have HF : $\bigwedge i. healthy$ (($?F$ \wedge i) $?bot$)
proof –
fix i **from** hb **show** ($?thesis$ i)
by($induct$ i , $simp\text{-}all$ $add:healthy\text{-}intros$)
qed

from $iterates\text{-}healthy$ [OF hb]
have $\bigwedge i. feasible$ ($iterates$ $body$ G i) **by**($auto$)
hence $fSup$: $feasible$ ($Sup\text{-}trans$ ($range$ ($iterates$ $body$ G)))
by($auto$ $intro:feasible\text{-}Sup\text{-}trans$)

{
fix i
have $le\text{-}trans$ (($?F$ \wedge i) $?bot$) $?X$

```

proof(induct i)
  show le-trans ((?F ^^ 0) ?bot) ?X
  proof(simp, intro le-transI)
    fix P::'s expect
    assume sound P
    with hb healthy-wp-loop
    have sound (wp ( $\mu$  x. body ;;  $x \ll G \gg \oplus \text{Skip}$ ) P)
      by(auto)
    thus  $\lambda s. 0 \Vdash \text{wp} (\mu x. \text{body} ;; x \ll G \gg \oplus \text{Skip}) P$ 
      by(auto)
  qed
  fix i
  assume IH: le-trans ((?F ^^ i) ?bot) ?X
  have equiv-trans ((?F ^^ (Suc i) ?bot) (?F ((?F ^^ i) ?bot)) by(simp)
  also have le-trans ... (?F ?X)
  proof(rule wp-loop-step-mono[OF hb IH])
    fix P::'s expect
    assume sP: sound P
    with hb healthy-wp-loop
    show sound (wp ( $\mu$  x. body ;;  $x \ll G \gg \oplus \text{Skip}$ ) P)
      by(auto)
    from sP show sound ((?F ^^ i) ?bot P)
      by(rule healthy-sound[OF HF])
  qed
  also {
    from hb have X: le-trans (wp (body ;; Embed ( $\lambda P s. \text{bound-of } P$ )  $\ll G \gg \oplus \text{Skip}$ ))
      ( $\lambda P s. \text{bound-of } P$ )
    by(intro le-transI, simp add:wp-eval, auto intro: lfp-loop-fp[unfolded negate-embed])
    have equiv-trans (?F ?X) ?X
    apply (simp only: wp-eval)
    by(intro iffDI[OF equiv-trans-comm, OF lfp-trans-unfold]
      wp-loop-step-mono[OF hb] wp-loop-step-sound[OF hb], (blast|rule X)+)
  }
  finally show le-trans ((?F ^^ (Suc i) ?bot) ?X .
  qed
}
hence  $\bigwedge i. \text{le-trans} (\text{iterates } \text{body } G \ i) (\text{wp } \text{do } G \longrightarrow \text{body } \text{od})$ 
by(simp add:iterates-def)
thus le-trans ?Y ?X
by(auto intro!:le-transI[OF Sup-trans-least2] sound-nneg
  healthy-sound[OF iterates-healthy, OF hb]
  healthy-bounded-byD[OF iterates-healthy, OF hb]
  healthy-sound[OF healthy-wp-loop] hb)

show le-trans ?X ?Y
proof(simp only: wp-eval, rule lfp-trans-lowerbound)
from hb cb have bd-cts-tr ?F by(rule cts-wp-loopstep)
with iterates-increasing[OF hb] iterates-healthy[OF hb]
have equiv-trans (?F ?Y) (Sup-trans (range (?F o (iterates body G))))

```

by (*auto intro!*: *healthy-feasibleD bd-cts-trD cong del: image-cong-simp*)
also have *le-trans* (*Sup-trans* (*range* (*?F o* (*iterates body G*)))) *?Y*
proof(*rule le-transI*)
fix *P::'s expect*
assume *sP: sound P*
show (*Sup-trans* (*range* (*?F o* (*iterates body G*)))) *P* \Vdash *?Y P*
proof(*rule Sup-trans-least2, clarsimp*)
show $\forall u \in \text{range} ((\lambda x. \text{wp} (\text{body} ;; \text{Embed } x \ll G \gg \oplus \text{Skip})) \circ \text{iterates body } G).$
 $\forall R. \text{nneg } R \wedge \text{bounded-by} (\text{bound-of } P) R \longrightarrow$
 $\text{nneg } (u R) \wedge \text{bounded-by} (\text{bound-of } P) (u R)$
proof(*clarsimp, intro conjI*)
fix *Q::'s expect and i*
assume *nQ: nneg Q and bQ: bounded-by (bound-of P) Q*
hence *sound Q by(auto)*
moreover from *iterates-healthy[OF hb]*
have $\bigwedge P. \text{sound } P \implies \text{sound} (\text{iterates body } G \ i \ P)$ **by**(*auto*)
moreover note *hb*
ultimately have *sound* (*wp* (*body* ;; *Embed* (*iterates body G i*) $\ll G \gg \oplus \text{Skip}$) *Q*)
by(*iprover intro:wp-loop-step-sound*)
thus *nneg* (*wp* (*body* ;; *Embed* (*iterates body G i*) $\ll G \gg \oplus \text{Skip}$) *Q*)
by(*auto*)
from *nQ bQ iterates-healthy[OF hb] hb*
show *bounded-by* (*bound-of P*) (*wp* (*body* ;; *Embed* (*iterates body G i*) $\ll G \gg \oplus \text{Skip}$) *Q*)
Q)
by(*rule wp-loop-step-bounded*)
qed
from *sP show nneg P bounded-by (bound-of P) P by(auto)*
next
fix *Q::'s expect*
assume *nQ: nneg Q and bQ: bounded-by (bound-of P) Q*
hence *sound Q by(auto)*
with *fSup have sound (Sup-trans (range (iterates body G)) Q) by(auto)*
thus *nneg (Sup-trans (range (iterates body G)) Q) by(auto)*

fix *i*
show *wp* (*body* ;; *Embed* (*iterates body G i*) $\ll G \gg \oplus \text{Skip}$) *Q* \Vdash
 $\text{Sup-trans} (\text{range} (\text{iterates body } G)) \ Q$
proof(*rule Sup-trans-upper2[OF - - nQ bQ]*)
from *iterates-healthy[OF hb]*
show $\forall u \in \text{range} (\text{iterates body } G).$
 $\forall R. \text{nneg } R \wedge \text{bounded-by} (\text{bound-of } P) R \longrightarrow$
 $\text{nneg } (u R) \wedge \text{bounded-by} (\text{bound-of } P) (u R)$
by(*auto*)
have *wp* (*body* ;; *Embed* (*iterates body G i*) $\ll G \gg \oplus \text{Skip}$) = *iterates body G (Suc i)*
by(*simp*)
also have $\dots \in \text{range} (\text{iterates body } G)$
by(*blast*)
finally show *wp* (*body* ;; *Embed* (*iterates body G i*) $\ll G \gg \oplus \text{Skip}$) \in
 $\text{range} (\text{iterates body } G) .$

```

    qed
  qed
  qed
  finally show le-trans (?F ?Y) ?Y .

  fix P::'s expect
  assume sound P
  with fSup show sound (?Y P) by(auto)
  qed
  qed

```

Therefore, evaluated at a given point (state), the sequence of iterates gives a sequence of real values that converges on that of the loop itself.

corollary *loop-iterates*:

```

  fixes body::'s prog
  assumes hb: healthy (wp body)
    and cb: bd-cts (wp body)
    and sP: sound P
  shows (λi. iterates body G i P s) ⟶ wp (do G ⟶ body od) P s
  proof –
  let ?X = {f s |f. f ∈ {t P |t. t ∈ range (iterates body G)}}
  have closure-Sup: Sup ?X ∈ closure ?X
  proof(rule closure-contains-Sup, simp, clarsimp)
  fix i
  from sP have bounded-by (bound-of P) P by(auto)
  with iterates-healthy[OF hb] sP have ∧j. bounded-by (bound-of P) (iterates body G j
P)
  by(auto)
  thus iterates body G i P s ≤ bound-of P by(auto)
  qed

```

```

  have (λi. iterates body G i P s) ⟶ Sup {f s |f. f ∈ {t P |t. t ∈ range (iterates body
G)}}
  proof(rule LIMSEQ-I)
  fix r::real assume posr: 0 < r
  with closure-Sup obtain y where yin: y ∈ ?X and ey: dist y (Sup ?X) < r
  by(simp only:closure-approachable, blast)
  from yin obtain i where yit: y = iterates body G i P s by(auto)
  {
  fix j
  have i ≤ j ⟶ le-trans (iterates body G i) (iterates body G j)
  proof(induct j, simp, clarify)
  fix k
  assume IH: i ≤ k ⟶ le-trans (iterates body G i) (iterates body G k)
    and le: i ≤ Suc k
  show le-trans (iterates body G i) (iterates body G (Suc k))
  proof(cases i = Suc k, simp)
  assume i ≠ Suc k
  with le have i ≤ k by(auto)

```

```

    with IH have le-trans (iterates body G i) (iterates body G k) by(auto)
    also note iterates-increasing[OF hb]
    finally show le-trans (iterates body G i) (iterates body G (Suc k)) .
  qed
  qed
}
with sP have  $\forall j \geq i. \text{iterates body G } i P s \leq \text{iterates body G } j P s$ 
  by(auto)
moreover {
  from sP have bounded-by (bound-of P) P by(auto)
  with iterates-healthy[OF hb] sP have  $\bigwedge j. \text{bounded-by (bound-of P) (iterates body G } j$ 
P)
    by(auto)
    hence  $\bigwedge j. \text{iterates body G } j P s \leq \text{bound-of P}$  by(auto)
    hence  $\bigwedge j. \text{iterates body G } j P s \leq \text{Sup } ?X$ 
    by(intro cSup-upper bdd-aboveI, auto)
  }
ultimately have  $\bigwedge j. i \leq j \implies$ 
  norm (iterates body G j P s - Sup ?X)  $\leq$ 
  norm (iterates body G i P s - Sup ?X)
  by(auto)
also from ey yit have norm (iterates body G i P s - Sup ?X)  $< r$ 
  by(simp add:dist-real-def)
finally show  $\exists no. \forall n \geq no. \text{norm (iterates body G } n P s -$ 
  Sup {f s | f. f  $\in \{t P \mid t. t \in \text{range (iterates body G)}\}} < r$ 
  by(auto)
qed
moreover
from hb cb sP have wp do G  $\longrightarrow$  body od P s = Sup-trans (range (iterates body G)) P s
  by(simp add:equiv-transD[OF lfp-iterates])
moreover have ... = Sup {f s | f. f  $\in \{t P \mid t. t \in \text{range (iterates body G)}\}}$ 
  by(simp add:Sup-trans-def Sup-exp-def)
ultimately show ?thesis by(simp)
qed

```

The iterates themselves are all continuous.

```

lemma cts-iterates:
  fixes body::'s prog
  assumes hb: healthy (wp body)
    and cb: bd-cts (wp body)
  shows bd-cts (iterates body G i)
proof(induct i, simp-all)
  have range ( $\lambda(n::nat) (s::'s). 0::real$ ) = { $\lambda s. 0::real$ }
  by(auto)
  thus bd-cts ( $\lambda P (s::'s). 0$ )
  by(intro bd-ctsI, simp add:o-def Sup-exp-def)
next
fix i
assume IH: bd-cts (iterates body G i)

```



```

thus bd-cts (wp (body ;; Embed (iterates body G i) « G » ⊕ Skip))
  by(blast intro:cts-wp-PC cts-wp-Seq cts-wp-Embed cts-wp-Skip
    healthy-intros iterates-healthy cb hb)
qed

```

Therefore so is the loop itself.

lemma *cts-wp-loop*:

```

fixes body::'s prog
assumes hb: healthy (wp body)
  and cb: bd-cts (wp body)
shows bd-cts (wp do G → body od)
proof(rule bd-ctsI)
fix M::nat ⇒ 's expect and b::real
assume chain: ∧i. M i ⊢ M (Suc i)
  and sM: ∧i. sound (M i)
  and bM: ∧i. bounded-by b (M i)

```

```

from sM bM iterates-healthy[OF hb]
have ∧j i. bounded-by b (iterates body G i (M j)) by(blast)
hence iB: ∧j i s. iterates body G i (M j) s ≤ b by(auto)

```

```

from sM bM have sSup: sound (Sup-exp (range M))
  by(auto intro:Sup-exp-sound)
with lfp-iterates[OF hb cb]
have wp do G → body od (Sup-exp (range M)) =
  Sup-trans (range (iterates body G)) (Sup-exp (range M))
  by(simp add:equiv-transD)
also {
  from chain sM bM
  have ∧i. iterates body G i (Sup-exp (range M)) = Sup-exp (range (iterates body G i o
M))
  by(blast intro:bd-ctsD cts-iterates[OF hb cb])
  hence {t (Sup-exp (range M)) | t. t ∈ range (iterates body G)} =
    {Sup-exp (range (t o M)) | t. t ∈ range (iterates body G)}
  by(auto intro:sym)
  hence Sup-trans (range (iterates body G)) (Sup-exp (range M)) =
    Sup-exp {Sup-exp (range (t o M)) | t. t ∈ range (iterates body G)}
  by(simp add:Sup-trans-def)
}
also {
have ∧s. {f s |f. ∃t. f = (λs. Sup {f s |f. f ∈ range (t o M)} )} ∧
  t ∈ range (iterates body G)} =
  range (λi. Sup (range (λj. iterates body G i (M j) s)))
  (is ∧s. ?X s = ?Y s)
proof(intro antisym subsetI)
fix s x
assume x ∈ ?X s
then obtain t where rx: x = Sup {f s |f. f ∈ range (t o M)}
  and t ∈ range (iterates body G) by(auto)

```

then obtain i where $t = \text{iterates body } G \ i$ by(auto)
with $rw x$ have $x = \text{Sup } \{f \ s \mid f. f \in \text{range } (\lambda j. \text{iterates body } G \ i \ (M \ j))\}$
by(simp add:o-def)
moreover have $\{f \ s \mid f. f \in \text{range } (\lambda j. \text{iterates body } G \ i \ (M \ j))\} =$
 $\text{range } (\lambda j. \text{iterates body } G \ i \ (M \ j) \ s)$ **by(auto)**
ultimately have $x = \text{Sup } (\text{range } (\lambda j. \text{iterates body } G \ i \ (M \ j) \ s))$
by(simp)
thus $x \in \text{range } (\lambda i. \text{Sup } (\text{range } (\lambda j. \text{iterates body } G \ i \ (M \ j) \ s)))$
by(auto)
next
fix $s \ x$
assume $x \in ?Y \ s$
then obtain i where $A: x = \text{Sup } (\text{range } (\lambda j. \text{iterates body } G \ i \ (M \ j) \ s))$
by(auto)

have $\bigwedge s. \{f \ s \mid f. f \in \text{range } (\lambda j. \text{iterates body } G \ i \ (M \ j))\} =$
 $\text{range } (\lambda j. \text{iterates body } G \ i \ (M \ j) \ s)$ **by(auto)**
hence $B: (\lambda s. \text{Sup } (\text{range } (\lambda j. \text{iterates body } G \ i \ (M \ j) \ s))) =$
 $(\lambda s. \text{Sup } \{f \ s \mid f. f \in \text{range } (\text{iterates body } G \ i \ o \ M)\})$
by(simp add:o-def)

have $C: \text{iterates body } G \ i \in \text{range } (\text{iterates body } G)$ by(auto)

have $\exists f. x = f \ s \wedge$
 $(\exists t. f = (\lambda s. \text{Sup } \{f \ s \mid f. f \in \text{range } (t \ o \ M)\}) \wedge$
 $t \in \text{range } (\text{iterates body } G))$
by(iprover intro:A B C)
thus $x \in ?X \ s$ by(simp)
qed
hence $\text{Sup-exp } \{\text{Sup-exp } (\text{range } (t \ o \ M)) \mid t. t \in \text{range } (\text{iterates body } G)\} =$
 $(\lambda s. \text{Sup } (\text{range } (\lambda i. \text{Sup } (\text{range } (\lambda j. \text{iterates body } G \ i \ (M \ j) \ s))))$
by(simp add:Sup-exp-def)
}
also have $(\lambda s. \text{Sup } (\text{range } (\lambda i. \text{Sup } (\text{range } (\lambda j. \text{iterates body } G \ i \ (M \ j) \ s)))) =$
 $(\lambda s. \text{Sup } (\text{range } (\lambda (i,j). \text{iterates body } G \ i \ (M \ j) \ s)))$
(is ?X = ?Y)
proof(rule ext, rule antisym)
fix $s::'s$
show $?Y \ s \leq ?X \ s$
proof(rule cSup-least, blast, clarify)
fix $i \ j::\text{nat}$
from iB have $\text{iterates body } G \ i \ (M \ j) \ s \leq \text{Sup } (\text{range } (\lambda j. \text{iterates body } G \ i \ (M \ j) \ s))$
by(intro cSup-upper bdd-aboveI, auto)
also from iB have $\dots \leq \text{Sup } (\text{range } (\lambda i. \text{Sup } (\text{range } (\lambda j. \text{iterates body } G \ i \ (M \ j) \ s))))$
by(intro cSup-upper cSup-least bdd-aboveI, (blast intro:cSup-least)+)
finally show $\text{iterates body } G \ i \ (M \ j) \ s \leq$
 $\text{Sup } (\text{range } (\lambda i. \text{Sup } (\text{range } (\lambda j. \text{iterates body } G \ i \ (M \ j) \ s)))) .$
qed
have $\bigwedge i \ j. \text{iterates body } G \ i \ (M \ j) \ s \leq$

```

      Sup (range (λ(i, j). iterates body G i (M j) s))
    by(rule cSup-upper, auto intro:iB)
  thus ?X s ≤ ?Y s
  by(intro cSup-least, blast, clarify, simp, blast intro:cSup-least)
qed
also have ... = (λs. Sup (range (λj. Sup (range (λi. iterates body G i (M j) s))))))
  (is ?X = ?Y)
proof(rule ext, rule antisym)
  fix s::'s
  have ∧i j. iterates body G i (M j) s ≤
    Sup (range (λ(i, j). iterates body G i (M j) s))
  by(rule cSup-upper, auto intro:iB)
  thus ?Y s ≤ ?X s
  by(intro cSup-least, blast, clarify, simp, blast intro:cSup-least)
  show ?X s ≤ ?Y s
  proof(rule cSup-least, blast, clarify)
    fix i j::nat
    from iB have iterates body G i (M j) s ≤ Sup (range (λi. iterates body G i (M j) s))
    by(intro cSup-upper bdd-aboveI, auto)
    also from iB have ... ≤ Sup (range (λj. Sup (range (λi. iterates body G i (M j) s))))
    by(intro cSup-upper cSup-least bdd-aboveI, blast, blast intro:cSup-least)
    finally show iterates body G i (M j) s ≤
      Sup (range (λj. Sup (range (λi. iterates body G i (M j) s)))) .
  qed
qed
also {
  have ∧s. range (λj. Sup (range (λi. iterates body G i (M j) s))) =
    {f s |f. f ∈ range ((λP s. Sup {f s |f. ∃t. f = t P ∧
      t ∈ range (iterates body G)})) ∘ M} (is ∧s. ?X s = ?Y s)
  proof(intro antisym subsetI)
    fix s x
    assume x ∈ ?X s
    then obtain j where rwx: x = Sup (range (λi. iterates body G i (M j) s)) by(auto)
    moreover {
      have ∧s. range (λi. iterates body G i (M j) s) =
        {f s |f. ∃t. f = t (M j) ∧ t ∈ range (iterates body G)}
      by(auto)
      hence (λs. Sup (range (λi. iterates body G i (M j) s))) ∈
        range ((λP s. Sup {f s |f.
          ∃t. f = t P ∧ t ∈ range (iterates body G)})) ∘ M
      by (simp add: o-def cong del: SUP-cong-simp)
    }
    ultimately show x ∈ ?Y s by(auto)
  next
  fix s x
  assume x ∈ ?Y s
  then obtain P where rwx: x = P s
    and Pin: P ∈ range ((λP s. Sup {f s |f.
      ∃t. f = t P ∧ t ∈ range (iterates body G)})) ∘ M

```

```

by(auto)
then obtain  $j$  where  $P = (\lambda s. \text{Sup } \{f s \mid f. \exists t. f = t (M j) \wedge$ 
                                      $t \in \text{range } (\text{iterates body } G)\})$ 

by(auto)
also {
have  $\bigwedge s. \{f s \mid f. \exists t. f = t (M j) \wedge t \in \text{range } (\text{iterates body } G)\} =$ 
       $\text{range } (\lambda i. \text{iterates body } G i (M j) s)$  by(auto)
hence  $(\lambda s. \text{Sup } \{f s \mid f. \exists t. f = t (M j) \wedge t \in \text{range } (\text{iterates body } G)\}) =$ 
       $(\lambda s. \text{Sup } (\text{range } (\lambda i. \text{iterates body } G i (M j) s)))$ 
by(simp)
}
finally have  $x = \text{Sup } (\text{range } (\lambda i. \text{iterates body } G i (M j) s))$ 
by(simp add: rwx)
thus  $x \in ?X s$  by(simp)
qed
hence  $(\lambda s. \text{Sup } (\text{range } (\lambda j. \text{Sup } (\text{range } (\lambda i. \text{iterates body } G i (M j) s)))) =$ 
       $\text{Sup-exp } (\text{range } (\text{Sup-trans } (\text{range } (\text{iterates body } G)) o M))$ 
by (simp add: Sup-exp-def Sup-trans-def cong del: SUP-cong-simp)
}
also have  $\text{Sup-exp } (\text{range } (\text{Sup-trans } (\text{range } (\text{iterates body } G)) o M)) =$ 
       $\text{Sup-exp } (\text{range } (\text{wp do } G \longrightarrow \text{body od } o M))$ 
by(simp add: o-def equiv-transD[OF lfp-iterates, OF hb cb, OF sM])
finally show  $\text{wp do } G \longrightarrow \text{body od } (\text{Sup-exp } (\text{range } M)) =$ 
       $\text{Sup-exp } (\text{range } (\text{wp do } G \longrightarrow \text{body od } o M))$  .
qed

lemmas cts-intros =
  cts-wp-Abort cts-wp-Skip
  cts-wp-Seq cts-wp-PC
  cts-wp-DC cts-wp-Embed
  cts-wp-Apply cts-wp-SetDC
  cts-wp-SetPC cts-wp-Bind
  cts-wp-repeat

end

```

4.5 Sublinearity

theory *Sublinearity* **imports** *Embedding Healthiness LoopInduction* **begin**

4.5.1 Nonrecursive Primitives

Sublinearity of non-recursive programs is generally straightforward, and follows from the algebraic properties of the underlying operations, together with healthiness.

lemma *sublinear-wp-Skip*:
sublinear (*wp Skip*)
by(*auto simp:wp-eval*)

lemma *sublinear-wp-Abort*:

sublinear (wp Abort)

by(*auto simp:wp-eval*)

lemma *sublinear-wp-Apply*:

sublinear (wp (Apply f))

by(*auto simp:wp-eval*)

lemma *sublinear-wp-Seq*:

fixes $x::'s \text{ prog}$

assumes $slx: \text{sublinear } (wp \ x)$ **and** $sly: \text{sublinear } (wp \ y)$

and $hx: \text{healthy } (wp \ x)$ **and** $hy: \text{healthy } (wp \ y)$

shows *sublinear (wp (x ;; y))*

proof(*rule sublinearI, simp add:wp-eval*)

fix $P::'s \Rightarrow \text{real}$ **and** $Q::'s \Rightarrow \text{real}$ **and** $s::'s$

and $a::\text{real}$ **and** $b::\text{real}$ **and** $c::\text{real}$

assume $sP: \text{sound } P$ **and** $sQ: \text{sound } Q$

and $nna: 0 \leq a$ **and** $n nb: 0 \leq b$ **and** $nnc: 0 \leq c$

with $slx \ hy$ **have** $a * wp \ x \ (wp \ y \ P) \ s + b * wp \ x \ (wp \ y \ Q) \ s \ominus c \leq$

$wp \ x \ (\lambda s. a * wp \ y \ P \ s + b * wp \ y \ Q \ s \ominus c) \ s$

by(*blast intro:sublinearD*)

also {

from $sP \ sQ \ nna \ n nb \ nnc \ sly$

have $\bigwedge s. a * wp \ y \ P \ s + b * wp \ y \ Q \ s \ominus c \leq$

$wp \ y \ (\lambda s. a * P \ s + b * Q \ s \ominus c) \ s$

by(*blast intro:sublinearD*)

moreover from $sP \ sQ \ hy$

have *sound (wp y P)* **and** *sound (wp y Q)* **by**(*auto*)

moreover with $nna \ n nb \ nnc$

have *sound* $(\lambda s. a * wp \ y \ P \ s + b * wp \ y \ Q \ s \ominus c)$

by(*auto intro!:sound-intros tminus-sound*)

moreover from $sP \ sQ \ nna \ n nb \ nnc$

have *sound* $(\lambda s. a * P \ s + b * Q \ s \ominus c)$

by(*auto intro!:sound-intros tminus-sound*)

moreover with hy **have** *sound* $(wp \ y \ (\lambda s. a * P \ s + b * Q \ s \ominus c))$

by(*blast*)

ultimately

have $wp \ x \ (\lambda s. a * wp \ y \ P \ s + b * wp \ y \ Q \ s \ominus c) \ s \leq$

$wp \ x \ (wp \ y \ (\lambda s. a * P \ s + b * Q \ s \ominus c)) \ s$

by(*blast intro!:le-funD[OF mono-transD[OF healthy-monoD[OF hx]]]*)

}

finally show $a * wp \ x \ (wp \ y \ P) \ s + b * wp \ x \ (wp \ y \ Q) \ s \ominus c \leq$

$wp \ x \ (wp \ y \ (\lambda s. a * P \ s + b * Q \ s \ominus c)) \ s .$

qed

lemma *sublinear-wp-PC*:

fixes $x::'s \text{ prog}$

assumes $slx: \text{sublinear } (wp \ x)$ **and** $sly: \text{sublinear } (wp \ y)$

and uP : unitary P
shows *sublinear* ($wp\ (x\ p \oplus\ y)$)
proof(*rule sublinearI, simp add:wp-eval*)
fix $R::'s \Rightarrow real$ **and** $Q::'s \Rightarrow real$ **and** $s::'s$
and $a::real$ **and** $b::real$ **and** $c::real$
assume sR : *sound* R **and** sQ : *sound* Q
and nna : $0 \leq a$ **and** nnb : $0 \leq b$ **and** nnc : $0 \leq c$

have $a * (P\ s * wp\ x\ Q\ s + (I - P\ s) * wp\ y\ Q\ s) +$
 $b * (P\ s * wp\ x\ R\ s + (I - P\ s) * wp\ y\ R\ s) \ominus c =$
 $(P\ s * a * wp\ x\ Q\ s + (I - P\ s) * a * wp\ y\ Q\ s) +$
 $(P\ s * b * wp\ x\ R\ s + (I - P\ s) * b * wp\ y\ R\ s) \ominus c$
by(*simp add:field-simps*)

also
have $\dots = (P\ s * a * wp\ x\ Q\ s + P\ s * b * wp\ x\ R\ s) +$
 $((I - P\ s) * a * wp\ y\ Q\ s + (I - P\ s) * b * wp\ y\ R\ s) \ominus c$
by(*simp add:ac-simps*)

also
have $\dots = P\ s * (a * wp\ x\ Q\ s + b * wp\ x\ R\ s) +$
 $(I - P\ s) * (a * wp\ y\ Q\ s + b * wp\ y\ R\ s) \ominus$
 $(P\ s * c + (I - P\ s) * c)$
by(*simp add:field-simps*)

also
have $\dots \leq (P\ s * (a * wp\ x\ Q\ s + b * wp\ x\ R\ s) \ominus P\ s * c) +$
 $((I - P\ s) * (a * wp\ y\ Q\ s + b * wp\ y\ R\ s) \ominus (I - P\ s) * c)$
by(*rule tminus-add-mono*)

also {
from uP **have** $0 \leq P\ s$ **and** $0 \leq I - P\ s$
by *auto*
hence $(P\ s * (a * wp\ x\ Q\ s + b * wp\ x\ R\ s) \ominus P\ s * c) +$
 $((I - P\ s) * (a * wp\ y\ Q\ s + b * wp\ y\ R\ s) \ominus (I - P\ s) * c) =$
 $P\ s * (a * wp\ x\ Q\ s + b * wp\ x\ R\ s \ominus c) +$
 $(I - P\ s) * (a * wp\ y\ Q\ s + b * wp\ y\ R\ s \ominus c)$
by(*simp add:tminus-left-distrib*)
}

also {
from $sQ\ sR\ nna\ nnb\ nnc\ slx$
have $a * wp\ x\ Q\ s + b * wp\ x\ R\ s \ominus c \leq$
 $wp\ x\ (\lambda s. a * Q\ s + b * R\ s \ominus c)\ s$
by(*blast*)
moreover
from $sQ\ sR\ nna\ nnb\ nnc\ sly$
have $a * wp\ y\ Q\ s + b * wp\ y\ R\ s \ominus c \leq$
 $wp\ y\ (\lambda s. a * Q\ s + b * R\ s \ominus c)\ s$
by(*blast*)
moreover
from uP **have** $0 \leq P\ s$ **and** $0 \leq I - P\ s$
by *auto*
ultimately

```

have  $P s * (a * wp\ x\ Q\ s + b * wp\ x\ R\ s \ominus c) +$ 
       $(I - P s) * (a * wp\ y\ Q\ s + b * wp\ y\ R\ s \ominus c) \leq$ 
       $P s * wp\ x\ (\lambda s. a * Q\ s + b * R\ s \ominus c)\ s +$ 
       $(I - P s) * wp\ y\ (\lambda s. a * Q\ s + b * R\ s \ominus c)\ s$ 
by(blast intro:add-mono mult-left-mono)
}
finally
show  $a * (P s * wp\ x\ Q\ s + (I - P s) * wp\ y\ Q\ s) +$ 
       $b * (P s * wp\ x\ R\ s + (I - P s) * wp\ y\ R\ s) \ominus c \leq$ 
       $P s * wp\ x\ (\lambda s. a * Q\ s + b * R\ s \ominus c)\ s +$ 
       $(I - P s) * wp\ y\ (\lambda s. a * Q\ s + b * R\ s \ominus c)\ s .$ 
qed

```

lemma *sublinear-wp-DC*:

```

fixes  $x::'s\ prog$ 
assumes  $slx: sublinear\ (wp\ x)$  and  $sly: sublinear\ (wp\ y)$ 
shows  $sublinear\ (wp\ (x \sqcap y))$ 
proof(rule sublinearI, simp only:wp-eval)
fix  $R::'s \Rightarrow real$  and  $Q::'s \Rightarrow real$  and  $s::'s$ 
and  $a::real$  and  $b::real$  and  $c::real$ 
assume  $sR: sound\ R$  and  $sQ: sound\ Q$ 
and  $nna: 0 \leq a$  and  $nmb: 0 \leq b$  and  $nnc: 0 \leq c$ 

```

from $nna\ nmb$

```

have  $a * \min\ (wp\ x\ Q\ s)\ (wp\ y\ Q\ s) +$ 
       $b * \min\ (wp\ x\ R\ s)\ (wp\ y\ R\ s) \ominus c =$ 
       $\min\ (a * wp\ x\ Q\ s)\ (a * wp\ y\ Q\ s) +$ 
       $\min\ (b * wp\ x\ R\ s)\ (b * wp\ y\ R\ s) \ominus c$ 
by(simp add:min-distrib)

```

also

```

have  $\dots \leq \min\ (a * wp\ x\ Q\ s + b * wp\ x\ R\ s)$ 
       $(a * wp\ y\ Q\ s + b * wp\ y\ R\ s) \ominus c$ 
by(auto intro!:tminus-left-mono)

```

also

```

have  $\dots = \min\ (a * wp\ x\ Q\ s + b * wp\ x\ R\ s \ominus c)$ 
       $(a * wp\ y\ Q\ s + b * wp\ y\ R\ s \ominus c)$ 
by(rule min-tminus-distrib)

```

also {

from $slx\ sQ\ sR\ nna\ nmb\ nnc$

```

have  $a * wp\ x\ Q\ s + b * wp\ x\ R\ s \ominus c \leq$ 
       $wp\ x\ (\lambda s. a * Q\ s + b * R\ s \ominus c)\ s$ 
by(blast)

```

moreover

from $sly\ sQ\ sR\ nna\ nmb\ nnc$

```

have  $a * wp\ y\ Q\ s + b * wp\ y\ R\ s \ominus c \leq$ 
       $wp\ y\ (\lambda s. a * Q\ s + b * R\ s \ominus c)\ s$ 
by(blast)

```

ultimately

```

have  $\min\ (a * wp\ x\ Q\ s + b * wp\ x\ R\ s \ominus c)$ 

```

$$\begin{aligned}
& (a * wp\ y\ Q\ s + b * wp\ y\ R\ s \ominus c) \leq \\
& \min (wp\ x\ (\lambda s. a * Q\ s + b * R\ s \ominus c)\ s) \\
& (wp\ y\ (\lambda s. a * Q\ s + b * R\ s \ominus c)\ s) \\
& \text{by}(\text{auto}) \\
& \} \\
& \text{finally show } a * \min (wp\ x\ Q\ s)\ (wp\ y\ Q\ s) + \\
& \quad b * \min (wp\ x\ R\ s)\ (wp\ y\ R\ s) \ominus c \leq \\
& \quad \min (wp\ x\ (\lambda s. a * Q\ s + b * R\ s \ominus c)\ s) \\
& \quad (wp\ y\ (\lambda s. a * Q\ s + b * R\ s \ominus c)\ s). \\
& \text{qed}
\end{aligned}$$

As for continuity, we insist on a finite support.

lemma *sublinear-wp-SetPC*:

fixes $p::'a \Rightarrow 's\ prog$

assumes $slp: \bigwedge s\ a. a \in \text{supp}\ (P\ s) \implies \text{sublinear}\ (wp\ (p\ a))$

and $sum: \bigwedge s. (\sum a \in \text{supp}\ (P\ s). P\ s\ a) \leq 1$

and $nnP: \bigwedge s\ a. 0 \leq P\ s\ a$

and $fin: \bigwedge s. \text{finite}\ (\text{supp}\ (P\ s))$

shows $\text{sublinear}\ (wp\ (\text{SetPC}\ p\ P))$

proof(*rule sublinearI, simp add:wp-eval*)

fix $R::'s \Rightarrow \text{real}$ **and** $Q::'s \Rightarrow \text{real}$ **and** $s::'s$

and $a::\text{real}$ **and** $b::\text{real}$ **and** $c::\text{real}$

assume $sR: \text{sound}\ R$ **and** $sQ: \text{sound}\ Q$

and $nna: 0 \leq a$ **and** $nnb: 0 \leq b$ **and** $nnc: 0 \leq c$

have $a * (\sum a' \in \text{supp}\ (P\ s). P\ s\ a' * wp\ (p\ a')\ Q\ s) +$
 $b * (\sum a' \in \text{supp}\ (P\ s). P\ s\ a' * wp\ (p\ a')\ R\ s) \ominus c =$
 $(\sum a' \in \text{supp}\ (P\ s). P\ s\ a' * (a * wp\ (p\ a')\ Q\ s + b * wp\ (p\ a')\ R\ s)) \ominus c$

by(*simp add:field-simps sum-distrib-left sum.distrib*)

also have $\dots \leq$

$$\begin{aligned}
& (\sum a' \in \text{supp}\ (P\ s). P\ s\ a' * (a * wp\ (p\ a')\ Q\ s + b * wp\ (p\ a')\ R\ s)) \ominus \\
& (\sum a' \in \text{supp}\ (P\ s). P\ s\ a' * c)
\end{aligned}$$

proof(*rule tminus-right-antimono*)

have $(\sum a' \in \text{supp}\ (P\ s). P\ s\ a' * c) \leq (\sum a' \in \text{supp}\ (P\ s). P\ s\ a') * c$

by(*simp add:sum-distrib-right*)

also from *sum* **and** *nnc* **have** $\dots \leq 1 * c$

by(*rule mult-right-mono*)

finally show $(\sum a' \in \text{supp}\ (P\ s). P\ s\ a' * c) \leq c$ **by**(*simp*)

qed

also from *fin*

have $\dots \leq (\sum a' \in \text{supp}\ (P\ s). P\ s\ a' * (a * wp\ (p\ a')\ Q\ s + b * wp\ (p\ a')\ R\ s) \ominus P\ s\ a' * c)$

by(*blast intro:tminus-sum-mono*)

also have $\dots = (\sum a' \in \text{supp}\ (P\ s). P\ s\ a' * (a * wp\ (p\ a')\ Q\ s + b * wp\ (p\ a')\ R\ s \ominus c))$

by(*simp add:nnP tminus-left-distrib*)

also {

from *slp sQ sR nna nnb nnc*

have $\bigwedge a'. a' \in \text{supp}\ (P\ s) \implies a * wp\ (p\ a')\ Q\ s + b * wp\ (p\ a')\ R\ s \ominus c \leq$
 $wp\ (p\ a')\ (\lambda s. a * Q\ s + b * R\ s \ominus c)\ s$

by(*blast*)

with nnP
have $(\sum a' \in \text{supp } (P s). P s a' * (a * \text{wp } (p a') Q s + b * \text{wp } (p a') R s \ominus c)) \leq$
 $(\sum a' \in \text{supp } (P s). P s a' * \text{wp } (p a') (\lambda s. a * Q s + b * R s \ominus c) s)$
by (*blast intro:sum-mono mult-left-mono*)
}
finally
show $a * (\sum a' \in \text{supp } (P s). P s a' * \text{wp } (p a') Q s) +$
 $b * (\sum a' \in \text{supp } (P s). P s a' * \text{wp } (p a') R s) \ominus c \leq$
 $(\sum a' \in \text{supp } (P s). P s a' * \text{wp } (p a') (\lambda s. a * Q s + b * R s \ominus c) s)$.
qed

lemma *sublinear-wp-SetDC*:

fixes $p::'a \Rightarrow 's \text{ prog}$
assumes $slp: \bigwedge s a. a \in S s \implies \text{sublinear } (\text{wp } (p a))$
and $hp: \bigwedge s a. a \in S s \implies \text{healthy } (\text{wp } (p a))$
and $ne: \bigwedge s. S s \neq \{\}$
shows $\text{sublinear } (\text{wp } (\text{SetDC } p S))$
proof (*rule sublinearI, simp add:wp-eval, rule cInf-greatest*)
fix $P::'s \Rightarrow \text{real}$ **and** $Q::'s \Rightarrow \text{real}$ **and** $s::'s$ **and** $x y$
and $a::\text{real}$ **and** $b::\text{real}$ **and** $c::\text{real}$
assume $sP: \text{sound } P$ **and** $sQ: \text{sound } Q$
and $nna: 0 \leq a$ **and** $nnb: 0 \leq b$ **and** $nnc: 0 \leq c$
from ne **show** $(\lambda pr. \text{wp } (p pr) (\lambda s. a * P s + b * Q s \ominus c) s) 'S s \neq \{\}$ **by** (*auto*)
assume $yin: y \in (\lambda pr. \text{wp } (p pr) (\lambda s. a * P s + b * Q s \ominus c) s) 'S s$
then obtain x **where** $xin: x \in S s$ **and** $rwy: y = \text{wp } (p x) (\lambda s. a * P s + b * Q s \ominus c) s$
by (*auto*)
from xin hp sP nna
have $a * \text{Inf } ((\lambda a. \text{wp } (p a) P s) 'S s) \leq a * \text{wp } (p x) P s$
by (*intro mult-left-mono[OF cInf-lower] bdd-belowI[where m=0], blast+*)
moreover from xin hp sQ nnb
have $b * \text{Inf } ((\lambda a. \text{wp } (p a) Q s) 'S s) \leq b * \text{wp } (p x) Q s$
by (*intro mult-left-mono[OF cInf-lower] bdd-belowI[where m=0], blast+*)
ultimately
have $a * \text{Inf } ((\lambda a. \text{wp } (p a) P s) 'S s) +$
 $b * \text{Inf } ((\lambda a. \text{wp } (p a) Q s) 'S s) \ominus c \leq$
 $a * \text{wp } (p x) P s + b * \text{wp } (p x) Q s \ominus c$
by (*blast intro:tminus-left-mono add-mono*)
also from xin slp sP sQ nna nnb nnc
have $\dots \leq \text{wp } (p x) (\lambda s. a * P s + b * Q s \ominus c) s$
by (*blast*)
finally show $a * \text{Inf } ((\lambda a. \text{wp } (p a) P s) 'S s) + b * \text{Inf } ((\lambda a. \text{wp } (p a) Q s) 'S s) \ominus c \leq$
 y
by (*simp add:rwy*)
qed

lemma *sublinear-wp-Embed*:
 $\text{sublinear } t \implies \text{sublinear } (\text{wp } (\text{Embed } t))$
by(*simp add:wp-eval*)

lemma *sublinear-wp-repeat*:
 $\llbracket \text{sublinear } (\text{wp } p); \text{healthy } (\text{wp } p) \rrbracket \implies \text{sublinear } (\text{wp } (\text{repeat } n p))$
by(*induct n, simp-all add:sublinear-wp-Seq sublinear-wp-Skip healthy-wp-repeat*)

lemma *sublinear-wp-Bind*:
 $\llbracket \bigwedge s. \text{sublinear } (\text{wp } (a (f s))) \rrbracket \implies \text{sublinear } (\text{wp } (\text{Bind } f a))$
by(*rule sublinearI, simp add:wp-eval, auto*)

4.5.2 Sublinearity for Loops

We break the proof of sublinearity loops into separate proofs of sub-distributivity and sub-additivity. The first follows by transfinite induction.

lemma *sub-distrib-wp-loop*:
fixes *body::'s prog*
assumes *sdb: sub-distrib (wp body)*
and *hb: healthy (wp body)*
and *nhb: nearly-healthy (wlp body)*
shows *sub-distrib (wp (do G \longrightarrow body od))*
proof –
have $\forall P s. \text{sound } P \longrightarrow \text{wp } (\text{do } G \longrightarrow \text{body od}) P s \oplus I \leq$
 $\text{wp } (\text{do } G \longrightarrow \text{body od}) (\lambda s. P s \oplus I) s$
proof(*rule loop-induct[OF hb nhb], safe*)
fix *S::('s trans \times 's trans) set* **and** *P::'s expect* **and** *s::'s*
assume *saS: $\forall x \in S. \forall P s. \text{sound } P \longrightarrow \text{fst } x P s \oplus I \leq \text{fst } x (\lambda s. P s \oplus I) s$*
and *sP: sound P*
and *fS: $\forall x \in S. \text{feasible } (\text{fst } x)$*

from *sP* **have** *sPm: sound $(\lambda s. P s \oplus I)$* **by**(*auto intro:tminus-sound*)

have *nnSup: $\bigwedge s. 0 \leq \text{Sup-trans } (\text{fst } ' S) (\lambda s. P s \oplus I) s$*
proof(*cases S={}, simp add:Sup-trans-def Sup-exp-def*)
fix *s*
assume *S \neq {}*
then obtain *x* **where** *xin: $x \in S$* **by**(*auto*)
with *fS sPm* **have** $0 \leq \text{fst } x (\lambda s. P s \oplus I) s$ **by**(*auto*)
also from *xin fS sPm* **have** $\dots \leq \text{Sup-trans } (\text{fst } ' S) (\lambda s. P s \oplus I) s$
by(*auto intro!: le-funD[OF Sup-trans-upper2]*)
finally show *?thesis s* .
qed

have $\bigwedge x s. \text{fst } x P s \leq (\text{fst } x P s \oplus I) + 1$ **by**(*simp add:tminus-def*)
also from *saS sP*
have $\bigwedge x s. x \in S \implies (\text{fst } x P s \oplus I) + 1 \leq \text{fst } x (\lambda s. P s \oplus I) s + 1$
by(*auto intro:add-right-mono*)

```

also {
  from sP have sound ( $\lambda s. P s \ominus I$ ) by(auto intro:tminus-sound)
  with fS have  $\bigwedge x s. x \in S \implies fst\ x (\lambda s. P s \ominus I) s + I \leq$ 
    Sup-trans (fst ' S) ( $\lambda s. P s \ominus I$ ) s + I
    by(blast intro!: add-right-mono le-funD[OF Sup-trans-upper2])
}
finally have le:  $\bigwedge s. \forall x \in S. fst\ x P s \leq Sup-trans (fst ' S) (\lambda s. P s \ominus I) s + I$ 
  by(auto)
moreover from nnSup have nn:  $\bigwedge s. 0 \leq Sup-trans (fst ' S) (\lambda s. P s \ominus I) s + I$ 
  by(auto intro:add-nonneg-nonneg)
ultimately
have leSup: Sup-trans (fst ' S) P s  $\leq Sup-trans (fst ' S) (\lambda s. P s \ominus I) s + I$ 
  unfolding Sup-trans-def
  by(intro le-funD[OF Sup-exp-least], auto)

show Sup-trans (fst ' S) P s  $\ominus I \leq Sup-trans (fst ' S) (\lambda s. P s \ominus I) s$ 
proof(cases Sup-trans (fst ' S) P s  $\leq I$ , simp-all add:nnSup)
  from leSup have Sup-trans (fst ' S) P s  $- I \leq$ 
    Sup-trans (fst ' S) ( $\lambda s. P s \ominus I$ ) s + I - I
    by(auto)
  thus Sup-trans (fst ' S) P s  $- I \leq Sup-trans (fst ' S) (\lambda s. P s \ominus I) s$  by(simp)
qed
next
fix t::'s trans and P::'s expect and s::'s
assume IH:  $\forall P s. sound\ P \longrightarrow t\ P s \ominus I \leq t (\lambda a. P a \ominus I) s$ 
  and ft: feasible t
  and sP: sound P

from sP have sound ( $\lambda s. P s \ominus I$ ) by(auto intro:tminus-sound)
with ft have s2: sound (t ( $\lambda s. P s \ominus I$ )) by(auto)
from sP ft have sound (t P) by(auto)
hence s3: sound ( $\lambda s. t\ P s \ominus I$ ) by(auto intro!:tminus-sound)

show wp (body ;; Embed t «G» $\oplus$ Skip) P s  $\ominus I \leq$ 
  wp (body ;; Embed t «G» $\oplus$ Skip) ( $\lambda a. P a \ominus I$ ) s
proof(simp add:wp-eval)
  have «G» s * wp body (t P) s + (I - «G» s) * P s  $\ominus I =$ 
    «G» s * wp body (t P) s + (I - «G» s) * P s  $\ominus$  («G» s + (I - «G» s))
    by(simp)
  also have ...  $\leq$  («G» s * wp body (t P) s  $\ominus$  «G» s) +
    ((I - «G» s) * P s  $\ominus$  (I - «G» s))
    by(rule tminus-add-mono)
  also have ... = «G» s * (wp body (t P) s  $\ominus I$ ) + (I - «G» s) * (P s  $\ominus I$ )
    by(simp add:tminus-left-distrib)
  also {
    from ft sP have wp body (t P) s  $\ominus I \leq wp body (\lambda s. t\ P s \ominus I) s$ 
      by(auto intro:sub-distrib[OF sdb])
    also {
      from IH sP have  $\lambda s. t\ P s \ominus I \Vdash t (\lambda s. P s \ominus I)$  by(auto)
    }
  }

```

```

with  $sP$  ft  $s2$   $s3$  have  $wp$   $body$   $(\lambda s. t P s \ominus I) s \leq wp$   $body$   $(t (\lambda s. P s \ominus I)) s$ 
by( $blast$   $intro:le-funD[OF$   $mono-transD$ ,  $OF$   $healthy-monoD$ ,  $OF$   $hb$ ])
}
finally have  $\langle G \rangle s * (wp$   $body$   $(t P) s \ominus I) + (I - \langle G \rangle s) * (P s \ominus I) \leq$ 
 $\langle G \rangle s * wp$   $body$   $(t (\lambda s. P s \ominus I)) s + (I - \langle G \rangle s) * (P s \ominus I)$ 
by( $auto$   $intro:add-right-mono$   $mult-left-mono$ )
}
finally show  $\langle G \rangle s * wp$   $body$   $(t P) s + (I - \langle G \rangle s) * P s \ominus I \leq$ 
 $\langle G \rangle s * wp$   $body$   $(t (\lambda s. P s \ominus I)) s + (I - \langle G \rangle s) * (P s \ominus I)$  .
qed
next
fix  $t t'::'s$  trans and  $P::'s$  expect and  $s::'s$ 
assume  $IH: \forall P s. sound P \longrightarrow t P s \ominus I \leq t (\lambda a. P a \ominus I) s$ 
and  $eq: equiv-trans$   $t t'$  and  $sP: sound P$ 

from  $sP$  have  $t' P s \ominus I = t P s \ominus I$  by( $simp$   $add:equiv-transD[OF$   $eq]$ )
also from  $sP$   $IH$  have  $\dots \leq t (\lambda s. P s \ominus I) s$  by( $auto$ )
also {
from  $sP$  have  $sound (\lambda s. P s \ominus I)$  by( $simp$   $add:tminus-sound$ )
hence  $t (\lambda s. P s \ominus I) s = t' (\lambda s. P s \ominus I) s$  by( $simp$   $add:equiv-transD[OF$   $eq]$ )
}
finally show  $t' P s \ominus I \leq t' (\lambda s. P s \ominus I) s$  .
qed
thus  $?thesis$  by( $auto$   $intro!:sub-distrib1$ )
qed

```

For sub-additivity, we again use the limit-of-iterates characterisation. Firstly, all iterates are sublinear:

lemma *sublinear-iterates*:

```

assumes  $hb: healthy$   $(wp$   $body)$ 
and  $sb: sublinear$   $(wp$   $body)$ 
shows  $sublinear$   $(iterates$   $body$   $G$   $i)$ 
by( $induct$   $i$ ,  $auto$   $intro!:sublinear-wp-PC$   $sublinear-wp-Seq$   $sublinear-wp-Skip$   $sublinear-wp-Embed$ 
 $assms$   $healthy-intros$   $iterates-healthy$ )

```

From this, sub-additivity follows for the limit (i.e. the loop), by appealing to the property at all steps.

lemma *sub-add-wp-loop*:

```

fixes  $body::'s$   $prog$ 
assumes  $sb: sublinear$   $(wp$   $body)$ 
and  $cb: bd-cts$   $(wp$   $body)$ 
and  $hwp: healthy$   $(wp$   $body)$ 
shows  $sub-add$   $(wp$   $(do$   $G \longrightarrow body$   $od))$ 
proof
fix  $P$   $Q::'s$  expect and  $s::'s$ 
assume  $sP: sound$   $P$  and  $sQ: sound$   $Q$ 

```

```

from  $hwp$   $cb$   $sP$  have  $(\lambda i. iterates$   $body$   $G$   $i$   $P$   $s) \longrightarrow wp$   $do$   $G \longrightarrow body$   $od$   $P$   $s$ 

```

```

  by(rule loop-iterates)
moreover
from hwp cb sQ have ( $\lambda i$ . iterates body G i Q s)  $\longrightarrow$  wp do G  $\longrightarrow$  body od Q s
  by(rule loop-iterates)
ultimately
have ( $\lambda i$ . iterates body G i P s + iterates body G i Q s)  $\longrightarrow$ 
  wp do G  $\longrightarrow$  body od P s + wp do G  $\longrightarrow$  body od Q s
  by(rule tendsto-add)
moreover {
  from sublinear-subadd[OF sublinear-iterates, OF hwp sb,
    OF healthy-feasibleD[OF iterates-healthy, OF hwp]] sP sQ
  have  $\bigwedge i$ . iterates body G i P s + iterates body G i Q s  $\leq$  iterates body G i ( $\lambda s$ . P s + Q
s) s
  by(rule sub-addD)
}
moreover {
from sP sQ have sound ( $\lambda s$ . P s + Q s) by(blast intro:sound-intros)
with hwp cb have ( $\lambda i$ . iterates body G i ( $\lambda s$ . P s + Q s) s)  $\longrightarrow$ 
  wp do G  $\longrightarrow$  body od ( $\lambda s$ . P s + Q s) s
  by(blast intro:loop-iterates)
}
ultimately
show wp do G  $\longrightarrow$  body od P s + wp do G  $\longrightarrow$  body od Q s  $\leq$  wp do G  $\longrightarrow$  body od ( $\lambda s$ .
P s + Q s) s
  by(blast intro:LIMSEQ-le)
qed

```

lemma sublinear-wp-loop:

```

fixes body::'s prog
assumes hb: healthy (wp body)
  and nhb: nearly-healthy (wlp body)
  and sb: sublinear (wp body)
  and cb: bd-cts (wp body)
shows sublinear (wp (do G  $\longrightarrow$  body od))
using sublinear-sub-distrib[OF sb] sublinear-subadd[OF sb]
  hb healthy-feasibleD[OF hb]
by(iprover intro:sd-sa-sublinear[OF - healthy-wp-loop[OF hb]]
  sub-distrib-wp-loop sub-add-wp-loop assms)

```

lemmas sublinear-intros =

```

sublinear-wp-Abort
sublinear-wp-Skip
sublinear-wp-Apply
sublinear-wp-Seq
sublinear-wp-PC
sublinear-wp-DC
sublinear-wp-SetPC
sublinear-wp-SetDC
sublinear-wp-Embed

```

```

sublinear-wp-repeat
sublinear-wp-Bind
sublinear-wp-loop

```

```
end
```

4.6 Determinism

theory Determinism imports WellDefined begin

We provide a set of lemmas for establishing that appropriately restricted programs are fully additive, and maximal in the refinement order. This is particularly useful with data refinement, as it implies correspondence.

4.6.1 Additivity

lemma additive-wp-Abort:
additive (wp (Abort))
by(*auto simp:wp-eval*)

wlp Abort is not additive.

lemma additive-wp-Skip:
additive (wp (Skip))
by(*auto simp:wp-eval*)

lemma additive-wp-Apply:
additive (wp (Apply f))
by(*auto simp:wp-eval*)

lemma additive-wp-Seq:
fixes *a::'s prog*
assumes *adda: additive (wp a)*
and *addb: additive (wp b)*
and *wb: well-def b*
shows *additive (wp (a ;; b))*
proof(*rule additiveI, unfold wp-eval o-def*)
fix *P::'s ⇒ real and Q::'s ⇒ real and s::'s*
assume *sP: sound P and sQ: sound Q*

note *hb = well-def-wp-healthy[OF wb]*

from *addb sP sQ*
have *wp b (λs. P s + Q s) = (λs. wp b P s + wp b Q s)*
by(*blast dest:additiveD*)
with *adda sP sQ hb*
show *wp a (wp b (λs. P s + Q s)) s =*
wp a (wp b P) s + (wp a (wp b Q)) s
by(*auto intro:fun-cong[OF additiveD]*)

qed

lemma additive-wp-PC:

$\llbracket \text{additive } (wp\ a); \text{additive } (wp\ b) \rrbracket \implies \text{additive } (wp\ (a\ p \oplus\ b))$
by(rule additiveI, simp add:additiveD field-simps wp-eval)

DC is not additive.

lemma additive-wp-SetPC:

$\llbracket \bigwedge x\ s.\ x \in \text{supp } (p\ s) \implies \text{additive } (wp\ (a\ x)); \bigwedge s.\ \text{finite } (\text{supp } (p\ s)) \rrbracket \implies$
 $\text{additive } (wp\ (\text{SetPC } a\ p))$
by(rule additiveI,
 simp add:wp-eval additiveD distrib-left sum.distrib)

lemma additive-wp-Bind:

$\llbracket \bigwedge x.\ \text{additive } (wp\ (a\ (f\ x))) \rrbracket \implies \text{additive } (wp\ (\text{Bind } f\ a))$
by(simp add:wp-eval additive-def)

lemma additive-wp-Embed:

$\llbracket \text{additive } t \rrbracket \implies \text{additive } (wp\ (\text{Embed } t))$
by(simp add:wp-eval)

lemma additive-wp-repeat:

$\text{additive } (wp\ a) \implies \text{well-def } a \implies \text{additive } (wp\ (\text{repeat } n\ a))$
by(induct n, auto simp:additive-wp-Skip intro:additive-wp-Seq wd-intros)

lemmas fa-intros =

additive-wp-Abort additive-wp-Skip
additive-wp-Apply additive-wp-Seq
additive-wp-PC additive-wp-SetPC
additive-wp-Bind additive-wp-Embed
additive-wp-repeat

4.6.2 Maximality

lemma max-wp-Skip:

$\text{maximal } (wp\ \text{Skip})$
by(simp add:maximal-def wp-eval)

lemma max-wp-Apply:

$\text{maximal } (wp\ (\text{Apply } f))$
by(auto simp:wp-eval o-def)

lemma max-wp-Seq:

$\llbracket \text{maximal } (wp\ a); \text{maximal } (wp\ b) \rrbracket \implies \text{maximal } (wp\ (a\ ;;\ b))$
by(simp add:wp-eval maximal-def)

lemma max-wp-PC:

$\llbracket \text{maximal } (wp\ a); \text{maximal } (wp\ b) \rrbracket \implies \text{maximal } (wp\ (a\ p \oplus\ b))$
by(rule maximall, simp add:maximalD field-simps wp-eval)

lemma *max-wp-DC*:

$\llbracket \text{maximal } (wp\ a); \text{maximal } (wp\ b) \rrbracket \implies \text{maximal } (wp\ (a \sqcap b))$
by(rule *maximall*, simp add:wp-eval *maximalD*)

lemma *max-wp-SetPC*:

$\llbracket \bigwedge s\ a.\ a \in \text{supp } (P\ s) \implies \text{maximal } (wp\ (p\ a)); \bigwedge s.\ (\sum a \in \text{supp } (P\ s).\ P\ s\ a) = I \rrbracket \implies$
 $\text{maximal } (wp\ (\text{SetPC } p\ P))$
by(auto simp: *maximalD* wp-def *SetPC-def* sum-distrib-right[symmetric])

lemma *max-wp-SetDC*:

fixes $p::'a \Rightarrow 's\ \text{prog}$
assumes $mp: \bigwedge s\ a.\ a \in S\ s \implies \text{maximal } (wp\ (p\ a))$
and $ne: \bigwedge s.\ S\ s \neq \{\}$
shows $\text{maximal } (wp\ (\text{SetDC } p\ S))$
proof(rule *maximall*, rule *ext*, unfold wp-eval)
fix $c::\text{real}$ **and** $s::'s$
assume $0 \leq c$
hence $\text{Inf } ((\lambda a.\ wp\ (p\ a)\ (\lambda\cdot.\ c)\ s) \text{ ` } S\ s) = \text{Inf } ((\lambda\cdot.\ c) \text{ ` } S\ s)$
using mp **by**(simp add:*maximalD* cong: *image-cong*)
also {
from ne **obtain** a **where** $a \in S\ s$ **by** *blast*
hence $\text{Inf } ((\lambda\cdot.\ c) \text{ ` } S\ s) = c$
by (auto simp add: *image-constant-conv* cong del: *INF-cong-simp*)
}
finally show $\text{Inf } ((\lambda a.\ wp\ (p\ a)\ (\lambda\cdot.\ c)\ s) \text{ ` } S\ s) = c$.
qed

lemma *max-wp-Embed*:

$\text{maximal } t \implies \text{maximal } (wp\ (\text{Embed } t))$
by(simp add:wp-eval)

lemma *max-wp-repeat*:

$\text{maximal } (wp\ a) \implies \text{maximal } (wp\ (\text{repeat } n\ a))$
by(induct n , simp-all add: *max-wp-Skip* *max-wp-Seq*)

lemma *max-wp-Bind*:

assumes $ma: \bigwedge s.\ \text{maximal } (wp\ (a\ (f\ s)))$
shows $\text{maximal } (wp\ (\text{Bind } f\ a))$
proof(rule *maximall*, rule *ext*, simp add:wp-eval)
fix $c::\text{real}$ **and** s
assume $0 \leq c$
with ma **have** $wp\ (a\ (f\ s))\ (\lambda\cdot.\ c) = (\lambda\cdot.\ c)$ **by**(*blast*)
thus $wp\ (a\ (f\ s))\ (\lambda\cdot.\ c)\ s = c$ **by**(auto)
qed

lemmas *max-intros* =

max-wp-Skip *max-wp-Apply*
max-wp-Seq *max-wp-PC*

max-wp-DC max-wp-SetPC
max-wp-SetDC max-wp-Embed
max-wp-Bind max-wp-repeat

A healthy transformer that terminates is maximal.

lemma *healthy-term-max*:
assumes *ht*: *healthy t*
and *trm*: $\lambda s. I \Vdash t (\lambda s. I)$
shows *maximal t*
proof(*intro maximalI ext*)
fix *c::real* **and** *s*
assume *nnc*: $0 \leq c$

have $t (\lambda s. c) s = t (\lambda s. I * c) s$ **by**(*simp*)
also from *nnc healthy-scalingD[OF ht]*
have $\dots = c * t (\lambda s. I) s$ **by**(*simp add:scalingD*)
also {
from *ht* **have** $t (\lambda s. I) \Vdash \lambda s. I$ **by**(*auto*)
with *trm* **have** $t (\lambda s. I) = (\lambda s. I)$ **by**(*auto*)
hence $c * t (\lambda s. I) s = c$ **by**(*simp*)
}
finally show $t (\lambda s. c) s = c$.
qed

4.6.3 Determinism

lemma *det-wp-Skip*:
determ (wp Skip)
using *max-intros fa-intros* **by**(*blast*)

lemma *det-wp-Apply*:
determ (wp (Apply f))
by(*intro determI fa-intros max-intros*)

lemma *det-wp-Seq*:
determ (wp a) \implies determ (wp b) \implies well-def b \implies determ (wp (a ;; b))
by(*intro determI fa-intros max-intros, auto*)

lemma *det-wp-PC*:
determ (wp a) \implies determ (wp b) \implies determ (wp (a \oplus b))
by(*intro determI fa-intros max-intros, auto*)

lemma *det-wp-SetPC*:
 $(\bigwedge x s. x \in \text{supp } (p s) \implies \text{determ } (wp (a x))) \implies$
 $(\bigwedge s. \text{finite } (\text{supp } (p s))) \implies$
 $(\bigwedge s. \text{sum } (p s) (\text{supp } (p s)) = I) \implies$
determ (wp (SetPC a p))
by(*intro determI fa-intros max-intros, auto*)

lemma *det-wp-Bind*:

$(\wedge x. \text{determ} (\text{wp} (a (fx)))) \implies \text{determ} (\text{wp} (\text{Bind } f a))$
by(*intro determI fa-intros max-intros, auto*)

lemma *det-wp-Embed*:

$\text{determ } t \implies \text{determ} (\text{wp} (\text{Embed } t))$
by(*simp add:wp-eval*)

lemma *det-wp-repeat*:

$\text{determ} (\text{wp } a) \implies \text{well-def } a \implies \text{determ} (\text{wp} (\text{repeat } n a))$
by(*intro determI fa-intros max-intros, auto*)

lemmas *determ-intros =*

det-wp-Skip det-wp-Apply
det-wp-Seq det-wp-PC
det-wp-SetPC det-wp-Bind
det-wp-Embed det-wp-repeat

end

4.7 Well-Defined Programs.

theory *WellDefined imports*

Healthiness
Sublinearity
LoopInduction

begin

The definition of a well-defined program collects the various notions of healthiness and well-behavedness that we have so far established: healthiness of the strict and liberal transformers, continuity and sublinearity of the strict transformers, and two new properties. These are that the strict transformer always lies below the liberal one (i.e. that it is at least as *strict*, recalling the standard embedding of a predicate), and that expectation conjunction is distributed between then in a particular manner, which will be crucial in establishing the loop rules.

4.7.1 Strict Implies Liberal

This establishes the first connection between the strict and liberal interpretations (*wp* and *wlp*).

definition

wp-under-wlp :: '*s prog* \Rightarrow *bool*

where

wp-under-wlp prog $\equiv \forall P. \text{unitary } P \longrightarrow \text{wp } \text{prog } P \Vdash \text{wlp } \text{prog } P$

lemma *wp-under-wlpI[intro]*:

$\llbracket \wedge P. \text{unitary } P \implies \text{wp } \text{prog } P \Vdash \text{wlp } \text{prog } P \rrbracket \implies \text{wp-under-wlp } \text{prog}$

unfolding *wp-under-wlp-def* **by**(*simp*)

lemma *wp-under-wlpD[dest]*:

$\llbracket \text{wp-under-wlp prog; unitary } P \rrbracket \implies \text{wp prog } P \Vdash \text{wlp prog } P$
unfolding *wp-under-wlp-def* **by**(*simp*)

lemma *wp-under-le-trans*:

$\text{wp-under-wlp } a \implies \text{le-utrans } (\text{wp } a) (\text{wlp } a)$
by(*blast*)

lemma *wp-under-wlp-Abort*:

wp-under-wlp Abort
by(*rule wp-under-wlpI, unfold wp-eval, auto*)

lemma *wp-under-wlp-Skip*:

wp-under-wlp Skip
by(*rule wp-under-wlpI, unfold wp-eval, blast*)

lemma *wp-under-wlp-Apply*:

wp-under-wlp (Apply f)
by(*auto simp:wp-eval*)

lemma *wp-under-wlp-Seq*:

assumes *h-wlp-a: nearly-healthy (wlp a)*
and *h-wp-b: healthy (wp b)*
and *h-wlp-b: nearly-healthy (wlp b)*
and *wp-u-a: wp-under-wlp a*
and *wp-u-b: wp-under-wlp b*
shows *wp-under-wlp (a ;; b)*

proof(*rule wp-under-wlpI, unfold wp-eval o-def*)

fix *P::'a \Rightarrow real* **assume** *uP: unitary P*
with *h-wp-b* **have** *unitary (wp b P)* **by**(*blast*)
with *wp-u-a* **have** $\text{wp } a (\text{wp } b P) \Vdash \text{wlp } a (\text{wp } b P)$ **by**(*auto*)
also {
from *wp-u-b* **and** *uP* **have** $\text{wp } b P \Vdash \text{wlp } b P$ **by**(*blast*)
with *h-wlp-a* **and** *h-wlp-b* **and** *h-wp-b* **and** *uP*
have $\text{wlp } a (\text{wp } b P) \Vdash \text{wlp } a (\text{wlp } b P)$
by(*blast intro:nearly-healthy-monoD[OF h-wlp-a]*)
}

finally show $\text{wp } a (\text{wp } b P) \Vdash \text{wlp } a (\text{wlp } b P)$.

qed

lemma *wp-under-wlp-PC*:

assumes *h-wp-a: healthy (wp a)*
and *h-wlp-a: nearly-healthy (wlp a)*
and *h-wp-b: healthy (wp b)*
and *h-wlp-b: nearly-healthy (wlp b)*
and *wp-u-a: wp-under-wlp a*
and *wp-u-b: wp-under-wlp b*

and uP : unitary P
shows $wp\text{-under}\text{-wlp} (a \oplus b)$
proof(rule $wp\text{-under}\text{-wlpI}$, unfold $wp\text{-eval}$, rule $le\text{-funI}$)
fix $Q::'a \Rightarrow \text{real}$ **and** s
assume uQ : unitary Q
from uP **have** $P s \leq 1$ **by**($blast$)
hence $0 \leq 1 - P s$ **by**($simp$)
moreover
from uQ **and** $wp\text{-u}\text{-}b$ **have** $wp\ b\ Q\ s \leq wlp\ b\ Q\ s$ **by**($blast$)
ultimately
have $(1 - P s) * wp\ b\ Q\ s \leq (1 - P s) * wlp\ b\ Q\ s$
by($blast\ intro$: $mult\text{-left}\text{-mono}$)

moreover {
from uQ **and** $wp\text{-u}\text{-}a$ **have** $wp\ a\ Q\ s \leq wlp\ a\ Q\ s$ **by**($blast$)
with uP **have** $P s * wp\ a\ Q\ s \leq P s * wlp\ a\ Q\ s$
by($blast\ intro$: $mult\text{-left}\text{-mono}$)
}

ultimately
show $P s * wp\ a\ Q\ s + (1 - P s) * wp\ b\ Q\ s \leq$
 $P s * wlp\ a\ Q\ s + (1 - P s) * wlp\ b\ Q\ s$
by($blast\ intro$: $add\text{-mono}$)
qed

lemma $wp\text{-under}\text{-wlp}\text{-DC}$:
assumes $wp\text{-u}\text{-}a$: $wp\text{-under}\text{-wlp}\ a$
and $wp\text{-u}\text{-}b$: $wp\text{-under}\text{-wlp}\ b$
shows $wp\text{-under}\text{-wlp}\ (a \sqcap b)$
proof(rule $wp\text{-under}\text{-wlpI}$, unfold $wp\text{-eval}$, rule $le\text{-funI}$)
fix $Q::'a \Rightarrow \text{real}$ **and** s
assume uQ : unitary Q

from $wp\text{-u}\text{-}a\ uQ$ **have** $wp\ a\ Q\ s \leq wlp\ a\ Q\ s$ **by**($blast$)
moreover
from $wp\text{-u}\text{-}b\ uQ$ **have** $wp\ b\ Q\ s \leq wlp\ b\ Q\ s$ **by**($blast$)
ultimately
show $\min (wp\ a\ Q\ s) (wp\ b\ Q\ s) \leq \min (wlp\ a\ Q\ s) (wlp\ b\ Q\ s)$
by($auto$)
qed

lemma $wp\text{-under}\text{-wlp}\text{-SetPC}$:
assumes $wp\text{-u}\text{-}f$: $\bigwedge s\ a. a \in \text{supp}\ (P\ s) \Longrightarrow wp\text{-under}\text{-wlp}\ (f\ a)$
and nP : $\bigwedge s\ a. a \in \text{supp}\ (P\ s) \Longrightarrow 0 \leq P\ s\ a$
shows $wp\text{-under}\text{-wlp}\ (SetPC\ f\ P)$
proof(rule $wp\text{-under}\text{-wlpI}$, unfold $wp\text{-eval}$, rule $le\text{-funI}$)
fix $Q::'a \Rightarrow \text{real}$ **and** s
assume uQ : unitary Q

from $wp\text{-}u\text{-}f\ uQ\ nP$
show $(\sum a \in \text{supp}(P\ s). P\ s\ a * wp\ (f\ a)\ Q\ s) \leq (\sum a \in \text{supp}(P\ s). P\ s\ a * wlp\ (f\ a)\ Q\ s)$
by(*auto intro! :sum-mono mult-left-mono*)
qed

lemma $wp\text{-}under\text{-}wlp\text{-}SetDC$:

assumes $wp\text{-}u\text{-}f$: $\bigwedge s\ a. a \in S\ s \implies wp\text{-}under\text{-}wlp\ (f\ a)$
and hf : $\bigwedge s\ a. a \in S\ s \implies healthy\ (wp\ (f\ a))$
and nS : $\bigwedge s. S\ s \neq \{\}$
shows $wp\text{-}under\text{-}wlp\ (SetDC\ f\ S)$
proof(*rule wp-under-wlpI, rule le-funI, unfold wp-eval*)
fix $Q::'a \Rightarrow real$ **and** s
assume uQ : *unitary Q*

show $Inf\ ((\lambda a. wp\ (f\ a)\ Q\ s) \text{' } S\ s) \leq Inf\ ((\lambda a. wlp\ (f\ a)\ Q\ s) \text{' } S\ s)$
proof(*rule cInf-mono*)
from nS **show** $(\lambda a. wlp\ (f\ a)\ Q\ s) \text{' } S\ s \neq \{\}$ **by**(*blast*)

fix x **assume** xin : $x \in (\lambda a. wlp\ (f\ a)\ Q\ s) \text{' } S\ s$
then obtain a **where** ain : $a \in S\ s$ **and** xrw : $x = wlp\ (f\ a)\ Q\ s$
by(*blast*)
with $wp\text{-}u\text{-}f\ uQ$
have $wp\ (f\ a)\ Q\ s \leq wlp\ (f\ a)\ Q\ s$ **by**(*blast*)
moreover from ain **have** $wp\ (f\ a)\ Q\ s \in (\lambda a. wp\ (f\ a)\ Q\ s) \text{' } S\ s$
by(*blast*)
ultimately show $\exists y \in (\lambda a. wp\ (f\ a)\ Q\ s) \text{' } S\ s. y \leq x$
by(*auto simp:xrw*)

next

fix y **assume** yin : $y \in (\lambda a. wp\ (f\ a)\ Q\ s) \text{' } S\ s$
then obtain a **where** ain : $a \in S\ s$ **and** yrw : $y = wp\ (f\ a)\ Q\ s$
by(*blast*)
with $hf\ uQ$ **have** *unitary (wp (f a) Q)* **by**(*auto*)
with yrw **show** $0 \leq y$ **by**(*auto*)

qed

qed

lemma $wp\text{-}under\text{-}wlp\text{-}Embed$:

$wp\text{-}under\text{-}wlp\ (Embed\ t)$
by(*rule wp-under-wlpI, unfold wp-eval, blast*)

lemma $wp\text{-}under\text{-}wlp\text{-}loop$:

fixes $body::'s\ prog$
assumes hwp : *healthy (wp body)*
and $hwlp$: *nearly-healthy (wlp body)*
and $wp\text{-}under$: $wp\text{-}under\text{-}wlp\ body$
shows $wp\text{-}under\text{-}wlp\ (do\ G \longrightarrow body\ od)$
proof(*rule wp-under-wlpI*)
fix $P::'s\ expect$

assume uP : unitary P **hence** sP : sound P **by**(*auto*)
let $?X \ Q \ s = \langle G \rangle s * wp \ body \ Q \ s + \langle \mathcal{N} \ G \rangle s * P \ s$
let $?Y \ Q \ s = \langle G \rangle s * wlp \ body \ Q \ s + \langle \mathcal{N} \ G \rangle s * P \ s$
show $wp \ (do \ G \ \longrightarrow \ body \ od) \ P \Vdash wlp \ (do \ G \ \longrightarrow \ body \ od) \ P$
proof(*simp add:hwp hwlp sP uP wp-Loop1 wlp-Loop1, rule gfp-exp-upperbound*)
thm *lfp-loop-fp*
from $hwp \ sP$ **have** $lfp\text{-}exp \ ?X = ?X \ (lfp\text{-}exp \ ?X)$
by(*rule lfp-wp-loop-unfold*)
hence $lfp\text{-}exp \ ?X \Vdash ?X \ (lfp\text{-}exp \ ?X)$ **by**(*simp*)
also {
from $hwp \ uP$ **have** $wp \ body \ (lfp\text{-}exp \ ?X) \Vdash wlp \ body \ (lfp\text{-}exp \ ?X)$
by(*auto intro:wp-under-wlpD[OF wp-under] lfp-loop-unitary*)
hence $?X \ (lfp\text{-}exp \ ?X) \Vdash ?Y \ (lfp\text{-}exp \ ?X)$
by(*auto intro:add-mono mult-left-mono*)
}
finally show $lfp\text{-}exp \ ?X \Vdash ?Y \ (lfp\text{-}exp \ ?X)$.
from $hwp \ uP$ **show** unitary $(lfp\text{-}exp \ ?X)$
by(*auto intro:lfp-loop-unitary*)
qed
qed

lemma *wp-under-wlp-repeat*:
 $\llbracket healthy \ (wp \ a); \ nearly\text{-}healthy \ (wlp \ a); \ wp\text{-}under\text{-}wlp \ a \rrbracket \Longrightarrow$
 $wp\text{-}under\text{-}wlp \ (repeat \ n \ a)$
by(*induct n, auto intro!:wp-under-wlp-Skip wp-under-wlp-Seq healthy-intros*)

lemma *wp-under-wlp-Bind*:
 $\llbracket \bigwedge s. wp\text{-}under\text{-}wlp \ (a \ (f \ s)) \rrbracket \Longrightarrow wp\text{-}under\text{-}wlp \ (Bind \ f \ a)$
unfolding *wp-under-wlp-def* **by**(*auto simp:wp-eval*)

lemmas *wp-under-wlp-intros* =
wp-under-wlp-Abort wp-under-wlp-Skip
wp-under-wlp-Apply wp-under-wlp-Seq
wp-under-wlp-PC wp-under-wlp-DC
wp-under-wlp-SetPC wp-under-wlp-SetDC
wp-under-wlp-Embed wp-under-wlp-loop
wp-under-wlp-repeat wp-under-wlp-Bind

4.7.2 Sub-Distributivity of Conjunction

definition

sub-distrib-pconj :: ' $s \ prog \Rightarrow bool$

where

sub-distrib-pconj $prog \equiv$

$\forall P \ Q. \ unitary \ P \ \longrightarrow \ unitary \ Q \ \longrightarrow$

$wlp \ prog \ P \ \&\& \ wp \ prog \ Q \ \Vdash \ wp \ prog \ (P \ \&\& \ Q)$

lemma *sub-distrib-pconjI*[*intro*]:
 $\llbracket \bigwedge P Q. \llbracket \text{unitary } P; \text{unitary } Q \rrbracket \implies \text{wlp prog } P \ \&\& \ \text{wp prog } Q \Vdash \text{wp prog } (P \ \&\& \ Q) \rrbracket$
 \implies
sub-distrib-pconj prog
unfolding *sub-distrib-pconj-def by(simp)*

lemma *sub-distrib-pconjD*[*dest*]:
 $\bigwedge P Q. \llbracket \text{sub-distrib-pconj prog}; \text{unitary } P; \text{unitary } Q \rrbracket \implies$
 $\text{wlp prog } P \ \&\& \ \text{wp prog } Q \Vdash \text{wp prog } (P \ \&\& \ Q)$
unfolding *sub-distrib-pconj-def by(simp)*

lemma *sdp-Abort*:
sub-distrib-pconj Abort
by(*rule sub-distrib-pconjI, unfold wp-eval, auto intro:exp-conj-rzero*)

lemma *sdp-Skip*:
sub-distrib-pconj Skip
by(*rule sub-distrib-pconjI, simp add:wp-eval*)

lemma *sdp-Seq*:
fixes *a and b*
assumes *sdp-a: sub-distrib-pconj a*
and *sdp-b: sub-distrib-pconj b*
and *h-wp-a: healthy (wp a)*
and *h-wp-b: healthy (wp b)*
and *h-wlp-b: nearly-healthy (wlp b)*
shows *sub-distrib-pconj (a ;; b)*
proof(*rule sub-distrib-pconjI, unfold wp-eval o-def*)
fix *P::'a \Rightarrow real and Q::'a \Rightarrow real*
assume *uP: unitary P and uQ: unitary Q*

with *h-wp-b and h-wlp-b*
have *wlp a (wlp b P) && wp a (wp b Q) \Vdash wp a (wlp b P && wp b Q)*
by(*blast intro!:sub-distrib-pconjD[OF sdp-a]*)
also {
from *sdp-b and uP and uQ*
have *wlp b P && wp b Q \Vdash wp b (P && Q)* **by**(*blast*)
with *h-wp-a h-wp-b h-wlp-b uP uQ*
have *wp a (wlp b P && wp b Q) \Vdash wp a (wp b (P && Q))*
by(*blast intro!:mono-transD[OF healthy-monoD, OF h-wp-a] unitary-sound*
unitary-intros sound-intros)
}
finally show *wlp a (wlp b P) && wp a (wp b Q) \Vdash wp a (wp b (P && Q))* .
qed

lemma *sdp-Apply*:
sub-distrib-pconj (Apply f)
by(*rule sub-distrib-pconjI, simp add:wp-eval*)

lemma *sdp-DC*:
fixes $a::'s$ prog **and** b
assumes *sdp-a*: sub-distrib-pconj a
and *sdp-b*: sub-distrib-pconj b
and *h-wp-a*: healthy (wp a)
and *h-wp-b*: healthy (wp b)
and *h-wlp-b*: nearly-healthy (wlp b)
shows sub-distrib-pconj ($a \sqcap b$)
proof(rule sub-distrib-pconjI, unfold wp-eval, rule le-funI)
fix $P::'s \Rightarrow \text{real}$ **and** $Q::'s \Rightarrow \text{real}$ **and** $s::'s$
assume uP : unitary P **and** uQ : unitary Q

have $((\lambda s. \min (\text{wlp } a \ P \ s) (\text{wlp } b \ P \ s)) \ \&\&$
 $(\lambda s. \min (\text{wp } a \ Q \ s) (\text{wp } b \ Q \ s))) \ s \leq$
 $\min (\text{wlp } a \ P \ s \ .\& \ \text{wp } a \ Q \ s) (\text{wlp } b \ P \ s \ .\& \ \text{wp } b \ Q \ s)$
unfolding exp-conj-def **by**(rule min-pconj)
also {
have $(\lambda s. \text{wlp } a \ P \ s \ .\& \ \text{wp } a \ Q \ s) = \text{wlp } a \ P \ \&\& \ \text{wp } a \ Q$
by(simp add:exp-conj-def)
also from *sdp-a* $uP \ uQ$ **have** $\dots \Vdash \text{wp } a \ (P \ \&\& \ Q)$
by(blast dest:sub-distrib-pconjD)
finally have $\text{wlp } a \ P \ s \ .\& \ \text{wp } a \ Q \ s \leq \text{wp } a \ (P \ \&\& \ Q) \ s$
by(rule le-funD)
moreover {
have $(\lambda s. \text{wlp } b \ P \ s \ .\& \ \text{wp } b \ Q \ s) = \text{wlp } b \ P \ \&\& \ \text{wp } b \ Q$
by(simp add:exp-conj-def)
also from *sdp-b* $uP \ uQ$ **have** $\dots \Vdash \text{wp } b \ (P \ \&\& \ Q)$
by(blast)
finally have $\text{wlp } b \ P \ s \ .\& \ \text{wp } b \ Q \ s \leq \text{wp } b \ (P \ \&\& \ Q) \ s$
by(rule le-funD)
}
ultimately
have $\min (\text{wlp } a \ P \ s \ .\& \ \text{wp } a \ Q \ s) (\text{wlp } b \ P \ s \ .\& \ \text{wp } b \ Q \ s) \leq$
 $\min (\text{wp } a \ (P \ \&\& \ Q) \ s) (\text{wp } b \ (P \ \&\& \ Q) \ s)$ **by**(auto)
}
finally
show $((\lambda s. \min (\text{wlp } a \ P \ s) (\text{wlp } b \ P \ s)) \ \&\&$
 $(\lambda s. \min (\text{wp } a \ Q \ s) (\text{wp } b \ Q \ s))) \ s \leq$
 $\min (\text{wp } a \ (P \ \&\& \ Q) \ s) (\text{wp } b \ (P \ \&\& \ Q) \ s) .$

qed

lemma *sdp-PC*:
fixes $a::'s$ prog **and** b
assumes *sdp-a*: sub-distrib-pconj a
and *sdp-b*: sub-distrib-pconj b
and *h-wp-a*: healthy (wp a)
and *h-wp-b*: healthy (wp b)
and *h-wlp-b*: nearly-healthy (wlp b)
and uP : unitary P

shows *sub-distrib-pconj* ($a \oplus b$)
proof(*rule sub-distrib-pconjI*, *unfold wp-eval*, *rule le-funI*)
fix $Q::'s \Rightarrow \text{real}$ **and** $R::'s \Rightarrow \text{real}$ **and** $s::'s$
assume uQ : *unitary* Q **and** uR : *unitary* R

have $nnA: 0 \leq P s$ **and** $nnB: P s \leq 1$
using uP **by** *auto*
note $nn = nnA \text{ } nnB$

have $((\lambda s. P s * wlp a Q s + (1 - P s) * wlp b Q s) \&\&$
 $(\lambda s. P s * wp a R s + (1 - P s) * wp b R s)) s \leq$
 $((P s * wlp a Q s + (1 - P s) * wlp b Q s) +$
 $(P s * wp a R s + (1 - P s) * wp b R s)) \ominus 1$
by(*simp add:exp-conj-def pconj-def*)
also have $\dots = P s * (wlp a Q s + wp a R s) +$
 $(1 - P s) * (wlp b Q s + wp b R s) \ominus 1$
by(*simp add:field-simps*)
also have $\dots = P s * (wlp a Q s + wp a R s) +$
 $(1 - P s) * (wlp b Q s + wp b R s) \ominus$
 $(P s + (1 - P s))$
by(*simp*)
also have $\dots \leq (P s * (wlp a Q s + wp a R s) \ominus P s) +$
 $((1 - P s) * (wlp b Q s + wp b R s) \ominus (1 - P s))$
by(*rule tminus-add-mono*)
also have $\dots = (P s * (wlp a Q s + wp a R s \ominus 1)) +$
 $((1 - P s) * (wlp b Q s + wp b R s \ominus 1))$
by(*simp add:nn tminus-left-distrib*)
also have $\dots = P s * ((wlp a Q \&\& wp a R) s) +$
 $(1 - P s) * ((wlp b Q \&\& wp b R) s)$
by(*simp add:exp-conj-def pconj-def*)
also {
from *sdp-a sdp-b uQ uR*
have $P s * (wlp a Q \&\& wp a R) s \leq P s * wp a (Q \&\& R) s$
and $(1 - P s) * (wlp b Q \&\& wp b R) s \leq (1 - P s) * wp b (Q \&\& R) s$
by (*simp-all add: entailsD mult-left-mono nn sub-distrib-pconjD*)
hence $P s * ((wlp a Q \&\& wp a R) s) +$
 $(1 - P s) * ((wlp b Q \&\& wp b R) s) \leq$
 $P s * wp a (Q \&\& R) s + (1 - P s) * wp b (Q \&\& R) s$
by(*auto*)
}

finally show $((\lambda s. P s * wlp a Q s + (1 - P s) * wlp b Q s) \&\&$
 $(\lambda s. P s * wp a R s + (1 - P s) * wp b R s)) s \leq$
 $P s * wp a (Q \&\& R) s + (1 - P s) * wp b (Q \&\& R) s .$

qed

lemma *sdp-Embed*:

$\llbracket \bigwedge P Q. \llbracket \text{unitary } P; \text{unitary } Q \rrbracket \implies t P \&\& t Q \vdash t (P \&\& Q) \rrbracket \implies$
sub-distrib-pconj (*Embed* t)
by(*auto simp:wp-eval*)

lemma *sdp-repeat*:
fixes $a::'s\ prog$
assumes $sdp_a: sub\text{-}distrib\text{-}pconj\ a$
and $hwp: healthy\ (wp\ a)$ **and** $hwlp: nearly\text{-}healthy\ (wlp\ a)$
shows $sub\text{-}distrib\text{-}pconj\ (repeat\ n\ a)$ (**is** $?X\ n$)
proof(*induct n*)
show $?X\ 0$ **by**(*simp add:sdp-Skip*)
fix n **assume** $IH: ?X\ n$
show $?X\ (Suc\ n)$
proof(*rule sub-distrib-pconjI, simp add:wp-eval*)
fix $P::'s\ \Rightarrow\ real$ **and** $Q::'s\ \Rightarrow\ real$
assume $uP: unitary\ P$ **and** $uQ: unitary\ Q$
from $assms$ **have** $hwlp_a: nearly\text{-}healthy\ (wlp\ (repeat\ n\ a))$
and $hwp_a: healthy\ (wp\ (repeat\ n\ a))$
by(*auto intro:healthy-intros*)
from uP **and** $hwlp_a$ **have** $unitary\ (wlp\ (repeat\ n\ a)\ P)$ **by**(*blast*)
moreover **from** uQ **and** hwp_a **have** $unitary\ (wp\ (repeat\ n\ a)\ Q)$ **by**(*blast*)
ultimately
have $wlp\ a\ (wlp\ (repeat\ n\ a)\ P)\ \&\&\ wp\ a\ (wp\ (repeat\ n\ a)\ Q)\ \Vdash$
 $wp\ a\ (wlp\ (repeat\ n\ a)\ P)\ \&\&\ wp\ (repeat\ n\ a)\ Q$
using sdp_a **by**(*blast*)
also {
from $hwlp$ **have** $nearly\text{-}healthy\ (wlp\ (repeat\ n\ a))$ **by**(*rule healthy-intros*)
with uP **have** $sound\ (wlp\ (repeat\ n\ a)\ P)$ **by**(*auto*)
moreover **from** $hwp\ uQ$ **have** $sound\ (wp\ (repeat\ n\ a)\ Q)$
by(*auto intro:healthy-intros*)
ultimately **have** $sound\ (wlp\ (repeat\ n\ a)\ P)\ \&\&\ wp\ (repeat\ n\ a)\ Q$
by(*rule exp-conj-sound*)
moreover {
from $uP\ uQ$ **have** $sound\ (P\ \&\&\ Q)$ **by**(*auto intro:exp-conj-sound*)
with hwp **have** $sound\ (wp\ (repeat\ n\ a)\ (P\ \&\&\ Q))$
by(*auto intro:healthy-intros*)
}
moreover **from** $uP\ uQ\ IH$
have $wlp\ (repeat\ n\ a)\ P\ \&\&\ wp\ (repeat\ n\ a)\ Q\ \Vdash\ wp\ (repeat\ n\ a)\ (P\ \&\&\ Q)$
by(*blast*)
ultimately
have $wp\ a\ (wlp\ (repeat\ n\ a)\ P)\ \&\&\ wp\ (repeat\ n\ a)\ Q)\ \Vdash$
 $wp\ a\ (wp\ (repeat\ n\ a)\ (P\ \&\&\ Q))$
by(*rule mono-transD[OF healthy-monoD, OF hwp]*)
}
finally **show** $wlp\ a\ (wlp\ (repeat\ n\ a)\ P)\ \&\&\ wp\ a\ (wp\ (repeat\ n\ a)\ Q)\ \Vdash$
 $wp\ a\ (wp\ (repeat\ n\ a)\ (P\ \&\&\ Q))$.
qed
qed

lemma *sdp-SetPC*:
fixes $p::'a\ \Rightarrow\ 's\ prog$

assumes $sdp: \bigwedge s a. a \in \text{supp } (P s) \implies \text{sub-distrib-pconj } (p a)$
and $fin: \bigwedge s. \text{finite } (\text{supp } (P s))$
and $nnp: \bigwedge s a. 0 \leq P s a$
and $sub: \bigwedge s. \text{sum } (P s) (\text{supp } (P s)) \leq 1$
shows $\text{sub-distrib-pconj } (\text{SetPC } p P)$
proof(*rule sub-distrib-pconjI, simp add:wp-eval, rule le-funI*)
fix $Q::'s \Rightarrow \text{real}$ **and** $R::'s \Rightarrow \text{real}$ **and** $s::'s$
assume $uQ: \text{unitary } Q$ **and** $uR: \text{unitary } R$
have $((\lambda s. \sum a \in \text{supp } (P s). P s a * \text{wlp } (p a) Q s) \&\&$
 $(\lambda s. \sum a \in \text{supp } (P s). P s a * \text{wp } (p a) R s)) s =$
 $(\sum a \in \text{supp } (P s). P s a * \text{wlp } (p a) Q s) + (\sum a \in \text{supp } (P s). P s a * \text{wp } (p a) R s) \ominus$
 I
by(*simp add:exp-conj-def pconj-def*)
also have $\dots = (\sum a \in \text{supp } (P s). P s a * (\text{wlp } (p a) Q s + \text{wp } (p a) R s)) \ominus I$
by(*simp add:sum.distrib field-simps*)
also from sub
have $\dots \leq (\sum a \in \text{supp } (P s). P s a * (\text{wlp } (p a) Q s + \text{wp } (p a) R s)) \ominus$
 $(\sum a \in \text{supp } (P s). P s a)$
by(*rule tminus-right-antimono*)
also from fin
have $\dots \leq (\sum a \in \text{supp } (P s). P s a * (\text{wlp } (p a) Q s + \text{wp } (p a) R s) \ominus P s a)$
by(*rule tminus-sum-mono*)
also from nnp
have $\dots = (\sum a \in \text{supp } (P s). P s a * (\text{wlp } (p a) Q s + \text{wp } (p a) R s \ominus I))$
by(*simp add:tminus-left-distrib*)
also have $\dots = (\sum a \in \text{supp } (P s). P s a * (\text{wlp } (p a) Q \&\& \text{wp } (p a) R) s)$
by(*simp add:pconj-def exp-conj-def*)
also {
from $sdp uQ uR$
have $\bigwedge a. a \in \text{supp } (P s) \implies \text{wlp } (p a) Q \&\& \text{wp } (p a) R \Vdash \text{wp } (p a) (Q \&\& R)$
by(*blast intro:sub-distrib-pconjD*)
with nnp
have $(\sum a \in \text{supp } (P s). P s a * (\text{wlp } (p a) Q \&\& \text{wp } (p a) R) s) \leq$
 $(\sum a \in \text{supp } (P s). P s a * (\text{wp } (p a) (Q \&\& R)) s)$
by(*blast intro:sum-mono mult-left-mono*)
}
finally show $((\lambda s. \sum a \in \text{supp } (P s). P s a * \text{wlp } (p a) Q s) \&\&$
 $(\lambda s. \sum a \in \text{supp } (P s). P s a * \text{wp } (p a) R s)) s \leq$
 $(\sum a \in \text{supp } (P s). P s a * \text{wp } (p a) (Q \&\& R) s).$
qed

lemma $sdp\text{-SetDC}$:

fixes $p::'a \Rightarrow 's \text{ prog}$
assumes $sdp: \bigwedge s a. a \in S s \implies \text{sub-distrib-pconj } (p a)$
and $hwp: \bigwedge s a. a \in S s \implies \text{healthy } (\text{wp } (p a))$
and $hwlp: \bigwedge s a. a \in S s \implies \text{nearly-healthy } (\text{wlp } (p a))$
and $ne: \bigwedge s. S s \neq \{\}$
shows $\text{sub-distrib-pconj } (\text{SetDC } p S)$
proof(*rule sub-distrib-pconjI, rule le-funI*)

```

fix  $P:: 's \Rightarrow \text{real}$  and  $Q:: 's \Rightarrow \text{real}$  and  $s:: 's$ 
assume  $uP$ : unitary  $P$  and  $uQ$ : unitary  $Q$ 

from  $uP$  hwlp
have  $\bigwedge x. x \in (\lambda a. \text{wlp } (p \ a) \ P) \ 'S \ s \Longrightarrow \text{unitary } x$  by (auto)
hence  $\bigwedge y. y \in (\lambda a. \text{wlp } (p \ a) \ P \ s) \ 'S \ s \Longrightarrow 0 \leq y$  by (auto)
hence  $\bigwedge a. a \in S \ s \Longrightarrow \text{wlp } (\text{SetDC } p \ S) \ P \ s \leq \text{wlp } (p \ a) \ P \ s$ 
unfolding wp-eval by (intro cInf-lower bdd-belowI, auto)
moreover {
  from  $uQ$  hwp have  $\bigwedge a. a \in S \ s \Longrightarrow 0 \leq \text{wp } (p \ a) \ Q \ s$  by (blast)
  hence  $\bigwedge a. a \in S \ s \Longrightarrow \text{wp } (\text{SetDC } p \ S) \ Q \ s \leq \text{wp } (p \ a) \ Q \ s$ 
  unfolding wp-eval by (intro cInf-lower bdd-belowI, auto)
}
ultimately
have  $\bigwedge a. a \in S \ s \Longrightarrow \text{wlp } (\text{SetDC } p \ S) \ P \ s + \text{wp } (\text{SetDC } p \ S) \ Q \ s \ominus I \leq$ 
 $\text{wlp } (p \ a) \ P \ s + \text{wp } (p \ a) \ Q \ s \ominus I$ 
by (auto intro:tminus-left-mono add-mono)
also have  $\bigwedge a. \text{wlp } (p \ a) \ P \ s + \text{wp } (p \ a) \ Q \ s \ominus I = (\text{wlp } (p \ a) \ P \ \&\& \ \text{wp } (p \ a) \ Q) \ s$ 
by (simp add:exp-conj-def pconj-def)
also from sdp  $uP \ uQ$ 
have  $\bigwedge a. a \in S \ s \Longrightarrow \dots a \leq \text{wp } (p \ a) \ (P \ \&\& \ Q) \ s$ 
by (blast)
also have  $\bigwedge a. \dots a = \text{wp } (p \ a) \ (\lambda s. P \ s + Q \ s \ominus I) \ s$ 
by (simp add:exp-conj-def pconj-def)
finally
show  $(\text{wlp } (\text{SetDC } p \ S) \ P \ \&\& \ \text{wp } (\text{SetDC } p \ S) \ Q) \ s \leq \text{wp } (\text{SetDC } p \ S) \ (P \ \&\& \ Q) \ s$ 
unfolding exp-conj-def pconj-def wp-eval
using ne by (blast intro!:cInf-greatest)
qed

```

lemma *sdp-Bind*:

```

 $\llbracket \bigwedge s. \text{sub-distrib-pconj } (p \ (f \ s)) \rrbracket \Longrightarrow \text{sub-distrib-pconj } (\text{Bind } f \ p)$ 
unfolding sub-distrib-pconj-def wp-eval exp-conj-def pconj-def
by (blast)

```

For loops, we again appeal to our transfinite induction principle, this time taking advantage of the simultaneous treatment of both strict and liberal transformers.

lemma *sdp-loop*:

```

fixes  $\text{body}:: 's \text{ prog}$ 
assumes sdp-body: sub-distrib-pconj  $\text{body}$ 
and hwlp: nearly-healthy  $(\text{wlp } \text{body})$ 
and hwp: healthy  $(\text{wp } \text{body})$ 
shows sub-distrib-pconj  $(\text{do } G \longrightarrow \text{body } od)$ 
proof (rule sub-distrib-pconjI, rule loop-induct[OF hwp hwlp])
fix  $P \ Q:: 's \text{ expect}$  and  $S:: ('s \text{ trans} \times 's \text{ trans}) \text{ set}$ 
assume  $uP$ : unitary  $P$  and  $uQ$ : unitary  $Q$ 
and ffst:  $\forall x \in S. \text{feasible } (fst \ x)$ 
and usnd:  $\forall x \in S. \forall Q. \text{unitary } Q \longrightarrow \text{unitary } (\text{snd } x \ Q)$ 
and IH:  $\forall x \in S. \text{snd } x \ P \ \&\& \ \text{fst } x \ Q \Vdash \text{fst } x \ (P \ \&\& \ Q)$ 

```

show $\text{Inf-utrans } (\text{snd } ' S) P \ \&\& \ \text{Sup-trans } (\text{fst } ' S) Q \Vdash$
 $\text{Sup-trans } (\text{fst } ' S) (P \ \&\& \ Q)$
proof(*cases*)
assume $S = \{\}$
thus *?thesis*
by(*simp add:Inf-trans-def Sup-trans-def Inf-utrans-def*
Inf-exp-def Sup-exp-def exp-conj-def)
next
assume $ne: S \neq \{\}$

let $?f s = 1 + \text{Sup-trans } (\text{fst } ' S) (P \ \&\& \ Q) s - \text{Inf-utrans } (\text{snd } ' S) P s$

from ne **obtain** t **where** $tin: t \in \text{fst } ' S$ **by**(*auto*)
from ne **obtain** u **where** $uin: u \in \text{snd } ' S$ **by**(*auto*)

from tin $ffst uP uQ$ **have** $utPQ: \text{unitary } (t (P \ \&\& \ Q))$
by(*auto intro:exp-conj-unitary*)
hence $\bigwedge s. 0 \leq t (P \ \&\& \ Q) s$ **by**(*auto*)
also {
from $ffst tin$ **have** $le: le\text{-utrans } t (\text{Sup-trans } (\text{fst } ' S))$
by(*auto intro:Sup-trans-upper*)
with $uP uQ$ **have** $\bigwedge s. t (P \ \&\& \ Q) s \leq \text{Sup-trans } (\text{fst } ' S) (P \ \&\& \ Q) s$
by(*auto intro:exp-conj-unitary*)
}
finally **have** $nn\text{-rhs}: \bigwedge s. 0 \leq \text{Sup-trans } (\text{fst } ' S) (P \ \&\& \ Q) s .$

have $\bigwedge R. \text{Inf-utrans } (\text{snd } ' S) P \ \&\& \ R \Vdash \text{Sup-trans } (\text{fst } ' S) (P \ \&\& \ Q) \implies R \leq ?f$
proof(*rule contrapos-pp, assumption*)
fix R
assume $\neg R \leq ?f$
then **obtain** s **where** $\neg R s \leq ?f s$ **by**(*auto*)
hence $gt: ?f s < R s$ **by**(*simp*)

from $nn\text{-rhs}$ **have** $g1: 1 \leq 1 + \text{Sup-trans } (\text{fst } ' S) (P \ \&\& \ Q) s$ **by**(*auto*)
hence $\text{Sup-trans } (\text{fst } ' S) (P \ \&\& \ Q) s = \text{Inf-utrans } (\text{snd } ' S) P s .\& \ ?f s$
by(*simp add:pconj-def*)
also **from** $g1$ **have** $\dots = \text{Inf-utrans } (\text{snd } ' S) P s + ?f s - 1$
by(*simp*)
also **from** gt **have** $\dots < \text{Inf-utrans } (\text{snd } ' S) P s + R s - 1$
by(*simp*)
also {
with $g1$ **have** $1 \leq \text{Inf-utrans } (\text{snd } ' S) P s + R s$
by(*simp*)
hence $\text{Inf-utrans } (\text{snd } ' S) P s + R s - 1 = \text{Inf-utrans } (\text{snd } ' S) P s .\& \ R s$
by(*simp add:pconj-def*)
}
finally
have $\neg (\text{Inf-utrans } (\text{snd } ' S) P \ \&\& \ R) s \leq \text{Sup-trans } (\text{fst } ' S) (P \ \&\& \ Q) s$

by(*simp add:exp-conj-def*)
thus $\neg \text{Inf-utrans } (\text{snd } ' S) P \ \&\& \ R \Vdash \text{Sup-trans } (\text{fst } ' S) (P \ \&\& \ Q)$
by(*auto*)
qed

moreover have $\forall t \in \text{fst } ' S. \text{Inf-utrans } (\text{snd } ' S) P \ \&\& \ t \ Q \Vdash \text{Sup-trans } (\text{fst } ' S) (P \ \&\& \ Q)$
proof
fix t **assume** $\text{tin}: t \in \text{fst } ' S$
then obtain x **where** $\text{xin}: x \in S$ **and** $\text{fx}: t = \text{fst } x$ **by**(*auto*)

from xin **have** $\text{snd } x \in \text{snd } ' S$ **by**(*auto*)
with uP **usnd** **have** $\text{Inf-utrans } (\text{snd } ' S) P \Vdash \text{snd } x \ P$
by(*auto intro:le-utransD[OF Inf-utrans-lower]*)
hence $\text{Inf-utrans } (\text{snd } ' S) P \ \&\& \ \text{fst } x \ Q \Vdash \text{snd } x \ P \ \&\& \ \text{fst } x \ Q$
by(*auto intro:entails-frame*)
also from xin **IH** **have** $\dots \Vdash \text{fst } x \ (P \ \&\& \ Q)$
by(*auto*)
also from xin **ffst exp-conj-unitary**[*OF uP uQ*]
have $\dots \Vdash \text{Sup-trans } (\text{fst } ' S) (P \ \&\& \ Q)$
by(*auto intro:le-utransD[OF Sup-trans-upper]*)
finally show $\text{Inf-utrans } (\text{snd } ' S) P \ \&\& \ t \ Q \Vdash \text{Sup-trans } (\text{fst } ' S) (P \ \&\& \ Q)$
by(*simp add:fx*)
qed

ultimately have $\text{bt}: \forall t \in \text{fst } ' S. t \ Q \Vdash ?f$ **by**(*blast*)

have $\text{Sup-trans } (\text{fst } ' S) Q = \text{Sup-exp } \{t \ Q \mid t. t \in \text{fst } ' S\}$
by(*simp add:Sup-trans-def*)
also have $\dots \Vdash ?f$
proof(*rule Sup-exp-least*)
from bt **show** $\forall R \in \{t \ Q \mid t. t \in \text{fst } ' S\}. R \Vdash ?f$ **by**(*blast*)
from ne **obtain** t **where** $\text{tin}: t \in \text{fst } ' S$ **by**(*auto*)
with $\text{ffst } uQ$ **have** $\text{unitary } (t \ Q)$ **by**(*auto*)
hence $\lambda s. 0 \Vdash t \ Q$ **by**(*auto*)
also from tin bt **have** $\dots \Vdash ?f$ **by**(*auto*)
finally show $\text{nneg } (\lambda s. 1 + \text{Sup-trans } (\text{fst } ' S) (P \ \&\& \ Q) \ s - \text{Inf-utrans } (\text{snd } ' S) P \ s)$
by(*auto*)
qed

finally have $\text{Inf-utrans } (\text{snd } ' S) P \ \&\& \ \text{Sup-trans } (\text{fst } ' S) Q \Vdash \text{Inf-utrans } (\text{snd } ' S) P \ \&\& \ ?f$
by(*auto intro:entails-frame*)
also from nn-rhs **have** $\dots \Vdash \text{Sup-trans } (\text{fst } ' S) (P \ \&\& \ Q)$
by(*simp add:exp-conj-def pconj-def*)
finally show $?thesis$.
qed

next
fix $P \ Q::'s \ \text{expect}$ **and** $t \ u::'s \ \text{trans}$

assume uP : unitary P and uQ : unitary Q
and ft : feasible t
and uu : $\bigwedge Q$. unitary $Q \implies$ unitary $(u Q)$
and IH : $u P \ \&\& \ t Q \Vdash t (P \ \&\& \ Q)$
show $wlp (body \ ; \ Embed \ u \ \ll G \gg \oplus \ Skip) P \ \&\&$
 $wp (body \ ; \ Embed \ t \ \ll G \gg \oplus \ Skip) Q \Vdash$
 $wp (body \ ; \ Embed \ t \ \ll G \gg \oplus \ Skip) (P \ \&\& \ Q)$
proof(rule *le-funI*, *simp add:wp-eval exp-conj-def pconj-def*)
fix $s::'s$
have $\ll G \gg s * wlp \ body \ (u P) \ s + (I - \ll G \gg s) * P \ s +$
 $(\ll G \gg s * wp \ body \ (t Q) \ s + (I - \ll G \gg s) * Q \ s) \ominus I =$
 $(\ll G \gg s * wlp \ body \ (u P) \ s + \ll G \gg s * wp \ body \ (t Q) \ s) +$
 $((I - \ll G \gg s) * P \ s + (I - \ll G \gg s) * Q \ s) \ominus (\ll G \gg s + (I - \ll G \gg s))$
by(*simp add:ac-simps*)
also have $\dots \leq$
 $(\ll G \gg s * wlp \ body \ (u P) \ s + \ll G \gg s * wp \ body \ (t Q) \ s \ominus \ll G \gg s) +$
 $((I - \ll G \gg s) * P \ s + (I - \ll G \gg s) * Q \ s \ominus (I - \ll G \gg s))$
by(rule *tminus-add-mono*)
also have $\dots =$
 $\ll G \gg s * (wlp \ body \ (u P) \ s + wp \ body \ (t Q) \ s \ominus I) +$
 $(I - \ll G \gg s) * (P \ s + Q \ s \ominus I)$
by(*simp add:tminus-left-distrib distrib-left*)
also {
from $uP \ uQ \ ft \ uu$
have $wlp \ body \ (u P) \ \&\& \ wp \ body \ (t Q) \ \Vdash \ wp \ body \ (u P \ \&\& \ t Q)$
by(*auto intro:sub-distrib-pconjD[OF sdp-body]*)
also from IH *unitary-sound[OF uP] unitary-sound[OF uQ] ft*
unitary-sound[OF uu[OF uP]]
have $\dots \leq wp \ body \ (t (P \ \&\& \ Q))$
by(*blast intro!:mono-transD[OF healthy-monoD, OF hwp] exp-conj-sound*)
finally have $wlp \ body \ (u P) \ s + wp \ body \ (t Q) \ s \ominus I \leq$
 $wp \ body \ (t (\lambda s. P \ s + Q \ s \ominus I)) \ s$
by(*auto simp:exp-conj-def pconj-def*)
hence $\ll G \gg s * (wlp \ body \ (u P) \ s + wp \ body \ (t Q) \ s \ominus I) +$
 $(I - \ll G \gg s) * (P \ s + Q \ s \ominus I) \leq$
 $\ll G \gg s * wp \ body \ (t (\lambda s. P \ s + Q \ s \ominus I)) \ s +$
 $(I - \ll G \gg s) * (P \ s + Q \ s \ominus I)$
by(*auto intro:add-right-mono mult-left-mono*)
}
finally
show $\ll G \gg s * wlp \ body \ (u P) \ s + (I - \ll G \gg s) * P \ s +$
 $(\ll G \gg s * wp \ body \ (t Q) \ s + (I - \ll G \gg s) * Q \ s) \ominus I \leq$
 $\ll G \gg s * wp \ body \ (t (\lambda s. P \ s + Q \ s \ominus I)) \ s +$
 $(I - \ll G \gg s) * (P \ s + Q \ s \ominus I)$.
qed
next
fix $P \ Q::'s \ expect$ and $t' \ u \ u'::'s \ trans$
assume *unitary P unitary Q*
equiv-trans t t' equiv-utrans u u'

$u P \&\& t Q \Vdash t (P \&\& Q)$
thus $u' P \&\& t' Q \Vdash t' (P \&\& Q)$
by(*simp add:equiv-transD unitary-sound equiv-utransD exp-conj-unitary*)
qed

lemmas *sdp-intros* =
sdp-Abort sdp-Skip sdp-Apply
sdp-Seq sdp-DC sdp-PC
sdp-SetPC sdp-SetDC sdp-Embed
sdp-repeat sdp-Bind sdp-loop

4.7.3 The Well-Defined Predicate.

definition

well-def :: 's prog \Rightarrow bool

where

$well-def\ prog \equiv healthy\ (wp\ prog) \wedge nearly_healthy\ (wlp\ prog)$
 $\wedge wp_under_wlp\ prog \wedge sub_distrib_pconj\ prog$
 $\wedge sublinear\ (wp\ prog) \wedge bd_cts\ (wp\ prog)$

lemma *well-def*[intro]:

$\llbracket healthy\ (wp\ prog); nearly_healthy\ (wlp\ prog);$
 $wp_under_wlp\ prog; sub_distrib_pconj\ prog; sublinear\ (wp\ prog);$
 $bd_cts\ (wp\ prog) \rrbracket \Longrightarrow$
 $well_def\ prog$

unfolding *well-def-def* **by**(*simp*)

lemma *well-def-wp-healthy*[dest]:

$well_def\ prog \Longrightarrow healthy\ (wp\ prog)$

unfolding *well-def-def* **by**(*simp*)

lemma *well-def-wlp-nearly-healthy*[dest]:

$well_def\ prog \Longrightarrow nearly_healthy\ (wlp\ prog)$

unfolding *well-def-def* **by**(*simp*)

lemma *well-def-wp-under*[dest]:

$well_def\ prog \Longrightarrow wp_under_wlp\ prog$

unfolding *well-def-def* **by**(*simp*)

lemma *well-def-sdp*[dest]:

$well_def\ prog \Longrightarrow sub_distrib_pconj\ prog$

unfolding *well-def-def* **by**(*simp*)

lemma *well-def-wp-sublinear*[dest]:

$well_def\ prog \Longrightarrow sublinear\ (wp\ prog)$

unfolding *well-def-def* **by**(*simp*)

lemma *well-def-wp-cts*[dest]:

$well_def\ prog \Longrightarrow bd_cts\ (wp\ prog)$

unfolding *well-def-def* **by**(*simp*)

lemmas *wd-dests* =

well-def-wp-healthy well-def-wlp-nearly-healthy
well-def-wp-under well-def-sdp
well-def-wp-sublinear well-def-wp-cts

lemma *wd-Abort*:

well-def Abort
by(*blast intro:healthy-wp-Abort nearly-healthy-wlp-Abort*
wp-under-wlp-Abort sdp-Abort sublinear-wp-Abort
cts-wp-Abort)

lemma *wd-Skip*:

well-def Skip
by(*blast intro:healthy-wp-Skip nearly-healthy-wlp-Skip*
wp-under-wlp-Skip sdp-Skip sublinear-wp-Skip
cts-wp-Skip)

lemma *wd-Apply*:

well-def (Apply f)
by(*blast intro:healthy-wp-Apply nearly-healthy-wlp-Apply*
wp-under-wlp-Apply sdp-Apply sublinear-wp-Apply
cts-wp-Apply)

lemma *wd-Seq*:

$\llbracket \text{well-def } a; \text{well-def } b \rrbracket \implies \text{well-def } (a ;; b)$
by(*blast intro:healthy-wp-Seq nearly-healthy-wlp-Seq*
wp-under-wlp-Seq sdp-Seq sublinear-wp-Seq
cts-wp-Seq)

lemma *wd-PC*:

$\llbracket \text{well-def } a; \text{well-def } b; \text{unitary } P \rrbracket \implies \text{well-def } (a \oplus b)$
by(*blast intro:healthy-wp-PC nearly-healthy-wlp-PC*
wp-under-wlp-PC sdp-PC sublinear-wp-PC
cts-wp-PC)

lemma *wd-DC*:

$\llbracket \text{well-def } a; \text{well-def } b \rrbracket \implies \text{well-def } (a \sqcap b)$
by(*blast intro:healthy-wp-DC nearly-healthy-wlp-DC*
wp-under-wlp-DC sdp-DC sublinear-wp-DC
cts-wp-DC)

lemma *wd-SetDC*:

$\llbracket \bigwedge x s. x \in S s \implies \text{well-def } (a x); \bigwedge s. S s \neq \{\};$
 $\bigwedge s. \text{finite } (S s) \rrbracket \implies \text{well-def } (\text{SetDC } a S)$

by (*simp add: cts-wp-SetDC ex-in-conv healthy-intros(17) healthy-intros(18) sdp-intros(8)*
sublinear-intros(8) well-def-def wp-under-wlp-intros(8))

lemma *wd-SetPC*:

$\llbracket \bigwedge x s. x \in (\text{supp } (p s)) \implies \text{well-def } (a x); \bigwedge s. \text{unitary } (p s); \bigwedge s. \text{finite } (\text{supp } (p s));$
 $\bigwedge s. \text{sum } (p s) (\text{supp } (p s)) \leq 1 \rrbracket \implies \text{well-def } (\text{SetPC } a p)$
by(*iprover intro!:**well-defI healthy-wp-SetPC nearly-healthy-wlp-SetPC*
wp-under-wlp-SetPC sdp-SetPC sublinear-wp-SetPC cts-wp-SetPC
dest:wd-dests unitary-sound sound-nneg[OF unitary-sound] nnegD)

lemma *wd-Embed*:

fixes *t::'s trans*

assumes *ht: healthy t and st: sublinear t and ct: bd-cts t*

shows *well-def (Embed t)*

proof(*intro well-defI*)

from *ht show healthy (wp (Embed t)) nearly-healthy (wlp (Embed t))*

by(*simp add:wp-def wlp-def Embed-def healthy-nearly-healthy*)**+**

from *st show sublinear (wp (Embed t))* **by**(*simp add:wp-def Embed-def*)

show *wp-under-wlp (Embed t)* **by**(*simp add:wp-under-wlp-def wp-eval*)

show *sub-distrib-pconj (Embed t)*

by(*rule sub-distrib-pconjI,*

auto intro:le-funI[OF sublinearD[OF st, where a=1 and b=1 and c=1, simplified]]

simp:exp-conj-def pconj-def wp-def wlp-def Embed-def)

from *ct show bd-cts (wp (Embed t))*

by(*simp add:wp-def Embed-def*)

qed

lemma *wd-repeat*:

well-def a \implies well-def (repeat n a)

by(*blast intro:healthy-wp-repeat nearly-healthy-wlp-repeat*

wp-under-wlp-repeat sdp-repeat sublinear-wp-repeat cts-wp-repeat)

lemma *wd-Bind*:

$\llbracket \bigwedge s. \text{well-def } (a (f s)) \rrbracket \implies \text{well-def } (\text{Bind } f a)$

by(*blast intro:healthy-wp-Bind nearly-healthy-wlp-Bind*

wp-under-wlp-Bind sdp-Bind sublinear-wp-Bind cts-wp-Bind)

lemma *wd-loop*:

well-def body \implies well-def (do G \longrightarrow body od)

by(*blast intro:healthy-wp-loop nearly-healthy-wlp-loop*

wp-under-wlp-loop sdp-loop sublinear-wp-loop cts-wp-loop)

lemmas *wd-intros =*

wd-Abort wd-Skip wd-Apply

wd-Embed wd-Seq wd-PC

wd-DC wd-SetPC wd-SetDC

wd-Bind wd-repeat wd-loop

end

4.8 The Loop Rules

theory *Loops* **imports** *WellDefined* **begin**

Given a well-defined body, we can annotate a loop using an invariant, just as in the classical setting.

4.8.1 Liberal and Strict Invariants.

A probabilistic invariant generalises a boolean one: it *entails* itself, given the loop guard.

definition

$wp\text{-}inv :: ('s \Rightarrow bool) \Rightarrow 's \text{ prog} \Rightarrow ('s \Rightarrow real) \Rightarrow bool$

where

$wp\text{-}inv\ G\ body\ I \longleftrightarrow (\forall s. \langle\langle G \rangle\rangle\ s * I\ s \leq wp\ body\ I\ s)$

lemma *wp-invI*:

$\bigwedge I. (\bigwedge s. \langle\langle G \rangle\rangle\ s * I\ s \leq wp\ body\ I\ s) \Longrightarrow wp\text{-}inv\ G\ body\ I$
by(*simp add:wp-inv-def*)

definition

$wlp\text{-}inv :: ('s \Rightarrow bool) \Rightarrow 's \text{ prog} \Rightarrow ('s \Rightarrow real) \Rightarrow bool$

where

$wlp\text{-}inv\ G\ body\ I \longleftrightarrow (\forall s. \langle\langle G \rangle\rangle\ s * I\ s \leq wlp\ body\ I\ s)$

lemma *wlp-invI*:

$\bigwedge I. (\bigwedge s. \langle\langle G \rangle\rangle\ s * I\ s \leq wlp\ body\ I\ s) \Longrightarrow wlp\text{-}inv\ G\ body\ I$
by(*simp add:wlp-inv-def*)

lemma *wlp-invD*:

$wlp\text{-}inv\ G\ body\ I \Longrightarrow \langle\langle G \rangle\rangle\ s * I\ s \leq wlp\ body\ I\ s$
by(*simp add:wlp-inv-def*)

For standard invariants, the multiplication reduces to conjunction.

lemma *wp-inv-stdD*:

assumes *inv*: $wp\text{-}inv\ G\ body\ \langle I \rangle$

and *hb*: *healthy* (*wp body*)

shows $\langle\langle G \rangle\rangle\ \&\&\ \langle I \rangle \Vdash wp\ body\ \langle I \rangle$

proof(*rule le-funI*)

fix *s*

show $\langle\langle G \rangle\rangle\ \&\&\ \langle I \rangle\ s \leq wp\ body\ \langle I \rangle\ s$

proof(*cases G s*)

case *False*

with *hb* **show** *?thesis*

by(*auto simp:exp-conj-def*)

next

case *True*

hence $\langle\langle G \rangle\rangle\ \&\&\ \langle I \rangle\ s = \langle\langle G \rangle\rangle\ s * \langle I \rangle\ s$

```

    by(simp add:exp-conj-def)
  also from inv have «G» s * «I» s ≤ wp body «I» s
    by(simp add:wp-inv-def)
  finally show ?thesis .
qed
qed

```

4.8.2 Partial Correctness

Partial correctness for loops[McIver and Morgan, 2004, Lemma 7.2.2, §7, p. 185].

```

lemma wlp-Loop:
  assumes wd: well-def body
    and ul: unitary I
    and inv: wlp-inv G body I
  shows I ≤ wlp do G → body od (λs. «N G» s * I s)
    (is I ≤ wlp do G → body od ?P)
proof -
  let ?f Q s = «G» s * wlp body Q s + «N G» s * ?P s
  have I ⊢ gfp-exp ?f
  proof(rule gfp-exp-upperbound[OF - ul])
    have I = (λs. «G» s + «N G» s * I s) by(simp add:negate-embed)
    also have ... = (λs. «G» s * I s + «N G» s * I s)
      by(simp add:algebra-simps)
    also have ... = (λs. «G» s * («G» s * I s) + «N G» s * («N G» s * I s))
      by(simp add:embed-bool-idem algebra-simps)
    also have ... ⊢ (λs. «G» s * wlp body I s + «N G» s * («N G» s * I s))
      using inv by(auto dest:wlp-invD intro:add-mono mult-left-mono)
    finally show I ⊢ (λs. «G» s * wlp body I s + «N G» s * («N G» s * I s)) .
  qed
  also from ul well-def-wlp-nearly-healthy[OF wd] have ... = wlp do G → body od ?P
    by(auto intro!:wlp-LoopI[symmetric] unitary-intros)
  finally show ?thesis .
qed

```

4.8.3 Total Correctness

The first total correctness lemma for loops which terminate with probability 1[McIver and Morgan, 2004, Lemma 7.3.1, §7, p. 186].

```

lemma wp-Loop:
  assumes wd: well-def body
    and inv: wlp-inv G body I
    and unit: unitary I
  shows I && wp (do G → body od) (λs. I) ⊢ wp (do G → body od) (λs. «N G» s * I s)
    (is I && ?T ⊢ wp ?loop ?X)
proof -

```

We first appeal to the *liberal* loop rule:

from *assms* **have** $I \&\& ?T \Vdash wlp ?loop ?X \&\& ?T$
by(*blast intro:exp-conj-mono-left wlp-Loop*)

Next, by sub-conjunctivity:

```

also {
  from wd have sdp-loop: sub-distrib-pconj (do G  $\longrightarrow$  body od)
  by(blast intro:sdp-intros)

  from wd unit have  $wlp ?loop ?X \&\& ?T \Vdash wp ?loop (?X \&\& (\lambda s. I))$ 
  by(blast intro:sub-distrib-pconjD sdp-intros unitary-intros)
}

```

Finally, the conjunction collapses:

finally show *?thesis*
by(*simp add:exp-conj-1-right sound-intros sound-nneg unit unitary-sound*)
qed

4.8.4 Unfolding

lemma *wp-loop-unfold*:

```

fixes body :: 's prog
assumes sP: sound P
  and h: healthy (wp body)
shows  $wp (do G \longrightarrow body od) P =$ 
  ( $\lambda s. \ll \mathcal{N} G \gg s * P s + \ll G \gg s * wp body (wp (do G \longrightarrow body od) P) s$ )
proof (simp only: wp-eval)
let  $?X t = wp (body ;; Embed t \ll G \gg \oplus Skip)$ 
have equiv-trans (lfp-trans ?X)
  ( $wp (body ;; Embed (lfp-trans ?X) \ll G \gg \oplus Skip)$ )
proof (intro lfp-trans-unfold)
fix t::'s trans and P::'s expect
assume st:  $\bigwedge Q. sound Q \implies sound (t Q)$ 
  and sP: sound P
with h show  $sound (?X t P)$ 
  by(rule wp-loop-step-sound)
next
fix t u::'s trans
assume  $le-trans t u (\bigwedge P. sound P \implies sound (t P))$ 
  ( $\bigwedge P. sound P \implies sound (u P)$ )
with h show  $le-trans (wp (body ;; Embed t \ll G \gg \oplus Skip))$ 
  ( $wp (body ;; Embed u \ll G \gg \oplus Skip)$ )
  by(iprover intro:wp-loop-step-mono)
next
let  $?v = \lambda P s. bound-of P$ 
from h show  $le-trans (wp (body ;; Embed ?v \ll G \gg \oplus Skip)) ?v$ 
  by(intro le-transI, simp add:wp-eval lfp-loop-fp[unfolding negate-embed])
fix P::'s expect
assume sound P thus  $sound (?v P)$  by(auto)
qed

```

also have *equiv-trans* ...

$(\lambda P s. \langle \mathcal{N} G \rangle s * P s + \langle G \rangle s * wp \text{ body } (wp (\text{Embed } (lfp\text{-trans } ?X)) P) s)$

by(*rule equiv-transI*, *simp add:wp-eval algebra-simps negate-embed*)

finally show *lfp-trans ?X P =*

$(\lambda s. \langle \mathcal{N} G \rangle s * P s + \langle G \rangle s * wp \text{ body } (lfp\text{-trans } ?X P) s)$

using *sP unfolding wp-eval by(blast)*

qed

lemma *wp-loop-nguard*:

$\llbracket \text{healthy } (wp \text{ body}); \text{sound } P; \neg G s \rrbracket \implies wp \text{ do } G \longrightarrow \text{body od } P s = P s$

by(*subst wp-loop-unfold, simp-all*)

lemma *wp-loop-guard*:

$\llbracket \text{healthy } (wp \text{ body}); \text{sound } P; G s \rrbracket \implies$

$wp \text{ do } G \longrightarrow \text{body od } P s = wp (\text{body} ;; \text{do } G \longrightarrow \text{body od}) P s$

by(*subst wp-loop-unfold, simp-all add:wp-eval*)

end

4.9 The Algebra of pGCL

theory *Algebra* **imports** *WellDefined* **begin**

Programs in pGCL have a rich algebraic structure, largely mirroring that for GCL. We show that programs form a lattice under refinement, with $a \sqcap b$ and $a \sqcup b$ as the meet and join operators, respectively. We also take advantage of the algebraic structure to establish a framework for the modular decomposition of proofs.

4.9.1 Program Refinement

Refinement in pGCL relates to refinement in GCL exactly as probabilistic entailment relates to implication. It turns out to have a very similar algebra, the rules of which we establish shortly.

definition

refines :: $'s \text{ prog} \Rightarrow 's \text{ prog} \Rightarrow \text{bool}$ (**infix** \sqsubseteq 70)

where

$\text{prog} \sqsubseteq \text{prog}' \equiv \forall P. \text{sound } P \longrightarrow wp \text{ prog } P \Vdash wp \text{ prog}' P$

lemma *refinesI*[*intro*]:

$\llbracket \bigwedge P. \text{sound } P \implies wp \text{ prog } P \Vdash wp \text{ prog}' P \rrbracket \implies \text{prog} \sqsubseteq \text{prog}'$

unfolding *refines-def* **by**(*simp*)

lemma *refinesD*[*dest*]:

$\llbracket \text{prog} \sqsubseteq \text{prog}'; \text{sound } P \rrbracket \implies wp \text{ prog } P \Vdash wp \text{ prog}' P$

unfolding *refines-def* **by**(*simp*)

The equivalence relation below will turn out to be that induced by refinement. It is also the application of *equiv-trans* to the weakest precondition.

definition

$$pequiv :: 's\ prog \Rightarrow 's\ prog \Rightarrow bool \text{ (infix } \simeq 70)$$
where

$$prog \simeq prog' \equiv \forall P. \text{sound } P \longrightarrow wp\ prog\ P = wp\ prog'\ P$$
lemma *pequivI*[*intro*]:
$$\llbracket \bigwedge P. \text{sound } P \Longrightarrow wp\ prog\ P = wp\ prog'\ P \rrbracket \Longrightarrow prog \simeq prog'$$

unfolding *pequiv-def* **by**(*simp*)

lemma *pequivD*[*dest, simp*]:
$$\llbracket prog \simeq prog'; \text{sound } P \rrbracket \Longrightarrow wp\ prog\ P = wp\ prog'\ P$$

unfolding *pequiv-def* **by**(*simp*)

lemma *pequiv-equiv-trans*:
$$a \simeq b \longleftrightarrow equiv\text{-trans } (wp\ a) (wp\ b)$$

by(*auto*)

4.9.2 Simple Identities

The following identities involve only the primitive operations as defined in [Section 4.1.1](#), and refinement as defined above.

Laws following from the basic arithmetic of the operators separately**lemma** *DC-comm*[*ac-simps*]:
$$a \sqcap b = b \sqcap a$$

unfolding *DC-def* **by**(*simp add:ac-simps*)

lemma *DC-assoc*[*ac-simps*]:
$$a \sqcap (b \sqcap c) = (a \sqcap b) \sqcap c$$

unfolding *DC-def* **by**(*simp add:ac-simps*)

lemma *DC-idem*:
$$a \sqcap a = a$$

unfolding *DC-def* **by**(*simp*)

lemma *AC-comm*[*ac-simps*]:
$$a \sqcup b = b \sqcup a$$

unfolding *AC-def* **by**(*simp add:ac-simps*)

lemma *AC-assoc*[*ac-simps*]:
$$a \sqcup (b \sqcup c) = (a \sqcup b) \sqcup c$$

unfolding *AC-def* **by**(*simp add:ac-simps*)

lemma *AC-idem*:
$$a \sqcup a = a$$

unfolding *AC-def* **by**(*simp*)

lemma *PC-quasi-comm*:

$a p \oplus b = b (\lambda s. I - p s) \oplus a$
unfolding *PC-def* **by**(*simp add:algebra-simps*)

lemma *PC-idem*:

$a p \oplus a = a$
unfolding *PC-def* **by**(*simp add:algebra-simps*)

lemma *Seq-assoc*[*ac-simps*]:

$A ;; (B ;; C) = A ;; B ;; C$
by(*simp add:Seq-def o-def*)

lemma *Abort-refines*[*intro*]:

$\text{well-def } a \implies \text{Abort} \sqsubseteq a$
by(*rule refinesI, unfold wp-eval, auto dest!:well-def-wp-healthy*)

Laws relating demonic choice and refinement

lemma *left-refines-DC*:

$(a \sqcap b) \sqsubseteq a$
by(*auto intro!:refinesI simp:wp-eval*)

lemma *right-refines-DC*:

$(a \sqcap b) \sqsubseteq b$
by(*auto intro!:refinesI simp:wp-eval*)

lemma *DC-refines*:

fixes $a::'s \text{ prog}$ **and** b **and** c
assumes $\text{rab}: a \sqsubseteq b$ **and** $\text{rac}: a \sqsubseteq c$
shows $a \sqsubseteq (b \sqcap c)$

proof

fix $P::'s \implies \text{real}$ **assume** $sP: \text{sound } P$
with *assms* **have** $\text{wp } a P \Vdash \text{wp } b P$ **and** $\text{wp } a P \Vdash \text{wp } c P$
by(*auto dest:refinesD*)
thus $\text{wp } a P \Vdash \text{wp } (b \sqcap c) P$
by(*auto simp:wp-eval intro:min.boundedI*)

qed

lemma *DC-mono*:

fixes $a::'s \text{ prog}$
assumes $\text{rab}: a \sqsubseteq b$ **and** $\text{rcd}: c \sqsubseteq d$
shows $(a \sqcap c) \sqsubseteq (b \sqcap d)$
proof(*rule refinesI, unfold wp-eval, rule le-funI*)
fix $P::'s \implies \text{real}$ **and** $s::'s$
assume $sP: \text{sound } P$
with *assms* **have** $\text{wp } a P s \leq \text{wp } b P s$ **and** $\text{wp } c P s \leq \text{wp } d P s$
by(*auto*)
thus $\min (\text{wp } a P s) (\text{wp } c P s) \leq \min (\text{wp } b P s) (\text{wp } d P s)$
by(*auto*)

qed

Laws relating angelic choice and refinement**lemma** *left-refines-AC*: $a \sqsubseteq (a \sqcup b)$ **by**(*auto intro!:refinesI simp:wp-eval*)**lemma** *right-refines-AC*: $b \sqsubseteq (a \sqcup b)$ **by**(*auto intro!:refinesI simp:wp-eval*)**lemma** *AC-refines*:**fixes** $a::'s \text{ prog}$ **and** b **and** c **assumes** $rac: a \sqsubseteq c$ **and** $rbc: b \sqsubseteq c$ **shows** $(a \sqcup b) \sqsubseteq c$ **proof****fix** $P::'s \Rightarrow \text{real}$ **assume** $sP: \text{sound } P$ **with** $assms$ **have** $\bigwedge s. wp \ a \ P \ s \leq wp \ c \ P \ s$ **and** $\bigwedge s. wp \ b \ P \ s \leq wp \ c \ P \ s$ **by**(*auto dest:refinesD*)**thus** $wp \ (a \sqcup b) \ P \ \vdash \ wp \ c \ P$ **unfolding** *wp-eval* **by**(*auto*)**qed****lemma** *AC-mono*:**fixes** $a::'s \text{ prog}$ **assumes** $rab: a \sqsubseteq b$ **and** $rcd: c \sqsubseteq d$ **shows** $(a \sqcup c) \sqsubseteq (b \sqcup d)$ **proof**(*rule refinesI, unfold wp-eval, rule le-funI*)**fix** $P::'s \Rightarrow \text{real}$ **and** $s::'s$ **assume** $sP: \text{sound } P$ **with** $assms$ **have** $wp \ a \ P \ s \leq wp \ b \ P \ s$ **and** $wp \ c \ P \ s \leq wp \ d \ P \ s$ **by**(*auto*)**thus** $\max (wp \ a \ P \ s) (wp \ c \ P \ s) \leq \max (wp \ b \ P \ s) (wp \ d \ P \ s)$ **by**(*auto*)**qed****Laws depending on the arithmetic of $a_p \oplus b$ and $a \sqcap b$ together****lemma** *DC-refines-PC*:**assumes** $unit: \text{unitary } p$ **shows** $(a \sqcap b) \sqsubseteq (a_p \oplus b)$ **proof**(*rule refinesI, unfold wp-eval, rule le-funI*)**fix** s **and** $P::'a \Rightarrow \text{real}$ **assume** $sound: \text{sound } P$ **from** $unit$ **have** $nn-p: 0 \leq p \ s$ **by**(*blast*)**from** $unit$ **have** $p \ s \leq 1$ **by**(*blast*)**hence** $nn-mp: 0 \leq 1 - p \ s$ **by**(*simp*)**show** $\min (wp \ a \ P \ s) (wp \ b \ P \ s) \leq p \ s * wp \ a \ P \ s + (1 - p \ s) * wp \ b \ P \ s$ **proof**(*cases wp \ a \ P \ s \leq wp \ b \ P \ s,**simp-all add:min.absorb1 min.absorb2*)**case** *True* **note** $le = \text{this}$

```

have  $wp\ a\ P\ s = (p\ s + (1 - p\ s)) * wp\ a\ P\ s$  by(simp)
also have  $\dots = p\ s * wp\ a\ P\ s + (1 - p\ s) * wp\ a\ P\ s$ 
by(simp only: distrib-right)
also {
  from le and nn-np have  $(1 - p\ s) * wp\ a\ P\ s \leq (1 - p\ s) * wp\ b\ P\ s$ 
  by(rule mult-left-mono)
  hence  $p\ s * wp\ a\ P\ s + (1 - p\ s) * wp\ a\ P\ s \leq$ 
   $p\ s * wp\ a\ P\ s + (1 - p\ s) * wp\ b\ P\ s$ 
  by(rule add-left-mono)
}
finally show  $wp\ a\ P\ s \leq p\ s * wp\ a\ P\ s + (1 - p\ s) * wp\ b\ P\ s .$ 
next
case False
then have le:  $wp\ b\ P\ s \leq wp\ a\ P\ s$  by(simp)
have  $wp\ b\ P\ s = (p\ s + (1 - p\ s)) * wp\ b\ P\ s$  by(simp)
also have  $\dots = p\ s * wp\ b\ P\ s + (1 - p\ s) * wp\ b\ P\ s$ 
by(simp only:distrib-right)
also {
  from le and nn-p have  $p\ s * wp\ b\ P\ s \leq p\ s * wp\ a\ P\ s$ 
  by(rule mult-left-mono)
  hence  $p\ s * wp\ b\ P\ s + (1 - p\ s) * wp\ b\ P\ s \leq$ 
   $p\ s * wp\ a\ P\ s + (1 - p\ s) * wp\ b\ P\ s$ 
  by(rule add-right-mono)
}
finally show  $wp\ b\ P\ s \leq p\ s * wp\ a\ P\ s + (1 - p\ s) * wp\ b\ P\ s .$ 
qed
qed

```

Laws depending on the arithmetic of $a \oplus b$ and $a \sqcup b$ together

lemma *PC-refines-AC*:

assumes *unit*: *unitary p*

shows $(a \oplus b) \sqsubseteq (a \sqcup b)$

proof(*rule refinesI, unfold wp-eval, rule le-funI*)

fix *s* **and** *P*::'*a* \Rightarrow *real assume* *sound*: *sound P*

from *unit* **have** $p\ s \leq 1$ **by**(*blast*)

hence *nn-np*: $0 \leq 1 - p\ s$ **by**(*simp*)

show $p\ s * wp\ a\ P\ s + (1 - p\ s) * wp\ b\ P\ s \leq$
 $\max (wp\ a\ P\ s) (wp\ b\ P\ s)$

proof(*cases wp\ a\ P\ s \leq wp\ b\ P\ s*)

case *True* **note** *leab = this*

with *unit nn-np*

have $p\ s * wp\ a\ P\ s + (1 - p\ s) * wp\ b\ P\ s \leq$
 $p\ s * wp\ b\ P\ s + (1 - p\ s) * wp\ b\ P\ s$

by(*auto intro:add-mono mult-left-mono*)

also have $\dots = wp\ b\ P\ s$

by(*auto simp:field-simps*)

```

also from leab
have ... =  $\max (wp\ a\ P\ s)\ (wp\ b\ P\ s)$ 
by(auto)
finally show ?thesis .
next
case False note leba = this
with unit nn-mp
have  $p\ s * wp\ a\ P\ s + (1 - p\ s) * wp\ b\ P\ s \leq$ 
 $p\ s * wp\ a\ P\ s + (1 - p\ s) * wp\ a\ P\ s$ 
by(auto intro:add-mono mult-left-mono)
also have ... =  $wp\ a\ P\ s$ 
by(auto simp:field-simps)
also from leba
have ... =  $\max (wp\ a\ P\ s)\ (wp\ b\ P\ s)$ 
by(auto)
finally show ?thesis .
qed
qed

```

Laws depending on the arithmetic of $a \sqcup b$ and $a \sqcap b$ together

```

lemma DC-refines-AC:
 $(a \sqcap b) \sqsubseteq (a \sqcup b)$ 
by(auto intro!:refinesI simp:wp-eval)

```

Laws Involving Refinement and Equivalence

```

lemma pr-trans[trans]:
fixes A::'a prog
assumes prAB:  $A \sqsubseteq B$ 
and prBC:  $B \sqsubseteq C$ 
shows  $A \sqsubseteq C$ 
proof
fix P::'a  $\Rightarrow$  real assume sP: sound P
with prAB have  $wp\ A\ P \Vdash wp\ B\ P$  by(blast)
also from sP and prBC have ...  $\Vdash wp\ C\ P$  by(blast)
finally show  $wp\ A\ P \Vdash \dots$  .
qed

```

```

lemma pequiv-refl[intro!,simp]:
 $a \simeq a$ 
by(auto)

```

```

lemma pequiv-comm[ac-simps]:
 $a \simeq b \longleftrightarrow b \simeq a$ 
unfolding pequiv-def
by(rule iffI, safe, simp-all)

```

```

lemma pequiv-pr[dest]:
 $a \simeq b \implies a \sqsubseteq b$ 

```

by(*auto*)

lemma *pequiv-trans*[*intro,trans*]:
 $\llbracket a \simeq b; b \simeq c \rrbracket \Longrightarrow a \simeq c$
unfolding *pequiv-def* **by**(*auto intro!:order-trans*)

lemma *pequiv-pr-trans*[*intro,trans*]:
 $\llbracket a \simeq b; b \sqsubseteq c \rrbracket \Longrightarrow a \sqsubseteq c$
unfolding *pequiv-def refines-def* **by**(*simp*)

lemma *pr-pequiv-trans*[*intro,trans*]:
 $\llbracket a \sqsubseteq b; b \simeq c \rrbracket \Longrightarrow a \sqsubseteq c$
unfolding *pequiv-def refines-def* **by**(*simp*)

Refinement induces equivalence by antisymmetry:

lemma *pequiv-antisym*:
 $\llbracket a \sqsubseteq b; b \sqsubseteq a \rrbracket \Longrightarrow a \simeq b$
by(*auto intro:antisym*)

lemma *pequiv-DC*:
 $\llbracket a \simeq c; b \simeq d \rrbracket \Longrightarrow (a \sqcap b) \simeq (c \sqcap d)$
by(*auto intro!:DC-mono pequiv-antisym simp:ac-simps*)

lemma *pequiv-AC*:
 $\llbracket a \simeq c; b \simeq d \rrbracket \Longrightarrow (a \sqcup b) \simeq (c \sqcup d)$
by(*auto intro!:AC-mono pequiv-antisym simp:ac-simps*)

4.9.3 Deterministic Programs are Maximal

Any sub-additive refinement of a deterministic program is in fact an equivalence. Deterministic programs are thus maximal (under the refinement order) among sub-additive programs.

lemma *refines-determ*:
fixes *a::'s prog*
assumes *da: determ (wp a)*
and *wa: well-def a*
and *wb: well-def b*
and *dr: a \sqsubseteq b*
shows $a \simeq b$

Proof by contradiction.

proof(*rule pequivI, rule contrapos-pp*)
from *wb* **have** *feasible (wp b)* **by**(*auto*)
with *wb* **have** *sab: sub-add (wp b)*
by(*auto dest: sublinear-subadd[OF well-def-wp-sublinear]*)
fix *P::'s \Rightarrow real* **assume** *sP: sound P*

Assume that *a* and *b* are not equivalent:

assume $ne: wp\ a\ P \neq wp\ b\ P$

Find a point at which they differ. As $a \sqsubseteq b$, $wp\ b\ P\ s$ must be strictly greater than $wp\ a\ P\ s$ here:

hence $\exists s. wp\ a\ P\ s < wp\ b\ P\ s$
proof(*rule contrapos-np*)
assume $\neg(\exists s. wp\ a\ P\ s < wp\ b\ P\ s)$
hence $\forall s. wp\ b\ P\ s \leq wp\ a\ P\ s$ **by**(*auto simp:not-less*)
hence $wp\ b\ P \Vdash wp\ a\ P$ **by**(*auto*)
moreover from $sP\ dr$ **have** $wp\ a\ P \Vdash wp\ b\ P$ **by**(*auto*)
ultimately show $wp\ a\ P = wp\ b\ P$ **by**(*auto*)
qed
then obtain s **where** $less: wp\ a\ P\ s < wp\ b\ P\ s$ **by**(*blast*)

Take a carefully constructed expectation:

let $?Pc = \lambda s. bound-of\ P - P\ s$
have $sPc: sound\ ?Pc$
proof(*rule soundI*)
from sP **have** $\bigwedge s. 0 \leq P\ s$ **by**(*auto*)
hence $\bigwedge s. ?Pc\ s \leq bound-of\ P$ **by**(*auto*)
thus bounded $?Pc$ **by**(*blast*)
from sP **have** $\bigwedge s. P\ s \leq bound-of\ P$ **by**(*auto*)
hence $\bigwedge s. 0 \leq ?Pc\ s$
by *auto*
thus nneg $?Pc$ **by**(*auto*)
qed

We then show that $wp\ b$ violates feasibility, and thus healthiness.

from sP **have** $0 \leq bound-of\ P$ **by**(*auto*)
with da **have** $bound-of\ P = wp\ a\ (\lambda s. bound-of\ P)\ s$
by(*simp add:maximalD determ-maximalD*)
also have $\dots = wp\ a\ (\lambda s. ?Pc\ s + P\ s)\ s$
by(*simp*)
also from $da\ sP\ sPc$ **have** $\dots = wp\ a\ ?Pc\ s + wp\ a\ P\ s$
by(*subst additiveD[OF determ-additiveD], simp-all add:sP sPc*)
also from $sPc\ dr$ **have** $\dots \leq wp\ b\ ?Pc\ s + wp\ a\ P\ s$
by(*auto*)
also from $less$ **have** $\dots < wp\ b\ ?Pc\ s + wp\ b\ P\ s$
by(*auto*)
also from $sab\ sP\ sPc$ **have** $\dots \leq wp\ b\ (\lambda s. ?Pc\ s + P\ s)\ s$
by(*blast*)
finally have $\neg wp\ b\ (\lambda s. bound-of\ P)\ s \leq bound-of\ P$
by(*simp*)
thus $\neg bounded-by\ (bound-of\ P)\ (wp\ b\ (\lambda s. bound-of\ P))$
by(*auto*)
next

However,

fix $P::'s \Rightarrow real$ **assume** $sP: sound\ P$

```

hence nneg ( $\lambda s. \text{bound-of } P$ ) by(auto)
moreover have bounded-by (bound-of  $P$ ) ( $\lambda s. \text{bound-of } P$ ) by(auto)
ultimately
show bounded-by (bound-of  $P$ ) (wp  $b$  ( $\lambda s. \text{bound-of } P$ ))
  using wb by(auto dest!:well-def-wp-healthy)
qed

```

4.9.4 The Algebraic Structure of Refinement

Well-defined programs form a half-bounded semilattice under refinement, where *Abort* is bottom, and $a \sqcap b$ is *inf*. There is no unique top element, but all fully-deterministic programs are maximal.

The type that we construct here is not especially useful, but serves as a convenient way to express this result.

```

quotient-type 's program =
  's prog / partial :  $\lambda a b. a \simeq b \wedge \text{well-def } a \wedge \text{well-def } b$ 
proof(rule part-equivpI)
  have Skip  $\simeq$  Skip and well-def Skip by(auto intro:wd-intros)
  thus  $\exists x. x \simeq x \wedge \text{well-def } x \wedge \text{well-def } x$  by(blast)
  show symp ( $\lambda a b. a \simeq b \wedge \text{well-def } a \wedge \text{well-def } b$ )
  proof(rule sympI, safe)
    fix  $a::'a \text{ prog}$  and  $b$ 
    assume  $a \simeq b$ 
    hence equiv-trans (wp  $a$ ) (wp  $b$ )
      by(simp add:pequiv-equiv-trans)
    thus  $b \simeq a$  by(simp add:ac-simps pequiv-equiv-trans)
  qed
  show transp ( $\lambda a b. a \simeq b \wedge \text{well-def } a \wedge \text{well-def } b$ )
    by(rule transpI, safe, rule pequiv-trans)
qed

```

```

instantiation program :: (type) semilattice-inf begin
lift-definition
  less-eq-program :: 'a program  $\Rightarrow$  'a program  $\Rightarrow$  bool is refines
proof(safe)
  fix  $a::'a \text{ prog}$  and  $b c d$ 
  assume  $a \simeq b$  hence  $b \simeq a$  by(simp add:ac-simps)
  also assume  $a \sqsubseteq c$ 
  also assume  $c \simeq d$ 
  finally show  $b \sqsubseteq d$  .
next
  fix  $a::'a \text{ prog}$  and  $b c d$ 
  assume  $a \simeq b$ 
  also assume  $b \sqsubseteq d$ 
  also assume  $c \simeq d$  hence  $d \simeq c$  by(simp add:ac-simps)
  finally show  $a \sqsubseteq c$  .
qed

```

lift-definition

less-program :: 'a program \Rightarrow 'a program \Rightarrow bool

is $\lambda a b. a \sqsubseteq b \wedge \neg b \sqsubseteq a$

proof(safe)

fix *a*::'a prog **and** *b c d*

assume $a \simeq b$ **hence** $b \simeq a$ **by**(simp add:ac-simps)

also assume $a \sqsubseteq c$

also assume $c \simeq d$

finally show $b \sqsubseteq d$.

next

fix *a*::'a prog **and** *b c d*

assume $a \simeq b$

also assume $b \sqsubseteq d$

also assume $c \simeq d$ **hence** $d \simeq c$ **by**(simp add:ac-simps)

finally show $a \sqsubseteq c$.

next

fix *a b* **and** *c*::'a prog **and** *d*

assume $c \simeq d$

also assume $d \sqsubseteq b$

also assume $a \simeq b$ **hence** $b \simeq a$ **by**(simp add:ac-simps)

finally have $c \sqsubseteq a$.

moreover assume $\neg c \sqsubseteq a$

ultimately show False **by**(auto)

next

fix *a b* **and** *c*::'a prog **and** *d*

assume $c \simeq d$ **hence** $d \simeq c$ **by**(simp add:ac-simps)

also assume $c \sqsubseteq a$

also assume $a \simeq b$

finally have $d \sqsubseteq b$.

moreover assume $\neg d \sqsubseteq b$

ultimately show False **by**(auto)

qed**lift-definition**

inf-program :: 'a program \Rightarrow 'a program \Rightarrow 'a program **is** DC

proof(safe)

fix *a b c d*::'s prog

assume $a \simeq b$ **and** $c \simeq d$

thus $(a \sqcap c) \simeq (b \sqcap d)$ **by**(rule pequiv-DC)

next

fix *a c*::'s prog

assume well-def *a* well-def *c*

thus well-def $(a \sqcap c)$ **by**(rule wd-intros)

next

fix *a c*::'s prog

assume well-def *a* well-def *c*

thus well-def $(a \sqcap c)$ **by**(rule wd-intros)

qed

```

instance
proof
  fix  $x\ y::'a\ program$ 
  show  $(x < y) = (x \leq y \wedge \neg y \leq x)$ 
    by(transfer, simp)
  show  $x \leq x$ 
    by(transfer, auto)
  show  $\inf x\ y \leq x$ 
    by(transfer, rule left-refines-DC)
  show  $\inf x\ y \leq y$ 
    by(transfer, rule right-refines-DC)
  assume  $x \leq y$  and  $y \leq x$  thus  $x = y$ 
    by(transfer, iprover intro:pequiv-antisym)
next
  fix  $x\ y\ z::'a\ program$ 
  assume  $x \leq y$  and  $y \leq z$ 
  thus  $x \leq z$ 
    by(transfer, iprover intro:pr-trans)
next
  fix  $x\ y\ z::'a\ program$ 
  assume  $x \leq y$  and  $x \leq z$ 
  thus  $x \leq \inf y\ z$ 
    by(transfer, iprover intro:DC-refines)
qed
end

instantiation program :: (type) bot begin
lift-definition
  bot-program :: 'a program is Abort
  by(auto intro:wd-intros)

instance ..
end

lemma eq-det:  $\bigwedge a\ b::'s\ prog. \llbracket a \simeq b; \text{determ } (wp\ a) \rrbracket \implies \text{determ } (wp\ b)$ 
proof(intro determI additiveI maximall)
  fix  $a\ b::'s\ prog$  and  $P::'s \Rightarrow real$ 
    and  $Q::'s \Rightarrow real$  and  $s::'s$ 
  assume  $da: \text{determ } (wp\ a)$ 
  assume  $sP: \text{sound } P$  and  $sQ: \text{sound } Q$ 
  and  $eq: a \simeq b$ 
  hence  $wp\ b\ (\lambda s. P\ s + Q\ s)\ s =$ 
     $wp\ a\ (\lambda s. P\ s + Q\ s)\ s$ 
    by(simp add:sound-intros)
  also from  $da\ sP\ sQ$ 
  have  $\dots = wp\ a\ P\ s + wp\ a\ Q\ s$ 
    by(simp add:additiveD determ-additiveD)
  also from  $eq\ sP\ sQ$ 
  have  $\dots = wp\ b\ P\ s + wp\ b\ Q\ s$ 

```



```

by(simp add:pequivD)
finally show wp b ( $\lambda s. P s + Q s$ ) s = wp b P s + wp b Q s .
next
fix a b::'s prog and c::real
assume da: determ (wp a)
assume a  $\simeq$  b hence b  $\simeq$  a by(simp add:ac-simps)
moreover assume nn: 0  $\leq$  c
ultimately have wp b ( $\lambda-. c$ ) = wp a ( $\lambda-. c$ )
  by(simp add:pequivD const-sound)
also from da nn have ... = ( $\lambda-. c$ )
  by(simp add:determ-maximalD maximalD)
finally show wp b ( $\lambda-. c$ ) = ( $\lambda-. c$ ) .
qed

```

lift-definition

```

pdeterm :: 's program  $\Rightarrow$  bool
is  $\lambda a. \text{determ } (\text{wp } a)$ 
proof(safe)
fix a b::'s prog
assume a  $\simeq$  b and determ (wp a)
thus determ (wp b) by(rule eq-det)
next
fix a b::'s prog
assume a  $\simeq$  b hence b  $\simeq$  a by(simp add:ac-simps)
moreover assume determ (wp b)
ultimately show determ (wp a) by(rule eq-det)
qed

```

lemma determ-maximal:

```

 $\llbracket \text{pdeterm } a; a \leq x \rrbracket \Longrightarrow a = x$ 
by(transfer, auto intro:refines-determ)

```

4.9.5 Data Refinement

A projective data refinement construction for pGCL. By projective, we mean that the abstract state is always a function (φ) of the concrete state. Refinement may be predicated (G) on the state.

definition

```

drefines :: ('b  $\Rightarrow$  'a)  $\Rightarrow$  ('b  $\Rightarrow$  bool)  $\Rightarrow$  'a prog  $\Rightarrow$  'b prog  $\Rightarrow$  bool

```

where

```

drefines  $\varphi$  G A B  $\equiv$   $\forall P Q. (\text{unitary } P \wedge \text{unitary } Q \wedge (P \Vdash \text{wp } A Q)) \longrightarrow$ 
  ( $\llbracket G \rrbracket \ \&\& \ (P \circ \varphi) \Vdash \text{wp } B (Q \circ \varphi)$ )

```

lemma drefinesD[dest]:

```

 $\llbracket \text{drefines } \varphi G A B; \text{unitary } P; \text{unitary } Q; P \Vdash \text{wp } A Q \rrbracket \Longrightarrow$ 
   $\llbracket G \rrbracket \ \&\& \ (P \circ \varphi) \Vdash \text{wp } B (Q \circ \varphi)$ 
unfolding drefines-def by(blast)

```

We can alternatively use G as an assumption:

lemma *drefinesD2*:
assumes *dr*: *drefines* φ *G* *A* *B*
and *uP*: *unitary* *P*
and *uQ*: *unitary* *Q*
and *wpA*: $P \Vdash wp\ A\ Q$
and *G*: $G\ s$
shows $(P\ o\ \varphi)\ s \leq wp\ B\ (Q\ o\ \varphi)\ s$
proof –
from *uP* **have** $0 \leq (P\ o\ \varphi)\ s$ **unfolding** *o-def* **by** (*blast*)
with *G* **have** $(P\ o\ \varphi)\ s = \langle\langle G \rangle\rangle \&\& (P\ o\ \varphi)\ s$
by (*simp add:exp-conj-def*)
also from *assms* **have** $\dots \leq wp\ B\ (Q\ o\ \varphi)\ s$ **by** (*blast*)
finally show $(P\ o\ \varphi)\ s \leq \dots$.
qed

This additional form is sometimes useful:

lemma *drefinesD3*:
assumes *dr*: *drefines* φ *G* *a* *b*
and *G*: $G\ s$
and *uQ*: *unitary* *Q*
and *wa*: *well-def* *a*
shows $wp\ a\ Q\ (\varphi\ s) \leq wp\ b\ (Q\ o\ \varphi)\ s$
proof –
let $?L\ s' = wp\ a\ Q\ s'$
from *uQ* *wa* **have** *sL*: *sound* $?L$ **by** (*blast*)
from *uQ* *wa* **have** *bL*: *bounded-by* 1 $?L$ **by** (*blast*)

have $?L \Vdash ?L$ **by** (*simp*)
with *sL* **and** *bL* **and** *assms*
show *thesis*
by (*blast intro:drefinesD2[OF dr, where P=?L, simplified]*)
qed

lemma *drefinesI[intro]*:
 $\llbracket \bigwedge P\ Q. \llbracket \text{unitary } P; \text{unitary } Q; P \Vdash wp\ A\ Q \rrbracket \implies$
 $\langle\langle G \rangle\rangle \&\& (P\ o\ \varphi) \Vdash wp\ B\ (Q\ o\ \varphi) \rrbracket \implies$
drefines φ *G* *A* *B*
unfolding *drefines-def* **by** (*blast*)

Use *G* as an assumption, when showing refinement:

lemma *drefinesI2*:
fixes *A*::'*a* *prog*
and *B*::'*b* *prog*
and φ ::'*b* \Rightarrow '*a*
and *G*::'*b* \Rightarrow *bool*
assumes *wB*: *well-def* *B*
and *withAs*:
 $\bigwedge P\ Q\ s. \llbracket \text{unitary } P; \text{unitary } Q;$
 $G\ s; P \Vdash wp\ A\ Q \rrbracket \implies (P\ o\ \varphi)\ s \leq wp\ B\ (Q\ o\ \varphi)\ s$

shows $drefines\ \varphi\ G\ A\ B$
proof
fix P **and** Q
assume uP : unitary P
and uQ : unitary Q
and wpA : $P \Vdash wp\ A\ Q$

hence $\bigwedge s. G\ s \implies (P\ o\ \varphi)\ s \leq wp\ B\ (Q\ o\ \varphi)\ s$
using $withAs$ **by**($blast$)
moreover
from uQ **have** unitary $(Q\ o\ \varphi)$
unfolding $o-def$ **by**($blast$)
moreover
from uP **have** unitary $(P\ o\ \varphi)$
unfolding $o-def$ **by**($blast$)
ultimately
show $\langle\langle G \rangle\rangle \ \&\&\ (P\ o\ \varphi) \Vdash wp\ B\ (Q\ o\ \varphi)$
using wB **by**($blast\ intro:entails-pconj-assumption$)
qed

lemma $dr-strengthen-guard$:
fixes $a::'s\ prog$ **and** $b::'t\ prog$
assumes fg : $\bigwedge s. F\ s \implies G\ s$
and $drab$: $drefines\ \varphi\ G\ a\ b$
shows $drefines\ \varphi\ F\ a\ b$
proof($intro\ drefinesI$)
fix $P\ Q::'s\ expect$
assume uP : unitary P **and** uQ : unitary Q
and wp : $P \Vdash wp\ a\ Q$
from fg **have** $\bigwedge s. \langle\langle F \rangle\rangle\ s \leq \langle\langle G \rangle\rangle\ s$ **by**($simp\ add:embed-bool-def$)
hence $(\langle\langle F \rangle\rangle \ \&\&\ (P\ o\ \varphi)) \Vdash (\langle\langle G \rangle\rangle \ \&\&\ (P\ o\ \varphi))$ **by**($auto\ intro:pconj-mono\ le-funI$
 $simp:exp-conj-def$)
also from $drab\ uP\ uQ\ wp$ **have** $\dots \Vdash wp\ b\ (Q\ o\ \varphi)$ **by**($auto$)
finally show $\langle\langle F \rangle\rangle \ \&\&\ (P\ o\ \varphi) \Vdash wp\ b\ (Q\ o\ \varphi)$.
qed

Probabilistic correspondence, $pcorres$, is equality on distribution transformers, modulo a guard. It is the analogue, for data refinement, of program equivalence for program refinement.

definition

$pcorres :: ('b \Rightarrow 'a) \Rightarrow ('b \Rightarrow bool) \Rightarrow 'a\ prog \Rightarrow 'b\ prog \Rightarrow bool$
where
 $pcorres\ \varphi\ G\ A\ B \longleftrightarrow$
 $(\forall Q. unitary\ Q \longrightarrow \langle\langle G \rangle\rangle \ \&\&\ (wp\ A\ Q\ o\ \varphi) = \langle\langle G \rangle\rangle \ \&\&\ wp\ B\ (Q\ o\ \varphi))$

lemma $pcorresI$:

$\llbracket \bigwedge Q. unitary\ Q \implies \langle\langle G \rangle\rangle \ \&\&\ (wp\ A\ Q\ o\ \varphi) = \langle\langle G \rangle\rangle \ \&\&\ wp\ B\ (Q\ o\ \varphi) \rrbracket \implies$
 $pcorres\ \varphi\ G\ A\ B$
by($simp\ add:pcorres-def$)

Often easier to use, as it allows one to assume the precondition.

```

lemma pcorresI2[intro]:
  fixes A::'a prog and B::'b prog
  assumes withG:  $\bigwedge Q s. \llbracket \text{unitary } Q; G s \rrbracket \implies \text{wp } A \ Q \ (\varphi s) = \text{wp } B \ (Q \circ \varphi) \ s$ 
    and wA: well-def A
    and wB: well-def B
  shows pcorres  $\varphi \ G \ A \ B$ 
proof(rule pcorresI, rule ext)
  fix Q::'a  $\Rightarrow$  real and s::'b
  assume uQ: unitary Q
  hence uQ $\varphi$ : unitary (Q  $\circ$   $\varphi$ ) by(auto)
  show ( $\llbracket G \rrbracket \ \&\& \ (\text{wp } A \ Q \circ \varphi) \ s = \llbracket G \rrbracket \ \&\& \ \text{wp } B \ (Q \circ \varphi) \ s$ )
  proof(cases G s)
    case True note this
    moreover
    from well-def-wp-healthy[OF wA] uQ have  $0 \leq \text{wp } A \ Q \ (\varphi s)$  by(blast)
    moreover
    from well-def-wp-healthy[OF wB] uQ $\varphi$  have  $0 \leq \text{wp } B \ (Q \circ \varphi) \ s$  by(blast)
    ultimately show ?thesis
    using uQ by(simp add:exp-conj-def withG)
  next
  case False note this
  moreover
  from well-def-wp-healthy[OF wA] uQ have  $\text{wp } A \ Q \ (\varphi s) \leq 1$  by(blast)
  moreover
  from well-def-wp-healthy[OF wB] uQ $\varphi$  have  $\text{wp } B \ (Q \circ \varphi) \ s \leq 1$ 
    by(blast dest!:healthy-bounded-byD intro:sound-nneg)
  ultimately show ?thesis by(simp add:exp-conj-def)
qed
qed

```

```

lemma pcorresD:
   $\llbracket \text{pcorres } \varphi \ G \ A \ B; \text{unitary } Q \rrbracket \implies \llbracket G \rrbracket \ \&\& \ (\text{wp } A \ Q \circ \varphi) = \llbracket G \rrbracket \ \&\& \ \text{wp } B \ (Q \circ \varphi)$ 
  unfolding pcorres-def by(simp)

```

Again, easier to use if the precondition is known to hold.

```

lemma pcorresD2:
  assumes pc: pcorres  $\varphi \ G \ A \ B$ 
    and uQ: unitary Q
    and wA: well-def A and wB: well-def B
    and G: G s
  shows  $\text{wp } A \ Q \ (\varphi s) = \text{wp } B \ (Q \circ \varphi) \ s$ 
proof –
  from uQ well-def-wp-healthy[OF wA] have  $0 \leq \text{wp } A \ Q \ (\varphi s)$  by(auto)
  with G have  $\text{wp } A \ Q \ (\varphi s) = \llbracket G \rrbracket \ s \ .\& \ \text{wp } A \ Q \ (\varphi s)$  by(simp)
  also {
    from pc uQ have  $\llbracket G \rrbracket \ \&\& \ (\text{wp } A \ Q \circ \varphi) = \llbracket G \rrbracket \ \&\& \ \text{wp } B \ (Q \circ \varphi)$ 
      by(rule pcorresD)
    hence  $\llbracket G \rrbracket \ s \ .\& \ \text{wp } A \ Q \ (\varphi s) = \llbracket G \rrbracket \ s \ .\& \ \text{wp } B \ (Q \circ \varphi) \ s$ 
  }

```

```

  unfolding exp-conj-def o-def by(rule fun-cong)
}
also {
  from uQ have sound Q by(auto)
  hence sound (Q o  $\varphi$ ) by(auto intro:sound-intros)
  with well-def-wp-healthy[OF wB] have  $0 \leq wp B (Q o \varphi) s$  by(auto)
  with G have  $\langle G \rangle s .\& wp B (Q o \varphi) s = wp B (Q o \varphi) s$  by(simp)
}
finally show ?thesis .
qed

```

4.9.6 The Algebra of Data Refinement

Program refinement implies a trivial data refinement:

```

lemma refines-drefines:
  fixes a::'s prog
  assumes rab:  $a \sqsubseteq b$  and wb: well-def b
  shows drefines ( $\lambda s. s$ ) G a b
proof(intro drefinesI2 wb, simp add:o-def)
  fix P::'s  $\Rightarrow$  real and Q::'s  $\Rightarrow$  real and s::'s
  assume sQ: unitary Q
  assume  $P \Vdash wp a Q$  hence  $P s \leq wp a Q s$  by(auto)
  also from rab sQ have  $\dots \leq wp b Q s$  by(auto)
  finally show  $P s \leq wp b Q s$  .
qed

```

Data refinement is transitive:

```

lemma dr-trans[trans]:
  fixes A::'a prog and B::'b prog and C::'c prog
  assumes drAB: drefines  $\varphi$  G A B
    and drBC: drefines  $\varphi'$  G' B C
    and Gimp:  $\bigwedge s. G' s \Longrightarrow G (\varphi' s)$ 
  shows drefines ( $\varphi o \varphi'$ ) G' A C
proof(rule drefinesI)
  fix P::'a  $\Rightarrow$  real and Q::'a  $\Rightarrow$  real and s::'a
  assume uP: unitary P and uQ: unitary Q
  and wpA:  $P \Vdash wp A Q$ 

  have  $\langle G' \rangle \&\& \langle G o \varphi' \rangle = \langle G' \rangle$ 
proof(rule ext, unfold exp-conj-def)
  fix x
  show  $\langle G' \rangle x .\& \langle G o \varphi' \rangle x = \langle G' \rangle x$  (is ?X)
proof(cases G' x)
  case False then show ?X by(simp)
next
  case True
  moreover
  with Gimp have  $(G o \varphi') x$  by(simp add:o-def)
  ultimately

```

```

    show ?X by(simp)
  qed
qed

with uP
have «G'» && (P o (φ o φ')) = «G'» && ((«G» && (P o φ)) o φ')
  by(simp add:exp-conj-assoc o-assoc)

also {
  from uP uQ wpA and drAB
  have «G» && (P o φ) ⊢ wp B (Q o φ)
    by(blast intro:drefinesD)

  with drBC and uP uQ
  have «G'» && ((«G» && (P o φ)) o φ') ⊢ wp C ((Q o φ) o φ')
    by(blast intro:unitary-intros drefinesD)
}

finally
show «G'» && (P o (φ o φ')) ⊢ wp C (Q o (φ o φ'))
  by(simp add:o-assoc)
qed

```

Data refinement composes with program refinement:

```

lemma pr-dr-trans[trans]:
  assumes prAB: A ⊆ B
    and drBC: drefines φ G B C
  shows drefines φ G A C
proof(rule drefinesI)
  fix P and Q
  assume uP: unitary P
    and uQ: unitary Q
    and wpA: P ⊢ wp A Q

  note wpA
  also from uQ and prAB have wp A Q ⊢ wp B Q by(blast)
  finally have P ⊢ wp B Q .
  with uP uQ drBC
  show «G» && (P o φ) ⊢ wp C (Q o φ) by(blast intro:drefinesD)
qed

```

```

lemma dr-pr-trans[trans]:
  assumes drAB: drefines φ G A B
  assumes prBC: B ⊆ C
  shows drefines φ G A C
proof(rule drefinesI)
  fix P and Q
  assume uP: unitary P
    and uQ: unitary Q

```

```

and wpA:  $P \Vdash wp A Q$ 

with drAB have  $\langle G \rangle \ \&\& \ (P \circ \varphi) \Vdash wp B (Q \circ \varphi)$  by(blast intro:drefinesD)
also from uQ prBC have  $\dots \Vdash wp C (Q \circ \varphi)$  by(blast)
finally show  $\langle G \rangle \ \&\& \ (P \circ \varphi) \Vdash \dots$  .
qed

```

If the projection φ commutes with the transformer, then data refinement is reflexive:

```

lemma dr-refl:
  assumes wa: well-def a
    and comm:  $\bigwedge Q. \text{unitary } Q \implies wp a Q \circ \varphi \Vdash wp a (Q \circ \varphi)$ 
  shows drefines  $\varphi G a a$ 
proof(intro drefinesI2 wa)
  fix P and Q and s
  assume wp:  $P \Vdash wp a Q$ 
  assume uQ: unitary Q

  have  $(P \circ \varphi) s = P (\varphi s)$  by(simp)
  also from wp have  $\dots \leq wp a Q (\varphi s)$  by(blast)
  also {
    from comm uQ have  $wp a Q \circ \varphi \Vdash wp a (Q \circ \varphi)$  by(blast)
    hence  $(wp a Q \circ \varphi) s \leq wp a (Q \circ \varphi) s$  by(blast)
    hence  $wp a Q (\varphi s) \leq \dots$  by(simp)
  }
  finally show  $(P \circ \varphi) s \leq wp a (Q \circ \varphi) s$  .
qed

```

Correspondence implies data refinement

```

lemma pcorres-drefine:
  assumes corres: pcorres  $\varphi G A C$ 
    and wC: well-def C
  shows drefines  $\varphi G A C$ 
proof
  fix P and Q
  assume uP: unitary P and uQ: unitary Q
    and wpA:  $P \Vdash wp A Q$ 
  from wpA have  $P \circ \varphi \Vdash wp A Q \circ \varphi$  by(simp add:o-def le-fun-def)
  hence  $\langle G \rangle \ \&\& \ (P \circ \varphi) \Vdash \langle G \rangle \ \&\& \ (wp A Q \circ \varphi)$ 
    by(rule exp-conj-mono-right)
  also from corres uQ
  have  $\dots = \langle G \rangle \ \&\& \ (wp C (Q \circ \varphi))$  by(rule pcorresD)
  also
  have  $\dots \Vdash wp C (Q \circ \varphi)$ 
  proof(rule le-funI)
    fix s
    from uQ have unitary  $(Q \circ \varphi)$  by(rule unitary-intros)
    with well-def-wp-healthy[OF wC] have nn-wpC:  $0 \leq wp C (Q \circ \varphi) s$  by(blast)
    show  $(\langle G \rangle \ \&\& \ wp C (Q \circ \varphi)) s \leq wp C (Q \circ \varphi) s$ 

```

```

proof(cases  $G$   $s$ )
  case True
    with  $nn\text{-}wpC$  show ?thesis by(simp add:exp-conj-def)
  next
    case False note this
    moreover {
      from  $uQ$  have unitary ( $Q \circ \varphi$ ) by(simp)
      with well-def-wp-healthy[OF  $wC$ ] have  $wp\ C\ (Q \circ \varphi)\ s \leq 1$  by(auto)
    }
    moreover note  $nn\text{-}wpC$ 
    ultimately show ?thesis by(simp add:exp-conj-def)
  qed
qed
finally show  $\langle\langle G \rangle\rangle \ \&\&\ (P \circ \varphi) \Vdash wp\ C\ (Q \circ \varphi)$  .
qed

```

Any *data* refinement of a deterministic program is correspondence. This is the analogous result to that relating program refinement and equivalence.

lemma *drefines-determ*:

```

fixes  $a::'a\ prog$  and  $b::'b\ prog$ 
assumes  $da: determ\ (wp\ a)$ 
  and  $wa: well\text{-}def\ a$ 
  and  $wb: well\text{-}def\ b$ 
  and  $dr: drefines\ \varphi\ G\ a\ b$ 
shows  $pcorres\ \varphi\ G\ a\ b$ 

```

The proof follows exactly the same form as that for program refinement: Assuming that correspondence *doesn't* hold, we show that $wp\ b$ is not feasible, and thus not healthy, contradicting the assumption.

```

proof(rule  $pcorresI$ , rule contrapos-pp)
  from  $wb$  show feasible ( $wp\ b$ ) by(auto)

  note  $ha = well\text{-}def\text{-}wp\text{-}healthy[OF\ wa]$ 
  note  $hb = well\text{-}def\text{-}wp\text{-}healthy[OF\ wb]$ 

  from  $wb$  have sublinear ( $wp\ b$ ) by(auto)
  moreover from  $hb$  have feasible ( $wp\ b$ ) by(auto)
  ultimately have  $sab: sub\text{-}add\ (wp\ b)$  by(rule sublinear-subadd)

  fix  $Q::'a \Rightarrow real$ 
  assume  $uQ: unitary\ Q$ 
  hence  $uQ\varphi: unitary\ (Q \circ \varphi)$  by(auto)
  assume  $ne: \langle\langle G \rangle\rangle \ \&\&\ (wp\ a\ Q \circ \varphi) \neq \langle\langle G \rangle\rangle \ \&\&\ wp\ b\ (Q \circ \varphi)$ 
  hence  $ne': wp\ a\ Q \circ \varphi \neq wp\ b\ (Q \circ \varphi)$  by(auto)

```

From refinement, $\langle\langle G \rangle\rangle \ \&\&\ (wp\ a\ Q \circ \varphi)$ lies below $\langle\langle G \rangle\rangle \ \&\&\ wp\ b\ (Q \circ \varphi)$.

```

from  $ha\ uQ$ 
have  $gle: \langle\langle G \rangle\rangle \ \&\&\ (wp\ a\ Q \circ \varphi) \Vdash wp\ b\ (Q \circ \varphi)$  by(blast intro!:drefinesD[OF  $dr$ ])
have  $le: \langle\langle G \rangle\rangle \ \&\&\ (wp\ a\ Q \circ \varphi) \Vdash \langle\langle G \rangle\rangle \ \&\&\ wp\ b\ (Q \circ \varphi)$ 

```



```

unfolding exp-conj-def
proof(rule le-funI)
  fix s
  from gle have  $\langle\langle G \rangle\rangle s .\& (wp\ a\ Q\ o\ \varphi)\ s \leq wp\ b\ (Q\ o\ \varphi)\ s$ 
    unfolding exp-conj-def by(auto)
  hence  $\langle\langle G \rangle\rangle s .\& (\langle\langle G \rangle\rangle s .\& (wp\ a\ Q\ o\ \varphi)\ s) \leq \langle\langle G \rangle\rangle s .\& wp\ b\ (Q\ o\ \varphi)\ s$ 
    by(auto intro:pconj-mono)
  moreover from uQ ha have  $wp\ a\ Q\ (\varphi\ s) \leq 1$ 
    by(auto dest:healthy-bounded-byD)
  moreover from uQ ha have  $0 \leq wp\ a\ Q\ (\varphi\ s)$ 
    by(auto)
  ultimately
  show  $\langle\langle G \rangle\rangle s .\& (wp\ a\ Q\ o\ \varphi)\ s \leq \langle\langle G \rangle\rangle s .\& wp\ b\ (Q\ o\ \varphi)\ s$ 
    by(simp add:pconj-assoc)
qed

```

If the programs do not correspond, the terms must differ somewhere, and given the previous result, the second must be somewhere strictly larger than the first:

```

have nle:  $\exists s. (\langle\langle G \rangle\rangle \&\& (wp\ a\ Q\ o\ \varphi))\ s < (\langle\langle G \rangle\rangle \&\& wp\ b\ (Q\ o\ \varphi))\ s$ 
proof(rule contrapos-np[OF ne], rule ext, rule antisym)
  fix s
  from le show  $(\langle\langle G \rangle\rangle \&\& (wp\ a\ Q\ o\ \varphi))\ s \leq (\langle\langle G \rangle\rangle \&\& wp\ b\ (Q\ o\ \varphi))\ s$ 
    by(blast)
next
  fix s
  assume  $\neg (\exists s. (\langle\langle G \rangle\rangle \&\& (wp\ a\ Q\ o\ \varphi))\ s < (\langle\langle G \rangle\rangle \&\& wp\ b\ (Q\ o\ \varphi))\ s)$ 
  thus  $(\langle\langle G \rangle\rangle \&\& (wp\ b\ (Q\ o\ \varphi)))\ s \leq (\langle\langle G \rangle\rangle \&\& (wp\ a\ Q\ o\ \varphi))\ s$ 
    by(simp add:not-less)
qed
from this obtain s where less-s:
   $(\langle\langle G \rangle\rangle \&\& (wp\ a\ Q\ o\ \varphi))\ s < (\langle\langle G \rangle\rangle \&\& wp\ b\ (Q\ o\ \varphi))\ s$ 
  by(blast)

```

The transformers themselves must differ at this point:

```

hence larger:  $wp\ a\ Q\ (\varphi\ s) < wp\ b\ (Q\ o\ \varphi)\ s$ 
proof(cases G s)
  case True
  moreover from ha uQ have  $0 \leq wp\ a\ Q\ (\varphi\ s)$ 
    by(blast)
  moreover from hb uQ have  $0 \leq wp\ b\ (Q\ o\ \varphi)\ s$ 
    by(blast)
  moreover note less-s
  ultimately show ?thesis by(simp add:exp-conj-def)
next
  case False
  moreover from ha uQ have  $wp\ a\ Q\ (\varphi\ s) \leq 1$ 
    by(blast)
  moreover {
    from uQ have bounded-by 1  $(Q\ o\ \varphi)$ 

```

```

    by(blast)
    moreover from unitary-sound[OF uQ]
    have sound ( $Q \circ \varphi$ ) by(auto)
    ultimately have wp b ( $Q \circ \varphi$ )  $s \leq 1$ 
    using hb by(auto)
  }
  moreover note less-s
  ultimately show ?thesis by(simp add:exp-conj-def)
qed
from less-s have ( $\langle\langle G \rangle\rangle \ \&\& \ (wp \ a \ Q \circ \varphi) \ s \neq \langle\langle G \rangle\rangle \ \&\& \ wp \ b \ (Q \circ \varphi) \ s$ )
by(force)

```

G must also hold, as otherwise both would be zero.

```

hence G-s:  $G \ s$ 
proof(rule contrapos-np)
  assume nG:  $\neg \ G \ s$ 
  moreover from ha uQ have wp a  $Q \ (\varphi \ s) \leq 1$ 
  by(blast)
  moreover {
    from uQ have bounded-by 1 ( $Q \circ \varphi$ )
    by(blast)
    moreover from unitary-sound[OF uQ]
    have sound ( $Q \circ \varphi$ ) by(auto)
    ultimately have wp b ( $Q \circ \varphi$ )  $s \leq 1$ 
    using hb by(auto)
  }
  ultimately
  show ( $\langle\langle G \rangle\rangle \ \&\& \ (wp \ a \ Q \circ \varphi) \ s = \langle\langle G \rangle\rangle \ \&\& \ wp \ b \ (Q \circ \varphi) \ s$ )
  by(simp add:exp-conj-def)
qed

```

Take a carefully constructed expectation:

```

let ?Qc =  $\lambda s. \text{bound-of } Q - Q \ s$ 
have bQc: bounded-by 1 ?Qc
proof(rule bounded-byI)
  fix s
  from uQ have bound-of  $Q \leq 1$  and  $0 \leq Q \ s$  by(auto)
  thus bound-of  $Q - Q \ s \leq 1$  by(auto)
qed
have sQc: sound ?Qc
proof(rule soundI)
  from bQc show bounded ?Qc by(auto)

  show nneg ?Qc
  proof(rule nnegI)
    fix s
    from uQ have  $Q \ s \leq \text{bound-of } Q$  by(auto)
    thus  $0 \leq \text{bound-of } Q - Q \ s$  by(auto)
  qed
qed

```

qed

By the maximality of $wp\ a$, $wp\ b$ must violate feasibility, by mapping s to something strictly greater than $bound\text{-}of\ Q$.

```

from  $uQ$  have  $0 \leq bound\text{-}of\ Q$  by(auto)
with  $da$  have  $bound\text{-}of\ Q = wp\ a\ (\lambda s. bound\text{-}of\ Q)\ (\varphi\ s)$ 
  by(simp add:maximalD determ-maximalD)
also have  $wp\ a\ (\lambda s. bound\text{-}of\ Q)\ (\varphi\ s) = wp\ a\ (\lambda s. Q\ s + ?Qc\ s)\ (\varphi\ s)$ 
  by(simp)
also {
  from  $da$  have additive ( $wp\ a$ ) by(blast)
  with  $uQ\ sQc$ 
  have  $wp\ a\ (\lambda s. Q\ s + ?Qc\ s)\ (\varphi\ s) =$ 
     $wp\ a\ Q\ (\varphi\ s) + wp\ a\ ?Qc\ (\varphi\ s)$  by(subst additiveD, blast+)
}
also {
  from  $ha$  and  $sQc$  and  $bQc$ 
  have  $\langle\langle G \rangle\rangle \ \&\&\ (wp\ a\ ?Qc\ o\ \varphi) \Vdash wp\ b\ (?Qc\ o\ \varphi)$ 
    by(blast intro!:drefinesD[OF dr])
  hence  $\langle\langle G \rangle\rangle \ \&\&\ (wp\ a\ ?Qc\ o\ \varphi) \ s \leq wp\ b\ (?Qc\ o\ \varphi)\ s$ 
    by(blast)
  moreover from  $sQc$  and  $ha$ 
  have  $0 \leq wp\ a\ (\lambda s. bound\text{-}of\ Q - Q\ s)\ (\varphi\ s)$ 
    by(blast)
  ultimately
  have  $wp\ a\ ?Qc\ (\varphi\ s) \leq wp\ b\ (?Qc\ o\ \varphi)\ s$ 
    using  $G\text{-}s$  by(simp add:exp-conj-def)
  hence  $wp\ a\ Q\ (\varphi\ s) + wp\ a\ ?Qc\ (\varphi\ s) \leq wp\ a\ Q\ (\varphi\ s) + wp\ b\ (?Qc\ o\ \varphi)\ s$ 
    by(rule add-left-mono)
  also with larger
  have  $wp\ a\ Q\ (\varphi\ s) + wp\ b\ (?Qc\ o\ \varphi)\ s <$ 
     $wp\ b\ (Q\ o\ \varphi)\ s + wp\ b\ (?Qc\ o\ \varphi)\ s$ 
    by(auto)
  finally
  have  $wp\ a\ Q\ (\varphi\ s) + wp\ a\ ?Qc\ (\varphi\ s) <$ 
     $wp\ b\ (Q\ o\ \varphi)\ s + wp\ b\ (?Qc\ o\ \varphi)\ s$  .
}
also from  $sab$  and unitary-sound[OF uQ] and  $sQc$ 
have  $wp\ b\ (Q\ o\ \varphi)\ s + wp\ b\ (?Qc\ o\ \varphi)\ s \leq$ 
   $wp\ b\ (\lambda s. (Q\ o\ \varphi)\ s + (?Qc\ o\ \varphi)\ s)\ s$ 
  by(blast)
also have  $\dots = wp\ b\ (\lambda s. bound\text{-}of\ Q)\ s$ 
  by(simp)
finally
show  $\neg\ feasible\ (wp\ b)$ 
proof(rule contrapos-pn)
  assume  $fb: feasible\ (wp\ b)$ 
  have bounded-by ( $bound\text{-}of\ Q$ ) ( $\lambda s. bound\text{-}of\ Q$ ) by(blast)
  hence bounded-by ( $bound\text{-}of\ Q$ ) ( $wp\ b\ (\lambda s. bound\text{-}of\ Q)$ )

```

```

    using  $uQ$  by (blast intro:feasible-boundedD[OF fb])
    hence  $wp\ b\ (\lambda s. \text{bound-of } Q)\ s \leq \text{bound-of } Q$  by (blast)
    thus  $\neg \text{bound-of } Q < wp\ b\ (\lambda s. \text{bound-of } Q)\ s$  by (simp)
qed
qed

```

4.9.7 Structural Rules for Correspondence

```

lemma pcorres-Skip:
  pcorres  $\varphi\ G\ \text{Skip}\ \text{Skip}$ 
  by (simp add: pcorres-def wp-eval)

```

Correspondence composes over sequential composition.

```

lemma pcorres-Seq:
  fixes  $A::'b\ \text{prog}$  and  $B::'c\ \text{prog}$ 
  and  $C::'b\ \text{prog}$  and  $D::'c\ \text{prog}$ 
  and  $\varphi::'c \Rightarrow 'b$ 
  assumes  $pcAB: pcorres\ \varphi\ G\ A\ B$ 
  and  $pcCD: pcorres\ \varphi\ H\ C\ D$ 
  and  $wA: \text{well-def } A$  and  $wB: \text{well-def } B$ 
  and  $wC: \text{well-def } C$  and  $wD: \text{well-def } D$ 
  and  $p3p2: \bigwedge Q. \text{unitary } Q \Longrightarrow \langle I \rangle \ \&\&\ wp\ B\ Q = wp\ B\ (\langle H \rangle \ \&\&\ Q)$ 
  and  $p1p3: \bigwedge s. G\ s \Longrightarrow I\ s$ 
  shows  $pcorres\ \varphi\ G\ (A;;C)\ (B;;D)$ 
proof (rule pcorresI)
  fix  $Q::'b \Rightarrow \text{real}$ 
  assume  $uQ: \text{unitary } Q$ 
  with  $\text{well-def-wp-healthy}[OF\ wC]$  have  $uCQ: \text{unitary } (wp\ C\ Q)$  by (auto)
  from  $uQ\ \text{well-def-wp-healthy}[OF\ wD]$  have  $uDQ: \text{unitary } (wp\ D\ (Q\ o\ \varphi))$ 
  by (auto dest:unitary-comp)

  have  $p3p1: \bigwedge R\ S. [\text{unitary } R; \text{unitary } S; \langle I \rangle \ \&\&\ R = \langle I \rangle \ \&\&\ S] \Longrightarrow$ 
     $\langle G \rangle \ \&\&\ R = \langle G \rangle \ \&\&\ S$ 
  proof (rule ext)
    fix  $R::'c \Rightarrow \text{real}$  and  $S::'c \Rightarrow \text{real}$  and  $s::'c$ 
    assume  $a3: \langle I \rangle \ \&\&\ R = \langle I \rangle \ \&\&\ S$ 
    and  $uR: \text{unitary } R$  and  $uS: \text{unitary } S$ 
    show  $\langle G \rangle \ \&\&\ R\ s = \langle G \rangle \ \&\&\ S\ s$ 
    proof (simp add:exp-conj-def, cases  $G\ s$ )
      case False note this
      moreover from  $uR$  have  $R\ s \leq I$  by (blast)
      moreover from  $uS$  have  $S\ s \leq I$  by (blast)
      ultimately show  $\langle G \rangle \ s.\&\ R\ s = \langle G \rangle \ s.\&\ S\ s$ 
      by (simp)
    next
      case True note  $p1 = \text{this}$ 
      with  $p1p3$  have  $I\ s$  by (blast)
      with  $\text{fun-cong}[OF\ a3, \text{where } x=s]$  have  $I.\&\ R\ s = I.\&\ S\ s$ 
      by (simp add:exp-conj-def)
    qed
  qed

```

```

with  $p1$  show  $\langle\langle G \rangle\rangle s . \& R s = \langle\langle G \rangle\rangle s . \& S s$ 
  by(simp)
qed
qed

show  $\langle\langle G \rangle\rangle \&\& (wp (A;;C) Q \circ \varphi) = \langle\langle G \rangle\rangle \&\& wp (B;;D) (Q \circ \varphi)$ 
proof(simp add:wp-eval)
  from  $uCQ$  pcAB have  $\langle\langle G \rangle\rangle \&\& (wp A (wp C Q) \circ \varphi) =$ 
     $\langle\langle G \rangle\rangle \&\& wp B ((wp C Q) \circ \varphi)$ 
  by(auto dest:pcorresD)
  also have  $\langle\langle G \rangle\rangle \&\& wp B ((wp C Q) \circ \varphi) =$ 
     $\langle\langle G \rangle\rangle \&\& wp B (wp D (Q \circ \varphi))$ 
proof(rule p3p1)
  from  $uCQ$  well-def-wp-healthy[OF wB] show unitary  $(wp B (wp C Q \circ \varphi))$ 
  by(auto intro:unitary-comp)
  from  $uDQ$  well-def-wp-healthy[OF wB] show unitary  $(wp B (wp D (Q \circ \varphi)))$ 
  by(auto)

  from  $uQ$  have  $\langle\langle H \rangle\rangle \&\& (wp C Q \circ \varphi) = \langle\langle H \rangle\rangle \&\& wp D (Q \circ \varphi)$ 
  by(blast intro:pcorresD[OF pcCD])
  thus  $\langle\langle I \rangle\rangle \&\& wp B (wp C Q \circ \varphi) = \langle\langle I \rangle\rangle \&\& wp B (wp D (Q \circ \varphi))$ 
  by(simp add:p3p2 uCQ uDQ)
qed
finally show  $\langle\langle G \rangle\rangle \&\& (wp A (wp C Q) \circ \varphi) = \langle\langle G \rangle\rangle \&\& wp B (wp D (Q \circ \varphi)) .$ 
qed
qed

```

4.9.8 Structural Rules for Data Refinement

```

lemma dr-Skip:
  fixes  $\varphi::'c \Rightarrow 'b$ 
  shows drefines  $\varphi$  G Skip Skip
proof(intro drefinesI2 wd-intros)
  fix  $P::'b \Rightarrow \text{real}$  and  $Q::'b \Rightarrow \text{real}$  and  $s::'c$ 
  assume  $P \Vdash wp \text{Skip } Q$ 
  hence  $(P \circ \varphi) s \leq wp \text{Skip } Q (\varphi s)$  by(simp, blast)
  thus  $(P \circ \varphi) s \leq wp \text{Skip } (Q \circ \varphi) s$  by(simp add:wp-eval)
qed

lemma dr-Abort:
  fixes  $\varphi::'c \Rightarrow 'b$ 
  shows drefines  $\varphi$  G Abort Abort
proof(intro drefinesI2 wd-intros)
  fix  $P::'b \Rightarrow \text{real}$  and  $Q::'b \Rightarrow \text{real}$  and  $s::'c$ 
  assume  $P \Vdash wp \text{Abort } Q$ 
  hence  $(P \circ \varphi) s \leq wp \text{Abort } Q (\varphi s)$  by(auto)
  thus  $(P \circ \varphi) s \leq wp \text{Abort } (Q \circ \varphi) s$  by(simp add:wp-eval)
qed

```

lemma *dr-Apply*:

fixes $\varphi::'c \Rightarrow 'b$

assumes *commutes*: $f \circ \varphi = \varphi \circ g$

shows *drefines* φ *G* (*Apply* *f*) (*Apply* *g*)

proof(*intro drefinesI2 wd-intros*)

fix $P::'b \Rightarrow \text{real}$ **and** $Q::'b \Rightarrow \text{real}$ **and** $s::'c$

assume $wp: P \Vdash wp$ (*Apply* *f*) *Q*

hence $P \Vdash (Q \circ f)$ **by**(*simp add:wp-eval*)

hence $P (\varphi s) \leq (Q \circ f) (\varphi s)$ **by**(*blast*)

also have $\dots = Q ((f \circ \varphi) s)$ **by**(*simp*)

also with *commutes*

have $\dots = ((Q \circ \varphi) \circ g) s$ **by**(*simp*)

also have $\dots = wp$ (*Apply* *g*) ($Q \circ \varphi$) *s*

by(*simp add:wp-eval*)

finally show $(P \circ \varphi) s \leq wp$ (*Apply* *g*) ($Q \circ \varphi$) *s* **by**(*simp*)

qed

lemma *dr-Seq*:

assumes *drAB*: *drefines* φ *P* *A* *B*

and *drBC*: *drefines* φ *Q* *C* *D*

and $wpB: \langle P \rangle \Vdash wp$ *B* $\langle Q \rangle$

and *wB*: *well-def* *B*

and *wC*: *well-def* *C*

and *wD*: *well-def* *D*

shows *drefines* φ *P* (*A*;;*C*) (*B*;;*D*)

proof

fix *R* **and** *S*

assume *uR*: *unitary* *R* **and** *uS*: *unitary* *S*

and $wpAC: R \Vdash wp$ (*A*;;*C*) *S*

from *uR*

have $\langle P \rangle \&\& (R \circ \varphi) = \langle P \rangle \&\& (\langle P \rangle \&\& (R \circ \varphi))$

by(*simp add:exp-conj-assoc*)

also {

from *well-def-wp-healthy*[*OF* *wC*] *uR* *uS*

and $wpAC$ [*unfolded eval-wp-Seq o-def*]

have $\langle P \rangle \&\& (R \circ \varphi) \Vdash wp$ *B* (wp *C* *S* $\circ \varphi$)

by(*auto intro:drefinesD*[*OF* *drAB*])

with wpB *well-def-wp-healthy*[*OF* *wC*] *uS*

sublinear-sub-conj[*OF* *well-def-wp-sublinear*, *OF* *wB*]

have $\langle P \rangle \&\& (\langle P \rangle \&\& (R \circ \varphi)) \Vdash wp$ *B* ($\langle Q \rangle \&\& (wp$ *C* *S* $\circ \varphi)$)

by(*auto intro!:entails-combine dest!:unitary-sound*)

}

also {

from *uS* *well-def-wp-healthy*[*OF* *wC*]

have $\langle Q \rangle \&\& (wp$ *C* *S* $\circ \varphi) \Vdash wp$ *D* (*S* $\circ \varphi$)

```

  by(auto intro!:drefinesD[OF drBC])
  with well-def-wp-healthy[OF wB] well-def-wp-healthy[OF wC]
    well-def-wp-healthy[OF wD] and unitary-sound[OF uS]
  have wp B («Q» && (wp C S o φ)) ⊢ wp B (wp D (S o φ))
  by(blast intro!:mono-transD)
}

finally
show «P» && (R o φ) ⊢ wp (B;;D) (S o φ)
  unfolding wp-eval o-def .
qed

lemma dr-repeat:
  fixes φ :: 'a ⇒ 'b
  assumes dr-ab: drefines φ G a b
    and Gpr: «G» ⊢ wp b «G»
    and wa: well-def a
    and wb: well-def b
  shows drefines φ G (repeat n a) (repeat n b) (is ?X n)
proof(induct n)
  show ?X 0 by(simp add:dr-Skip)

  fix n
  assume IH: ?X n
  thus ?X (Suc n) by(auto intro!:dr-Seq Gpr assms wd-intros)
qed

end

```

4.10 Structured Reasoning

theory StructuredReasoning imports Algebra begin

By linking the algebraic, the syntactic, and the semantic views of computation, we derive a set of rules for decomposing expectation entailment proofs, firstly over the syntactic structure of a program, and secondly over the refinement relation. These rules also form the basis for automated reasoning.

4.10.1 Syntactic Decomposition

```

lemma wp-Abort:
  (λs. 0) ⊢ wp Abort Q
  unfolding wp-eval by(simp)

```

```

lemma wlp-Abort:
  (λs. 1) ⊢ wlp Abort Q
  unfolding wp-eval by(simp)

```

lemma *wp-Skip*:

$P \Vdash wp \text{ Skip } P$

unfolding *wp-eval* **by**(*blast*)

lemma *wlp-Skip*:

$P \Vdash wlp \text{ Skip } P$

unfolding *wp-eval* **by**(*blast*)

lemma *wp-Apply*:

$Q \text{ of } \Vdash wp \text{ (Apply } f) Q$

unfolding *wp-eval* **by**(*simp*)

lemma *wlp-Apply*:

$Q \text{ of } \Vdash wlp \text{ (Apply } f) Q$

unfolding *wp-eval* **by**(*simp*)

lemma *wp-Seq*:

assumes *ent-a*: $P \Vdash wp \ a \ Q$

and *ent-b*: $Q \Vdash wp \ b \ R$

and *wa*: *well-def* *a*

and *wb*: *well-def* *b*

and *s-Q*: *sound* *Q*

and *s-R*: *sound* *R*

shows $P \Vdash wp \ (a ;;) \ b \ R$

proof –

note *ha* = *well-def-wp-healthy*[*OF* *wa*]

note *hb* = *well-def-wp-healthy*[*OF* *wb*]

note *ent-a*

also from *ent-b* *ha* *hb* *s-Q* *s-R* **have** $wp \ a \ Q \Vdash wp \ a \ (wp \ b \ R)$

by(*blast* *intro:healthy-monoD2*)

finally show *?thesis* **by**(*simp* *add:wp-eval*)

qed

lemma *wlp-Seq*:

assumes *ent-a*: $P \Vdash wlp \ a \ Q$

and *ent-b*: $Q \Vdash wlp \ b \ R$

and *wa*: *well-def* *a*

and *wb*: *well-def* *b*

and *u-Q*: *unitary* *Q*

and *u-R*: *unitary* *R*

shows $P \Vdash wlp \ (a ;;) \ b \ R$

proof –

note *ha* = *well-def-wlp-nearly-healthy*[*OF* *wa*]

note *hb* = *well-def-wlp-nearly-healthy*[*OF* *wb*]

note *ent-a*

also from *ent-b* *ha* *hb* *u-Q* *u-R* **have** $wlp \ a \ Q \Vdash wlp \ a \ (wlp \ b \ R)$

by(*blast* *intro:nearly-healthy-monoD*[*OF* *ha*])

finally show *?thesis* **by**(*simp* *add:wp-eval*)

qed

lemma *wp-PC*:

$(\lambda s. P s * wp a Q s + (I - P s) * wp b Q s) \Vdash wp (a \oplus b) Q$
by(*simp add:wp-eval*)

lemma *wlp-PC*:

$(\lambda s. P s * wlp a Q s + (I - P s) * wlp b Q s) \Vdash wlp (a \oplus b) Q$
by(*simp add:wp-eval*)

A simpler rule for when the probability does not depend on the state.

lemma *PC-fixed*:

assumes *wpa*: $P \Vdash a \text{ ab } R$
and *wpb*: $Q \Vdash b \text{ ab } R$
and *np*: $0 \leq p$ **and** *bp*: $p \leq 1$
shows $(\lambda s. p * P s + (I - p) * Q s) \Vdash (a (\lambda s. p) \oplus b) \text{ ab } R$
unfolding *PC-def*
proof(*rule le-funI*)
fix *s*
from *wpa* **and** *np* **have** $p * P s \leq p * a \text{ ab } R s$
by(*auto intro:mult-left-mono*)
moreover {
from *bp* **have** $0 \leq 1 - p$ **by**(*simp*)
with *wpb* **have** $(I - p) * Q s \leq (I - p) * b \text{ ab } R s$
by(*auto intro:mult-left-mono*)
}
ultimately show $p * P s + (I - p) * Q s \leq$
 $p * a \text{ ab } R s + (I - p) * b \text{ ab } R s$
by(*rule add-mono*)
qed

lemma *wp-PC-fixed*:

$\llbracket P \Vdash wp a R; Q \Vdash wp b R; 0 \leq p; p \leq 1 \rrbracket \implies$
 $(\lambda s. p * P s + (I - p) * Q s) \Vdash wp (a (\lambda s. p) \oplus b) R$
by(*simp add:wp-def PC-fixed*)

lemma *wlp-PC-fixed*:

$\llbracket P \Vdash wlp a R; Q \Vdash wlp b R; 0 \leq p; p \leq 1 \rrbracket \implies$
 $(\lambda s. p * P s + (I - p) * Q s) \Vdash wlp (a (\lambda s. p) \oplus b) R$
by(*simp add:wlp-def PC-fixed*)

lemma *wp-DC*:

$(\lambda s. \min (wp a Q s) (wp b Q s)) \Vdash wp (a \sqcap b) Q$
unfolding *wp-eval* **by**(*simp*)

lemma *wlp-DC*:

$(\lambda s. \min (wlp a Q s) (wlp b Q s)) \Vdash wlp (a \sqcap b) Q$
unfolding *wp-eval* **by**(*simp*)

Combining annotations for both branches:

lemma *DC-split*:

fixes $a::'s \text{ prog}$ **and** b

assumes $wpa: P \Vdash a \text{ ab } R$

and $wpb: Q \Vdash b \text{ ab } R$

shows $(\lambda s. \min (P s) (Q s)) \Vdash (a \sqcap b) \text{ ab } R$

unfolding *DC-def*

proof(*rule le-funI*)

fix s

from $wpa \text{ } wpb$

have $P s \leq a \text{ ab } R s$ **and** $Q s \leq b \text{ ab } R s$ **by**(*auto*)

thus $\min (P s) (Q s) \leq \min (a \text{ ab } R s) (b \text{ ab } R s)$ **by**(*auto*)

qed

lemma *wp-DC-split*:

$\llbracket P \Vdash wp \text{ prog } R; Q \Vdash wp \text{ prog}' R \rrbracket \implies$

$(\lambda s. \min (P s) (Q s)) \Vdash wp (\text{prog} \sqcap \text{prog}') R$

by(*simp add:wp-def DC-split*)

lemma *wlp-DC-split*:

$\llbracket P \Vdash wlp \text{ prog } R; Q \Vdash wlp \text{ prog}' R \rrbracket \implies$

$(\lambda s. \min (P s) (Q s)) \Vdash wlp (\text{prog} \sqcap \text{prog}') R$

by(*simp add:wlp-def DC-split*)

lemma *wp-DC-split-same*:

$\llbracket P \Vdash wp \text{ prog } Q; P \Vdash wp \text{ prog}' Q \rrbracket \implies P \Vdash wp (\text{prog} \sqcap \text{prog}') Q$

unfolding *wp-eval* **by**(*blast intro:min.boundedI*)

lemma *wlp-DC-split-same*:

$\llbracket P \Vdash wlp \text{ prog } Q; P \Vdash wlp \text{ prog}' Q \rrbracket \implies P \Vdash wlp (\text{prog} \sqcap \text{prog}') Q$

unfolding *wp-eval* **by**(*blast intro:min.boundedI*)

lemma *SetPC-split*:

fixes $f::'x \Rightarrow 'y \text{ prog}$

and $p::'y \Rightarrow 'x \Rightarrow \text{real}$

assumes $rec: \bigwedge x s. x \in \text{supp} (p s) \implies P x \Vdash f x \text{ ab } Q$

and $nnp: \bigwedge s. \text{nneg} (p s)$

shows $(\lambda s. \sum x \in \text{supp} (p s). p s x * P x s) \Vdash \text{SetPC } f p \text{ ab } Q$

unfolding *SetPC-def*

proof(*rule le-funI*)

fix s

from rec **have** $\bigwedge x. x \in \text{supp} (p s) \implies P x s \leq f x \text{ ab } Q s$ **by**(*blast*)

moreover from nnp **have** $\bigwedge x. 0 \leq p s x$ **by**(*blast*)

ultimately have $\bigwedge x. x \in \text{supp} (p s) \implies p s x * P x s \leq p s x * f x \text{ ab } Q s$

by(*blast intro:mult-left-mono*)

thus $(\sum x \in \text{supp} (p s). p s x * P x s) \leq (\sum x \in \text{supp} (p s). p s x * f x \text{ ab } Q s)$

by(*rule sum-mono*)

qed

lemma *wp-SetPC-split*:

$$\begin{aligned} & \llbracket \bigwedge x s. x \in \text{supp } (p s) \implies P x \Vdash \text{wp } (f x) Q; \bigwedge s. \text{nneg } (p s) \rrbracket \implies \\ & (\lambda s. \sum x \in \text{supp } (p s). p s x * P x s) \Vdash \text{wp } (\text{SetPC } f p) Q \\ & \text{by}(\text{simp add:wp-def SetPC-split}) \end{aligned}$$

lemma *wlp-SetPC-split*:

$$\begin{aligned} & \llbracket \bigwedge x s. x \in \text{supp } (p s) \implies P x \Vdash \text{wlp } (f x) Q; \bigwedge s. \text{nneg } (p s) \rrbracket \implies \\ & (\lambda s. \sum x \in \text{supp } (p s). p s x * P x s) \Vdash \text{wlp } (\text{SetPC } f p) Q \\ & \text{by}(\text{simp add:wlp-def SetPC-split}) \end{aligned}$$

lemma *wp-SetDC-split*:

$$\begin{aligned} & \llbracket \bigwedge s x. x \in S s \implies P \Vdash \text{wp } (f x) Q; \bigwedge s. S s \neq \{\} \rrbracket \implies \\ & P \Vdash \text{wp } (\text{SetDC } f S) Q \\ & \text{by}(\text{rule le-funI, unfold wp-eval, blast intro!:cInf-greatest}) \end{aligned}$$

lemma *wlp-SetDC-split*:

$$\begin{aligned} & \llbracket \bigwedge s x. x \in S s \implies P \Vdash \text{wlp } (f x) Q; \bigwedge s. S s \neq \{\} \rrbracket \implies \\ & P \Vdash \text{wlp } (\text{SetDC } f S) Q \\ & \text{by}(\text{rule le-funI, unfold wp-eval, blast intro!:cInf-greatest}) \end{aligned}$$

lemma *wp-SetDC*:

assumes *wp*: $\bigwedge s x. x \in S s \implies P x \Vdash \text{wp } (f x) Q$
and *ne*: $\bigwedge s. S s \neq \{\}$
and *sP*: $\bigwedge x. \text{sound } (P x)$
shows $(\lambda s. \text{Inf } ((\lambda x. P x s) \text{ ` } S s)) \Vdash \text{wp } (\text{SetDC } f S) Q$
using *assms* **by**(*intro le-funI, simp add:wp-eval, blast intro!:cInf-mono*)

lemma *wlp-SetDC*:

assumes *wp*: $\bigwedge s x. x \in S s \implies P x \Vdash \text{wlp } (f x) Q$
and *ne*: $\bigwedge s. S s \neq \{\}$
and *sP*: $\bigwedge x. \text{sound } (P x)$
shows $(\lambda s. \text{Inf } ((\lambda x. P x s) \text{ ` } S s)) \Vdash \text{wlp } (\text{SetDC } f S) Q$
using *assms* **by**(*intro le-funI, simp add:wp-eval, blast intro!:cInf-mono*)

lemma *wp-Embed*:

$P \Vdash t Q \implies P \Vdash \text{wp } (\text{Embed } t) Q$
by(*simp add:wp-def Embed-def*)

lemma *wlp-Embed*:

$P \Vdash t Q \implies P \Vdash \text{wlp } (\text{Embed } t) Q$
by(*simp add:wlp-def Embed-def*)

lemma *wp-Bind*:

$\llbracket \bigwedge s. P s \leq \text{wp } (a (f s)) Q s \rrbracket \implies P \Vdash \text{wp } (\text{Bind } f a) Q$
by(*auto simp:wp-def Bind-def*)

lemma *wlp-Bind*:

$\llbracket \bigwedge s. P s \leq \text{wlp } (a (f s)) Q s \rrbracket \implies P \Vdash \text{wlp } (\text{Bind } f a) Q$
by(*auto simp:wlp-def Bind-def*)

lemma *wp-repeat*:

$$\begin{aligned} & \llbracket P \Vdash wp\ a\ Q; Q \Vdash wp\ (\text{repeat } n\ a)\ R; \\ & \text{well-def } a; \text{ sound } Q; \text{ sound } R \rrbracket \Longrightarrow P \Vdash wp\ (\text{repeat } (\text{Suc } n)\ a)\ R \\ & \mathbf{by}(\text{auto intro!:wp-Seq wd-intros}) \end{aligned}$$

lemma *wlp-repeat*:

$$\begin{aligned} & \llbracket P \Vdash wlp\ a\ Q; Q \Vdash wlp\ (\text{repeat } n\ a)\ R; \\ & \text{well-def } a; \text{ unitary } Q; \text{ unitary } R \rrbracket \Longrightarrow P \Vdash wlp\ (\text{repeat } (\text{Suc } n)\ a)\ R \\ & \mathbf{by}(\text{auto intro!:wlp-Seq wd-intros}) \end{aligned}$$

Note that the loop rules presented in section [Section 4.8](#) are of the same form, and would belong here, had they not already been stated.

The following rules are specialisations of those for general transformers, and are easier for the unifier to match.

lemmas *wp-strengthen-post=*

entails-strengthen-post[**where** $t=wp\ a\ \mathbf{for}\ a$]

lemma *wlp-strengthen-post*:

$$\begin{aligned} & P \Vdash wlp\ a\ Q \Longrightarrow \text{nearly-healthy } (wlp\ a) \Longrightarrow \text{unitary } R \Longrightarrow Q \Vdash R \Longrightarrow \text{unitary } Q \Longrightarrow \\ & P \Vdash wlp\ a\ R \\ & \mathbf{by}(\text{blast intro:entails-trans}) \end{aligned}$$

lemmas *wp-weaken-pre=*

entails-weaken-pre[**where** $t=wp\ a\ \mathbf{for}\ a$]

lemmas *wlp-weaken-pre=*

entails-weaken-pre[**where** $t=wlp\ a\ \mathbf{for}\ a$]

lemmas *wp-scale=*

entails-scale[**where** $t=wp\ a\ \mathbf{for}\ a$, *OF - well-def-wp-healthy*]

4.10.2 Algebraic Decomposition

Refinement is a powerful tool for decomposition, belied by the simplicity of the rule. This is an *axiomatic* formulation of refinement (all annotations of the a are annotations of b), rather than an operational version (all traces of b are traces of a).

lemma *wp-refines*:

$$\begin{aligned} & \llbracket a \sqsubseteq b; P \Vdash wp\ a\ Q; \text{ sound } Q \rrbracket \Longrightarrow P \Vdash wp\ b\ Q \\ & \mathbf{by}(\text{auto intro:entails-trans}) \end{aligned}$$

lemmas *wp-drefines = drefinesD*

4.10.3 Hoare triples

The Hoare triple, or validity predicate, is logically equivalent to the weakest-precondition entailment form. The benefit is that it allows us to define transitivity rules for computational (also/finally) reasoning.

definition

$$wp\text{-valid} :: ('a \Rightarrow real) \Rightarrow 'a \text{ prog} \Rightarrow ('a \Rightarrow real) \Rightarrow bool (\{-\} - \{-\}p)$$
where

$$wp\text{-valid } P \text{ prog } Q \equiv P \Vdash wp \text{ prog } Q$$
lemma *wp-validI*:
$$P \Vdash wp \text{ prog } Q \Longrightarrow \{P\} \text{ prog } \{Q\}p$$
unfolding *wp-valid-def* **by** (*assumption*)
lemma *wp-validD*:
$$\{P\} \text{ prog } \{Q\}p \Longrightarrow P \Vdash wp \text{ prog } Q$$
unfolding *wp-valid-def* **by** (*assumption*)
lemma *valid-Seq*:
$$\llbracket \{P\} a \{Q\}p; \{Q\} b \{R\}p; \text{well-def } a; \text{well-def } b; \text{sound } Q; \text{sound } R \rrbracket \Longrightarrow$$

$$\{P\} a ;; b \{R\}p$$
unfolding *wp-valid-def* **by** (*rule wp-Seq*)

We make it available to the computational reasoner:

declare *valid-Seq*[*trans*]

end

4.11 Loop Termination

theory *Termination* **imports** *Embedding StructuredReasoning Loops* **begin**

Termination for loops can be shown by classical means (using a variant, or a measure function), or by probabilistic means: We only need that the loop terminates *with probability one*.

4.11.1 Trivial Termination

A maximal transformer (program) doesn't affect termination. This is essentially saying that such a program doesn't abort (or diverge).

lemma *maximal-Seq-term*:
fixes *r*:: '*s* prog **and** *s*:: '*s* prog

assumes *mr*: *maximal* (*wp r*)

and *ws*: *well-def s*
and *ts*: $(\lambda s. I) \Vdash wp \ s \ (\lambda s. I)$
shows $(\lambda s. I) \Vdash wp \ (r ;; s) \ (\lambda s. I)$
proof –

note *hs* = *well-def-wp-healthy*[*OF ws*]

have $wp \ s \ (\lambda s. I) = (\lambda s. I)$
proof (*rule antisym*)

show $(\lambda s. I) \Vdash wp \ s \ (\lambda s. I)$ **by** (*rule ts*)

have *bounded-by* *I* (*wp s* ($\lambda s. I$))

```

by(auto intro! : healthy-bounded-byD[OF hs])
thus wp s ( $\lambda s. I$ )  $\Vdash$  ( $\lambda s. I$ ) by(auto intro! : le-funI)
qed
with mr show ?thesis
by(simp add : wp-eval embed-bool-def maximalD)
qed

```

From any state where the guard does not hold, a loop terminates in a single step.

```

lemma term-onestep:
assumes wb : well-def body
shows  $\langle\mathcal{N} G\rangle \Vdash wp \text{ do } G \longrightarrow body \text{ od } (\lambda s. I)$ 
proof(rule le-funI)
note hb = well-def-wp-healthy[OF wb]
fix s
show  $\langle\mathcal{N} G\rangle s \leq wp \text{ do } G \longrightarrow body \text{ od } (\lambda s. I) s$ 
proof(cases G s, simp-all add : wp-loop-nguard hb)
from hb have sound ( $wp \text{ do } G \longrightarrow body \text{ od } (\lambda s. I)$ )
by(auto intro : healthy-sound[OF healthy-wp-loop])
thus  $0 \leq wp \text{ do } G \longrightarrow body \text{ od } (\lambda s. I) s$  by(auto)
qed
qed

```

4.11.2 Classical Termination

The first non-trivial termination result is quite standard: If we can provide a natural-number-valued measure, that decreases on every iteration, and implies termination on reaching zero, the loop terminates.

```

lemma loop-term-nat-measure-noinv:
fixes m :: 's  $\Rightarrow$  nat and body :: 's prog
assumes wb : well-def body
and guard :  $\bigwedge s. m s = 0 \longrightarrow \neg G s$ 
and variant :  $\bigwedge n. \langle\lambda s. m s = Suc n\rangle \Vdash wp \text{ body } \langle\lambda s. m s = n\rangle$ 
shows  $\lambda s. I \Vdash wp \text{ do } G \longrightarrow body \text{ od } (\lambda s. I)$ 
proof –
note hb = well-def-wp-healthy[OF wb]
have  $\bigwedge n. (\forall s. m s = n \longrightarrow I \leq wp \text{ do } G \longrightarrow body \text{ od } (\lambda s. I) s)$ 
proof(induct-tac n)
fix n
show  $\forall s. m s = 0 \longrightarrow I \leq wp \text{ do } G \longrightarrow body \text{ od } (\lambda s. I) s$ 
proof(clarify)
fix s
assume m s = 0
with guard have  $\neg G s$  by(blast)
with hb show  $I \leq wp \text{ do } G \longrightarrow body \text{ od } (\lambda s. I) s$ 
by(simp add : wp-loop-nguard)
qed
assume IH :  $\forall s. m s = n \longrightarrow I \leq wp \text{ do } G \longrightarrow body \text{ od } (\lambda s. I) s$ 
hence IH' :  $\forall s. m s = n \longrightarrow I \leq wp \text{ do } G \longrightarrow body \text{ od } \langle\lambda s. True\rangle s$ 

```

```

  by(simp add:embed-bool-def)
have  $\forall s. m s = \text{Suc } n \longrightarrow I \leq \text{wp do } G \longrightarrow \text{body od } \langle \lambda s. \text{True} \rangle s$ 
proof(intro fold-premise healthy-intros hb, rule le-funI)
  fix s
  show  $\langle \lambda s. m s = \text{Suc } n \rangle s \leq \text{wp do } G \longrightarrow \text{body od } \langle \lambda s. \text{True} \rangle s$ 
  proof(cases G s)
    case False
    hence  $I = \langle \mathcal{N} G \rangle s$  by(auto)
    also from wb have  $\dots \leq \text{wp do } G \longrightarrow \text{body od } (\lambda s. I) s$ 
      by(rule le-funD[OF term-onestep])
    finally show ?thesis by(simp add:embed-bool-def)
  next
  case True note  $G = \text{this}$ 
  from IH' have  $\langle \lambda s. m s = n \rangle \Vdash \text{wp do } G \longrightarrow \text{body od } \langle \lambda s. \text{True} \rangle$ 
    by(blast intro:use-premise healthy-intros hb)
  with variant wb
  have  $\langle \lambda s. m s = \text{Suc } n \rangle \Vdash \text{wp } (\text{body} ;; \text{do } G \longrightarrow \text{body od}) \langle \lambda s. \text{True} \rangle$ 
    by(blast intro:wp-Seq wd-intros)
  hence  $\langle \lambda s. m s = \text{Suc } n \rangle s \leq \text{wp } (\text{body} ;; \text{do } G \longrightarrow \text{body od}) \langle \lambda s. \text{True} \rangle s$ 
    by(auto)
  also from hb G have  $\dots = \text{wp do } G \longrightarrow \text{body od } \langle \lambda s. \text{True} \rangle s$ 
    by(simp add:wp-loop-guard)
  finally show ?thesis .
qed
qed
thus  $\forall s. m s = \text{Suc } n \longrightarrow I \leq \text{wp do } G \longrightarrow \text{body od } (\lambda s. I) s$ 
  by(simp add:embed-bool-def)
qed
thus ?thesis by(auto)
qed

```

This version allows progress to depend on an invariant. Termination is then determined by the invariant's value in the initial state.

lemma *loop-term-nat-measure*:

```

fixes  $m :: 's \Rightarrow \text{nat}$  and  $\text{body} :: 's \text{ prog}$ 
assumes wb: well-def body
and guard:  $\bigwedge s. m s = 0 \longrightarrow \neg G s$ 
and variant:  $\bigwedge n. \langle \lambda s. m s = \text{Suc } n \rangle \&\& \langle I \rangle \Vdash \text{wp body } \langle \lambda s. m s = n \rangle$ 
and inv:  $\text{wp-inv } G \text{ body } \langle I \rangle$ 
shows  $\langle I \rangle \Vdash \text{wp do } G \longrightarrow \text{body od } (\lambda s. I)$ 
proof –
  note hb = well-def-wp-healthy[OF wb]
  note scb = sublinear-sub-conj[OF well-def-wp-sublinear, OF wb]
  have  $\langle I \rangle \Vdash \text{wp do } G \longrightarrow \text{body od } \langle \lambda s. \text{True} \rangle$ 
  proof(rule use-premise, intro healthy-intros hb)
    fix s
    have  $\bigwedge n. (\forall s. m s = n \wedge I s \longrightarrow I \leq \text{wp do } G \longrightarrow \text{body od } \langle \lambda s. \text{True} \rangle s)$ 
    proof(induct-tac n)
      fix n

```

```

show  $\forall s. m\ s = 0 \wedge I\ s \longrightarrow I \leq wp\ do\ G \longrightarrow body\ od\ \langle\lambda s. True\rangle\ s$ 
proof(clarify)
  fix s
  assume  $m\ s = 0$ 
  with guard have  $\neg G\ s$  by(blast)
  with hb show  $I \leq wp\ do\ G \longrightarrow body\ od\ \langle\lambda s. True\rangle\ s$ 
  by(simp add:wp-loop-nguard)
qed
assume IH:  $\forall s. m\ s = n \wedge I\ s \longrightarrow I \leq wp\ do\ G \longrightarrow body\ od\ \langle\lambda s. True\rangle\ s$ 
show  $\forall s. m\ s = Suc\ n \wedge I\ s \longrightarrow I \leq wp\ do\ G \longrightarrow body\ od\ \langle\lambda s. True\rangle\ s$ 
proof(intro fold-premise healthy-intros hb le-funI)
  fix s
  show  $\langle\lambda s. m\ s = Suc\ n \wedge I\ s\rangle\ s \leq wp\ do\ G \longrightarrow body\ od\ \langle\lambda s. True\rangle\ s$ 
  proof(cases G s)
    case False with hb show ?thesis
      by(simp add:wp-loop-nguard)
  next
    case True note  $G = this$ 
    have  $\langle\lambda s. m\ s = Suc\ n\rangle\ \&\&\ \langle I \rangle\ \&\&\ \langle G \rangle =$ 
       $\langle\lambda s. m\ s = Suc\ n\rangle\ \&\&\ (\langle I \rangle\ \&\&\ \langle I \rangle)\ \&\&\ \langle G \rangle$ 
      by(simp)
    also have  $\dots = (\langle\lambda s. m\ s = Suc\ n\rangle\ \&\&\ \langle I \rangle)\ \&\&\ (\langle I \rangle\ \&\&\ \langle G \rangle)$ 
      by(simp add:exp-conj-assoc exp-conj-unitary del:exp-conj-idem)
    also have  $\dots = (\langle\lambda s. m\ s = Suc\ n\rangle\ \&\&\ \langle I \rangle)\ \&\&\ (\langle G \rangle\ \&\&\ \langle I \rangle)$ 
      by(simp only:exp-conj-comm)
    also {
      from inv hb have  $\langle G \rangle\ \&\&\ \langle I \rangle \Vdash wp\ body\ \langle I \rangle$ 
      by(rule wp-inv-stdD)
      with variant
      have  $(\langle\lambda s. m\ s = Suc\ n\rangle\ \&\&\ \langle I \rangle)\ \&\&\ (\langle G \rangle\ \&\&\ \langle I \rangle) \Vdash$ 
         $wp\ body\ \langle\lambda s. m\ s = n\rangle\ \&\&\ wp\ body\ \langle I \rangle$ 
      by(rule entails-frame)
    }
    also from scb
    have  $wp\ body\ \langle\lambda s. m\ s = n\rangle\ \&\&\ wp\ body\ \langle I \rangle \Vdash$ 
       $wp\ body\ (\langle\lambda s. m\ s = n\rangle\ \&\&\ \langle I \rangle)$ 
      by(blast)
    finally have  $\langle\lambda s. m\ s = Suc\ n\rangle\ \&\&\ \langle I \rangle\ \&\&\ \langle G \rangle \Vdash$ 
       $wp\ body\ (\langle\lambda s. m\ s = n\rangle\ \&\&\ \langle I \rangle)$ 
    moreover {
      from IH have  $\langle\lambda s. m\ s = n \wedge I\ s\rangle \Vdash wp\ do\ G \longrightarrow body\ od\ \langle\lambda s. True\rangle$ 
      by(blast intro:use-premise healthy-intros hb)
      hence  $\langle\lambda s. m\ s = n\rangle\ \&\&\ \langle I \rangle \Vdash wp\ do\ G \longrightarrow body\ od\ \langle\lambda s. True\rangle$ 
      by(simp add:exp-conj-std-split)
    }
  ultimately
  have  $\langle\lambda s. m\ s = Suc\ n\rangle\ \&\&\ \langle I \rangle\ \&\&\ \langle G \rangle \Vdash$ 
     $wp\ (body\ ;;\ do\ G \longrightarrow body\ od)\ \langle\lambda s. True\rangle$ 
  using wb by(blast intro:wp-Seq wd-intros)

```



```

hence ( $\llbracket \lambda s. m\ s = \text{Suc } n \wedge I\ s \rrbracket \&\& \llbracket G \rrbracket$ )  $s \leq$ 
   $wp\ (body\ ;;\ do\ G \longrightarrow body\ od)\ \llbracket \lambda s. True \rrbracket\ s$ 
by(auto simp:exp-conj-std-split)
with  $G$  have  $\llbracket \lambda s. m\ s = \text{Suc } n \wedge I\ s \rrbracket\ s \leq$ 
   $wp\ (body\ ;;\ do\ G \longrightarrow body\ od)\ \llbracket \lambda s. True \rrbracket\ s$ 
by(simp add:exp-conj-def)
also from  $hb\ G$  have  $\dots = wp\ do\ G \longrightarrow body\ od\ \llbracket \lambda s. True \rrbracket\ s$ 
by(simp add:wp-loop-guard)
finally show ?thesis .
qed
qed
qed
moreover assume  $I\ s$ 
ultimately show  $I \leq wp\ do\ G \longrightarrow body\ od\ \llbracket \lambda s. True \rrbracket\ s$ 
by(auto)
qed
thus ?thesis by(simp add:embed-bool-def)
qed

```

4.11.3 Probabilistic Termination

Any loop that has a non-zero chance of terminating after each step terminates with probability 1.

lemma *termination-0-1*:

fixes $body :: 's\ prog$

assumes $wb: well-def\ body$

— The loop terminates in one step with nonzero probability

and $onestep: (\lambda s. p) \Vdash wp\ body\ \llbracket \mathcal{N}\ G \rrbracket$

and $nzp: 0 < p$

— The body is maximal i.e. it terminates absolutely.

and $mb: maximal\ (wp\ body)$

shows $\lambda s. I \Vdash wp\ do\ G \longrightarrow body\ od\ (\lambda s. I)$

proof —

note $hb = well-def-wp-healthy[OF\ wb]$

note $sb = healthy-scalingD[OF\ hb]$

note $sab = sublinear-subadd[OF\ well-def-wp-sublinear,\ OF\ wb,\ OF\ healthy-feasibleD,\ OF\ hb]$

from hb **have** $hloop: healthy\ (wp\ do\ G \longrightarrow body\ od)$

by(*rule healthy-intros*)

hence $swp: sound\ (wp\ do\ G \longrightarrow body\ od\ (\lambda s. I))$ **by**(*blast*)

p is no greater than 1, by feasibility.

from $onestep$ **have** $onestep': \bigwedge s. p \leq wp\ body\ \llbracket \mathcal{N}\ G \rrbracket\ s$ **by**(*auto*)

also {

from hb **have** $unitary\ (wp\ body\ \llbracket \mathcal{N}\ G \rrbracket)$ **by**(*auto*)

hence $\bigwedge s. wp\ body\ \llbracket \mathcal{N}\ G \rrbracket\ s \leq I$ **by**(*auto*)

}

finally have $p1: p \leq I$.

This is the crux of the proof: that given a lower bound below 1, we can find another, higher one.

have *new-bound*: $\bigwedge k. 0 \leq k \implies k \leq 1 \implies (\lambda s. k) \Vdash \text{wp do } G \longrightarrow \text{body od } (\lambda s. 1) \implies$
 $(\lambda s. p * (1-k) + k) \Vdash \text{wp do } G \longrightarrow \text{body od } (\lambda s. 1)$

proof(*rule le-funI*)

fix $k\ s$

assume $X: \lambda s. k \Vdash \text{wp do } G \longrightarrow \text{body od } (\lambda s. 1)$

and $k0: 0 \leq k$ **and** $k1: k \leq 1$

from $k1$ **have** $\text{nz1k}: 0 \leq 1 - k$ **by**(*auto*)

with $p1$ **have** $p * (1-k) + k \leq 1 * (1-k) + k$

by(*blast intro:mult-right-mono add-mono*)

hence $p * (1 - k) + k \leq 1$

by(*simp*)

The new bound is $p * (1 - k) + k$.

hence $p * (1-k) + k \leq \llbracket \mathcal{N} G \rrbracket s + \llbracket G \rrbracket s * (p * (1-k) + k)$

by(*cases G s, simp-all*)

By the one-step termination assumption:

also from *onestep'* nz1k

have $\dots \leq \llbracket \mathcal{N} G \rrbracket s + \llbracket G \rrbracket s * (\text{wp body } \llbracket \mathcal{N} G \rrbracket s * (1-k) + k)$

by (*simp add: mult-right-mono ordered-comm-semiring-class.comm-mult-left-mono*)

By scaling:

also from nz1k

have $\dots = \llbracket \mathcal{N} G \rrbracket s + \llbracket G \rrbracket s * (\text{wp body } (\lambda s. \llbracket \mathcal{N} G \rrbracket s * (1-k)) s + k)$

by(*simp add:right-scalingD[OF sb]*)

By the maximality (termination) of the loop body:

also from $\text{mb } k0$

have $\dots = \llbracket \mathcal{N} G \rrbracket s + \llbracket G \rrbracket s * (\text{wp body } (\lambda s. \llbracket \mathcal{N} G \rrbracket s * (1-k)) s + \text{wp body } (\lambda s. k) s)$

by(*simp add:maximalD*)

By sub-additivity of the loop body:

also from $k0$ nz1k

have $\dots \leq \llbracket \mathcal{N} G \rrbracket s + \llbracket G \rrbracket s * (\text{wp body } (\lambda s. \llbracket \mathcal{N} G \rrbracket s * (1-k) + k) s)$

by(*auto intro!:add-left-mono mult-left-mono sub-addD[OF sab] sound-intros*)

also

have $\dots = \llbracket \mathcal{N} G \rrbracket s + \llbracket G \rrbracket s * (\text{wp body } (\lambda s. \llbracket \mathcal{N} G \rrbracket s + \llbracket G \rrbracket s * k) s)$

by(*simp add:negate-embed algebra-simps*)

By monotonicity of the loop body, and that k is a lower bound:

also from $k0$ *hloop le-funD*[*OF X*]

have $\dots \leq \llbracket \mathcal{N} G \rrbracket s +$

$\llbracket G \rrbracket s * (\text{wp body } (\lambda s. \llbracket \mathcal{N} G \rrbracket s + \llbracket G \rrbracket s * \text{wp do } G \longrightarrow \text{body od } (\lambda s. 1) s) s)$

by(*iprover intro:add-left-mono mult-left-mono le-funI embed-ge-0*
le-funD[*OF mono-transD, OF healthy-monoD, OF hb*])

sound-sum standard-sound sound-intros swp)

Unrolling the loop once and simplifying:

```

also {
  have  $\bigwedge s. \langle \mathcal{N} G \rangle s + \langle G \rangle s * wp \text{ body } (wp \text{ do } G \longrightarrow \text{body od } (\lambda s. I)) s =$ 
     $\langle \mathcal{N} G \rangle s + \langle G \rangle s * (\langle \mathcal{N} G \rangle s + \langle G \rangle s * wp \text{ body } (wp \text{ do } G \longrightarrow \text{body od } (\lambda s. I)) s)$ 
  by(simp only:distrib-left mult.assoc[symmetric] embed-bool-idem embed-bool-cancel)
  also have  $\bigwedge s. \dots s = \langle \mathcal{N} G \rangle s + \langle G \rangle s * wp \text{ do } G \longrightarrow \text{body od } (\lambda s. I) s$ 
  by(simp add:fun-cong[OF wp-loop-unfold[symmetric, where P= $\lambda s. I$ , simplified, OF hb]])
  finally have  $X: \bigwedge s. \langle \mathcal{N} G \rangle s + \langle G \rangle s * wp \text{ body } (wp \text{ do } G \longrightarrow \text{body od } (\lambda s. I)) s =$ 
     $\langle \mathcal{N} G \rangle s + \langle G \rangle s * wp \text{ do } G \longrightarrow \text{body od } (\lambda s. I) s .$ 
  have  $\langle \mathcal{N} G \rangle s + \langle G \rangle s * (wp \text{ body } (\lambda s. \langle \mathcal{N} G \rangle s + \langle G \rangle s * wp \text{ do } G \longrightarrow \text{body od } (\lambda s. I)) s) =$ 
     $\langle \mathcal{N} G \rangle s + \langle G \rangle s * (wp \text{ body } (\lambda s. \langle \mathcal{N} G \rangle s + \langle G \rangle s * wp \text{ body } (wp \text{ do } G \longrightarrow \text{body od } (\lambda s. I)) s) s)$ 
  by(simp only:X)
}

```

Lastly, by folding two loop iterations:

```

also
have  $\langle \mathcal{N} G \rangle s + \langle G \rangle s * (wp \text{ body } (\lambda s. \langle \mathcal{N} G \rangle s + \langle G \rangle s * wp \text{ body } (wp \text{ do } G \longrightarrow \text{body od } (\lambda s. I)) s) s) =$ 
   $wp \text{ do } G \longrightarrow \text{body od } (\lambda s. I) s$ 
by(simp add:wp-loop-unfold[OF - hb, where P= $\lambda s. I$ , simplified, symmetric] fun-cong[OF wp-loop-unfold[OF - hb, where P= $\lambda s. I$ , simplified, symmetric]])
finally show  $p * (I - k) + k \leq wp \text{ do } G \longrightarrow \text{body od } (\lambda s. I) s .$ 
qed

```

If the previous bound lay in $[0, 1)$, the new bound is strictly greater. This is where we appeal to the fact that p is nonzero.

```

from nzp have inc:  $\bigwedge k. 0 \leq k \implies k < I \implies k < p * (I - k) + k$ 
by(auto intro:mult-pos-pos)

```

The result follows by contradiction.

```

show ?thesis
proof(rule ccontr)

```

If the loop does not terminate everywhere, then there must exist some state from which the probability of termination is strictly less than one.

```

assume  $\neg ?thesis$ 
hence  $\neg (\forall s. I \leq wp \text{ do } G \longrightarrow \text{body od } (\lambda s. I) s)$  by(auto)
then obtain  $s$  where point:  $\neg I \leq wp \text{ do } G \longrightarrow \text{body od } (\lambda s. I) s$  by(auto)

```

```

let  $?k = \text{Inf } (\text{range } (wp \text{ do } G \longrightarrow \text{body od } (\lambda s. I)))$ 

```

```

from hloop
have InfIb:  $\bigwedge s. ?k \leq wp \text{ do } G \longrightarrow \text{body od } (\lambda s. I) s$ 
by(intro cInf-lower bdd-belowI, auto)

```

also from point have $wp\ do\ G \longrightarrow body\ od\ (\lambda s. I)\ s < I$ **by**(*auto*)

Thus the least (infimum) probability of termination is strictly less than one.

finally have $kI: ?k < I$.
hence $?k \leq I$ **by**(*auto*)
moreover from hloop have $k0: 0 \leq ?k$
by(*intro cInf-greatest, auto*)

The infimum is, naturally, a lower bound.

moreover from Infb have $(\lambda s. ?k) \Vdash wp\ do\ G \longrightarrow body\ od\ (\lambda s. I)$ **by**(*auto*)
ultimately

We can therefore use the previous result to find a new bound, ...

have $\bigwedge s. p * (I - ?k) + ?k \leq wp\ do\ G \longrightarrow body\ od\ (\lambda s. I)\ s$
by(*blast intro:le-funD[OF new-bound]*)

... which is lower than the infimum, by minimality, ...

hence $p * (I - ?k) + ?k \leq ?k$
by(*blast intro:cInf-greatest*)

... yet also strictly greater than it.

moreover from k0 kI have $?k < p * (I - ?k) + ?k$ **by**(*rule inc*)

We thus have a contradiction.

ultimately show *False* **by**(*simp*)
qed
qed
end

4.12 Automated Reasoning

theory *Automation* **imports** *StructuredReasoning*
begin

This theory serves as a container for automated reasoning tactics for pGCL, implemented in ML. At present, there is a basic verification condition generator (VCG).

named-theorems *wd*
theorems to automatically establish well-definedness
named-theorems *pwp-core*
core probabilistic wp rules, for evaluating primitive terms
named-theorems *pwp*
user-supplied probabilistic wp rules
named-theorems *pwlp*
user-supplied probabilistic wlp rules

ML-file $\langle pVCG.ML \rangle$

```
method-setup pvcg =  
  <Scan.succeed (fn ctxt => SIMPLE-METHOD' (pVCG.pVCG-tac ctxt))>  
  Probabilistic weakest preexpectation tactic  
  
declare wd-intros[wd]  
  
lemmas core-wp-rules =  
  wp-Skip      wlp-Skip  
  wp-Abort     wlp-Abort  
  wp-Apply     wlp-Apply  
  wp-Seq       wlp-Seq  
  wp-DC-split  wlp-DC-split  
  wp-PC-fixed  wlp-PC-fixed  
  wp-SetDC     wlp-SetDC  
  wp-SetPC-split wlp-SetPC-split  
  
declare core-wp-rules[pwp-core]  
  
end
```


Additional Material

4.13 Miscellaneous Mathematics

theory *Misc*

imports

HOL-Analysis.Multivariate-Analysis

begin lemma *sum-UNIV*:

fixes *S::'a::finite set*

assumes *complete*: $\bigwedge x. x \notin S \implies f x = 0$

shows $\text{sum } f S = \text{sum } f UNIV$

proof –

from *complete* **have** $\text{sum } f S = \text{sum } f (UNIV - S) + \text{sum } f S$ **by** (*simp*)

also have $\dots = \text{sum } f UNIV$

by (*auto intro: sum.subset-diff[symmetric]*)

finally show *?thesis* .

qed

lemma *cInf-mono*:

fixes *A::'a::conditionally-complete-lattice set*

assumes *lower*: $\bigwedge b. b \in B \implies \exists a \in A. a \leq b$

and *bounded*: $\bigwedge a. a \in A \implies c \leq a$

and *ne*: $B \neq \{\}$

shows $\text{Inf } A \leq \text{Inf } B$

proof (*rule cInf-greatest[OF ne]*)

fix *b* **assume** *bin*: $b \in B$

with *lower* **obtain** *a* **where** *ain*: $a \in A$ **and** *le*: $a \leq b$ **by** (*auto*)

from *ain bounded* **have** $\text{Inf } A \leq a$ **by** (*intro cInf-lower bdd-belowI, auto*)

also note *le*

finally show $\text{Inf } A \leq b$.

qed

lemma *max-distrib*:

fixes *c::real*

assumes *nn*: $0 \leq c$

shows $c * \text{max } a b = \text{max } (c * a) (c * b)$

proof (*cases a ≤ b*)

case *True*

moreover with *nn* **have** $c * a \leq c * b$ **by** (*auto intro:mult-left-mono*)

ultimately show *?thesis* **by** (*simp add:max.absorb2*)

next
case *False* **then have** $b \leq a$ **by**(*auto*)
moreover with *mn* **have** $c * b \leq c * a$ **by**(*auto intro:mult-left-mono*)
ultimately show *?thesis* **by**(*simp add:max.absorb1*)
qed

lemma *mult-div-mono-left*:
fixes *c::real*
assumes *nnc*: $0 \leq c$ **and** *nzc*: $c \neq 0$
and *inv*: $a \leq \text{inverse } c * b$
shows $c * a \leq b$
proof –
from *nnc inv* **have** $c * a \leq (c * \text{inverse } c) * b$
by(*auto simp:mult.assoc intro:mult-left-mono*)
also from *nzc* **have** $\dots = b$ **by**(*simp*)
finally show $c * a \leq b$.
qed

lemma *mult-div-mono-right*:
fixes *c::real*
assumes *nnc*: $0 \leq c$ **and** *nzc*: $c \neq 0$
and *inv*: $\text{inverse } c * a \leq b$
shows $a \leq c * b$
proof –
from *nzc* **have** $a = (c * \text{inverse } c) * a$ **by**(*simp*)
also from *nnc inv* **have** $(c * \text{inverse } c) * a \leq c * b$
by(*auto simp:mult.assoc intro:mult-left-mono*)
finally show $a \leq c * b$.
qed

lemma *min-distrib*:
fixes *c::real*
assumes *nnc*: $0 \leq c$
shows $c * \min a b = \min (c * a) (c * b)$
proof(*cases a ≤ b*)
case *True* **moreover with** *nnc* **have** $c * a \leq c * b$
by(*blast intro:mult-left-mono*)
ultimately show *?thesis* **by**(*auto*)
next
case *False* **hence** $b \leq a$ **by**(*auto*)
moreover with *nnc* **have** $c * b \leq c * a$
by(*blast intro:mult-left-mono*)
ultimately show *?thesis* **by**(*simp add:min.absorb2*)
qed

lemma *finite-set-least*:
fixes *S::'a::linorder set*
assumes *finite*: *finite S*
and *ne*: $S \neq \{\}$

shows $\exists x \in S. \forall y \in S. x \leq y$
proof –
have $S = \{\}$ $\vee (\exists x \in S. \forall y \in S. x \leq y)$
proof(*rule finite-induct, simp-all add:assms*)
fix $x::'a$ **and** $S::'a \text{ set}$
assume $IH: S = \{\} \vee (\exists x \in S. \forall y \in S. x \leq y)$
show $(\forall y \in S. x \leq y) \vee (\exists x' \in S. x' \leq x \wedge (\forall y \in S. x' \leq y))$
proof(*cases S={}*)
case *True* **then show** ?thesis **by**(*auto*)
next
case *False* **with** IH **have** $\exists x \in S. \forall y \in S. x \leq y$ **by**(*auto*)
then obtain z **where** $z \in S$ **and** $z \text{min}: \forall y \in S. z \leq y$ **by**(*auto*)
thus ?thesis **by**(*cases z ≤ x, auto*)
qed
qed
with ne **show** ?thesis **by**(*auto*)
qed

lemma *cSup-add*:
fixes $c::\text{real}$
assumes $ne: S \neq \{\}$
and $bS: \bigwedge x. x \in S \implies x \leq b$
shows $\text{Sup } S + c = \text{Sup } \{x + c \mid x. x \in S\}$
proof(*rule antisym*)
from ne bS **show** $\text{Sup } \{x + c \mid x. x \in S\} \leq \text{Sup } S + c$
by(*auto intro!: cSup-least add-right-mono cSup-upper bdd-aboveI*)

have $\text{Sup } S \leq \text{Sup } \{x + c \mid x. x \in S\} - c$
proof(*intro cSup-least ne*)
fix x **assume** $x \in S$
from bS **have** $\bigwedge x. x \in S \implies x + c \leq b + c$ **by**(*auto intro:add-right-mono*)
hence *bdd-above* $\{x + c \mid x. x \in S\}$ **by**(*intro bdd-aboveI, blast*)
with x **have** $x + c \leq \text{Sup } \{x + c \mid x. x \in S\}$ **by**(*auto intro:cSup-upper*)
thus $x \leq \text{Sup } \{x + c \mid x. x \in S\} - c$ **by**(*auto*)
qed
thus $\text{Sup } S + c \leq \text{Sup } \{x + c \mid x. x \in S\}$ **by**(*auto*)
qed

lemma *cSup-mult*:
fixes $c::\text{real}$
assumes $ne: S \neq \{\}$
and $bS: \bigwedge x. x \in S \implies x \leq b$
and $nnc: 0 \leq c$
shows $c * \text{Sup } S = \text{Sup } \{c * x \mid x. x \in S\}$
proof(*cases*)
assume $c = 0$
moreover from ne **have** $\exists x. x \in S$ **by**(*auto*)
ultimately show ?thesis **by**(*simp*)
next

```

assume  $cnz: c \neq 0$ 
show ?thesis
proof(rule antisym)
  from  $bS$  have  $baS: bdd\text{-above } S$  by(intro bdd-aboveI, auto)
  with  $ne$   $nnc$  show  $Sup \{c * x \mid x. x \in S\} \leq c * Sup S$ 
    by(blast intro!:cSup-least mult-left-mono[OF cSup-upper])
  have  $Sup S \leq inverse\ c * Sup \{c * x \mid x. x \in S\}$ 
  proof(intro cSup-least ne)
    fix  $x$  assume  $xin: x \in S$ 
    moreover from  $bS$   $nnc$  have  $\bigwedge x. x \in S \implies c * x \leq c * b$  by(auto intro:mult-left-mono)
    ultimately have  $c * x \leq Sup \{c * x \mid x. x \in S\}$ 
      by(auto intro!:cSup-upper bdd-aboveI)
    moreover from  $nnc$  have  $0 \leq inverse\ c$  by(auto)
    ultimately have  $inverse\ c * (c * x) \leq inverse\ c * Sup \{c * x \mid x. x \in S\}$ 
      by(auto intro:mult-left-mono)
    with  $cnz$  show  $x \leq inverse\ c * Sup \{c * x \mid x. x \in S\}$ 
      by(simp add:mult.assoc[symmetric])
  qed
  with  $nnc$  have  $c * Sup S \leq c * (inverse\ c * Sup \{c * x \mid x. x \in S\})$ 
    by(auto intro:mult-left-mono)
  with  $cnz$  show  $c * Sup S \leq Sup \{c * x \mid x. x \in S\}$ 
    by(simp add:mult.assoc[symmetric])
qed
qed

```

lemma *closure-contains-Sup*:

```

fixes  $S :: real\ set$ 
assumes  $neS: S \neq \{\}$  and  $bS: \forall x \in S. x \leq B$ 
shows  $Sup S \in closure\ S$ 
proof –
  let  $?T = uminus\ 'S$ 
  from  $neS$  have  $neT: ?T \neq \{\}$  by(auto)
  from  $bS$  have  $bT: \forall x \in ?T. -B \leq x$  by(auto)
  hence  $bbT: bdd\text{-below } ?T$  by(intro bdd-belowI, blast)

```

```

have  $Sup S = -\ Inf\ ?T$ 
proof(rule antisym)
  from  $neT$   $bbT$ 
  have  $\bigwedge x. x \in S \implies Inf\ (uminus\ 'S) \leq -x$ 
    by(blast intro:cInf-lower)
  hence  $\bigwedge x. x \in S \implies -I * -x \leq -I * Inf\ (uminus\ 'S)$ 
    by(rule mult-left-mono-neg, auto)
  hence  $lenInf: \bigwedge x. x \in S \implies x \leq -\ Inf\ (uminus\ 'S)$ 
    by(simp)
  with  $neS$   $bS$  show  $Sup S \leq -\ Inf\ ?T$ 
    by(blast intro:cSup-least)

```

```

have  $-\ Sup S \leq Inf\ ?T$ 
proof(rule cInf-greatest[OF neT])

```

```

fix  $x$  assume  $x \in \text{uminus } ' S$ 
then obtain  $y$  where  $y \in S$  and  $rx: x = -y$  by(auto)
from  $y \in S$  have  $y \leq \text{Sup } S$ 
by(intro cSup-upper bdd-belowI, auto)
hence  $-1 * \text{Sup } S \leq -1 * y$ 
by(simp add:mult-left-mono-neg)
with  $rx$  show  $-\text{Sup } S \leq x$  by(simp)
qed
hence  $-1 * \text{Inf } ?T \leq -1 * (-\text{Sup } S)$ 
by(simp add:mult-left-mono-neg)
thus  $-\text{Inf } ?T \leq \text{Sup } S$  by(simp)
qed
also {
from  $neT \text{ bbT}$  have  $\text{Inf } ?T \in \text{closure } ?T$  by(rule closure-contains-Inf)
hence  $-\text{Inf } ?T \in \text{uminus } ' \text{closure } ?T$  by(auto)
}
also {
have linear uminus by(auto intro:linearI)
hence  $\text{uminus } ' \text{closure } ?T \subseteq \text{closure } (\text{uminus } ' ?T)$ 
by(rule closure-linear-image-subset)
}
also {
have  $\text{uminus } ' ?T \subseteq S$  by(auto)
hence  $\text{closure } (\text{uminus } ' ?T) \subseteq \text{closure } S$  by(rule closure-mono)
}
finally show  $\text{Sup } S \in \text{closure } S$  .
qed

```

lemma *tendsto-min*:

```

fixes  $x y::\text{real}$ 
assumes  $ta: a \longrightarrow x$ 
and  $tb: b \longrightarrow y$ 
shows  $(\lambda i. \min (a i) (b i)) \longrightarrow \min x y$ 
proof(rule LIMSEQ-I, simp)
fix  $e::\text{real}$  assume  $pe: 0 < e$ 

from  $ta$  pe obtain  $noa$  where  $balla: \forall n \geq noa. \text{abs } (a n - x) < e$ 
by(auto dest:LIMSEQ-D)
from  $tb$   $pe$  obtain  $nob$  where  $ballb: \forall n \geq nob. \text{abs } (b n - y) < e$ 
by(auto dest:LIMSEQ-D)

```

```

{
fix  $n$ 
assume  $ge: \max noa nob \leq n$ 
hence  $gea: noa \leq n$  and  $geb: nob \leq n$  by(auto)
have  $\text{abs } (\min (a n) (b n) - \min x y) < e$ 
proof cases
assume  $le: \min (a n) (b n) \leq \min x y$ 
show ?thesis

```

proof cases

assume $a\ n \leq b\ n$

hence $rwmin: \min (a\ n) (b\ n) = a\ n$ **by**(*auto*)

with le **have** $a\ n \leq \min\ x\ y$ **by**(*simp*)

moreover from *gea balla* **have** $abs (a\ n - x) < e$ **by**(*simp*)

moreover have $\min\ x\ y \leq x$ **by**(*auto*)

ultimately have $abs (a\ n - \min\ x\ y) < e$ **by**(*auto*)

with $rwmin$ **show** $abs (\min (a\ n) (b\ n) - \min\ x\ y) < e$ **by**(*simp*)

next

assume $\neg a\ n \leq b\ n$

hence $b\ n \leq a\ n$ **by**(*auto*)

hence $rwmin: \min (a\ n) (b\ n) = b\ n$ **by**(*auto*)

with le **have** $b\ n \leq \min\ x\ y$ **by**(*simp*)

moreover from *geb ballb* **have** $abs (b\ n - y) < e$ **by**(*simp*)

moreover have $\min\ x\ y \leq y$ **by**(*auto*)

ultimately have $abs (b\ n - \min\ x\ y) < e$ **by**(*auto*)

with $rwmin$ **show** $abs (\min (a\ n) (b\ n) - \min\ x\ y) < e$ **by**(*simp*)

qed

next

assume $\neg \min (a\ n) (b\ n) \leq \min\ x\ y$

hence $le: \min\ x\ y \leq \min (a\ n) (b\ n)$ **by**(*auto*)

show *?thesis*

proof cases

assume $x \leq y$

hence $rwmin: \min\ x\ y = x$ **by**(*auto*)

with le **have** $x \leq \min (a\ n) (b\ n)$ **by**(*simp*)

moreover from *gea balla* **have** $abs (a\ n - x) < e$ **by**(*simp*)

moreover have $\min (a\ n) (b\ n) \leq a\ n$ **by**(*auto*)

ultimately have $abs (\min (a\ n) (b\ n) - x) < e$ **by**(*auto*)

with $rwmin$ **show** $abs (\min (a\ n) (b\ n) - \min\ x\ y) < e$ **by**(*simp*)

next

assume $\neg x \leq y$

hence $y \leq x$ **by**(*auto*)

hence $rwmin: \min\ x\ y = y$ **by**(*auto*)

with le **have** $y \leq \min (a\ n) (b\ n)$ **by**(*simp*)

moreover from *geb ballb* **have** $abs (b\ n - y) < e$ **by**(*simp*)

moreover have $\min (a\ n) (b\ n) \leq b\ n$ **by**(*auto*)

ultimately have $abs (\min (a\ n) (b\ n) - y) < e$ **by**(*auto*)

with $rwmin$ **show** $abs (\min (a\ n) (b\ n) - \min\ x\ y) < e$ **by**(*simp*)

qed

qed

}

thus $\exists n\ no. \forall n \geq no. |\min (a\ n) (b\ n) - \min\ x\ y| < e$ **by**(*blast*)

qed

definition $supp :: ('s \Rightarrow real) \Rightarrow 's\ set$

where $supp\ f = \{x. f\ x \neq 0\}$

definition $dist-remove :: ('s \Rightarrow real) \Rightarrow 's \Rightarrow 's \Rightarrow real$

where $\text{dist-remove } p \ x = (\lambda y. \text{if } y=x \text{ then } 0 \text{ else } p \ y / (1 - p \ x))$

lemma *supp-dist-remove*:

$p \ x \neq 0 \implies p \ x \neq 1 \implies \text{supp } (\text{dist-remove } p \ x) = \text{supp } p - \{x\}$
by(*auto simp:dist-remove-def supp-def*)

lemma *supp-empty*:

$\text{supp } f = \{\} \implies f \ x = 0$
by(*simp add:supp-def*)

lemma *nsupp-zero*:

$x \notin \text{supp } f \implies f \ x = 0$
by(*simp add:supp-def*)

lemma *sum-supp*:

fixes $f :: 'a :: \text{finite} \Rightarrow \text{real}$
shows $\text{sum } f \ (\text{supp } f) = \text{sum } f \ \text{UNIV}$

proof –

have $\text{sum } f \ (\text{UNIV} - \text{supp } f) = 0$
by(*simp add:supp-def*)

hence $\text{sum } f \ (\text{supp } f) = \text{sum } f \ (\text{UNIV} - \text{supp } f) + \text{sum } f \ (\text{supp } f)$
by(*simp*)

also have $\dots = \text{sum } f \ \text{UNIV}$
by(*simp add:sum.subset-diff[symmetric]*)

finally show *?thesis* .

qed

4.13.1 Truncated Subtraction

definition

$\text{tminus} :: \text{real} \Rightarrow \text{real} \Rightarrow \text{real}$ (**infixl** \ominus 60)

where

$x \ominus y = \max (x - y) \ 0$

lemma *minus-le-tminus*[*intro!,simp*]:

$a - b \leq a \ominus b$
unfolding *tminus-def* **by**(*auto*)

lemma *tminus-cancel-1*:

$0 \leq a \implies a + 1 \ominus 1 = a$
unfolding *tminus-def* **by**(*simp*)

lemma *tminus-zero-imp-le*:

$x \ominus y \leq 0 \implies x \leq y$
by(*simp add:tminus-def*)

lemma *tminus-zero*[*simp*]:

$0 \leq x \implies x \ominus 0 = x$
by(*simp add:tminus-def*)

lemma *tminus-left-mono*:
 $a \leq b \implies a \ominus c \leq b \ominus c$
unfolding *tminus-def*
by(*case-tac a ≤ c, simp-all*)

lemma *tminus-less*:
 $\llbracket 0 \leq a; 0 \leq b \rrbracket \implies a \ominus b \leq a$
unfolding *tminus-def* **by**(*force*)

lemma *tminus-left-distrib*:
assumes *nna*: $0 \leq a$
shows $a * (b \ominus c) = a * b \ominus a * c$
proof(*cases b ≤ c*)
case *True* **note** *le = this*
hence $a * \max (b - c) 0 = 0$ **by**(*simp add:max.absorb2*)
also {
from *nna le* **have** $a * b \leq a * c$ **by**(*blast intro:mult-left-mono*)
hence $0 = \max (a * b - a * c) 0$ **by**(*simp add:max.absorb1*)
}
finally show *?thesis* **by**(*simp add:tminus-def*)
next
case *False* **hence** *le*: $c \leq b$ **by**(*auto*)
hence $a * \max (b - c) 0 = a * (b - c)$ **by**(*simp only:max.absorb1*)
also {
from *nna le* **have** $a * c \leq a * b$ **by**(*blast intro:mult-left-mono*)
hence $a * (b - c) = \max (a * b - a * c) 0$ **by**(*simp add:max.absorb1 field-simps*)
}
finally show *?thesis* **by**(*simp add:tminus-def*)
qed

lemma *tminus-le[simp]*:
 $b \leq a \implies a \ominus b = a - b$
unfolding *tminus-def* **by**(*simp*)

lemma *tminus-le-alt[simp]*:
 $a \leq b \implies a \ominus b = 0$
by(*simp add:tminus-def*)

lemma *tminus-nle[simp]*:
 $\neg b \leq a \implies a \ominus b = 0$
unfolding *tminus-def* **by**(*simp*)

lemma *tminus-add-mono*:
 $(a+b) \ominus (c+d) \leq (a \ominus c) + (b \ominus d)$
proof(*cases 0 ≤ a - c*)
case *True* **note** *pac = this*
show *?thesis*
proof(*cases 0 ≤ b - d*)

```

case True note pb = this
from pac and pb have  $(c + d) \leq (a + b)$  by(simp)
with pac and pb show ?thesis by(simp)
next
case False with pac show ?thesis
by(cases  $c + d \leq a + b$ , auto)
qed
next
case False note nac = this
show ?thesis
proof(cases  $0 \leq b - d$ )
case True with nac show ?thesis
by(cases  $c + d \leq a + b$ , auto)
next
case False note nbd = this
with nac have  $\neg(c + d) \leq (a + b)$  by(simp)
with nac and nbd show ?thesis by(simp)
qed
qed

lemma tminus-sum-mono:
assumes fS: finite S
shows  $\text{sum } f S \ominus \text{sum } g S \leq \text{sum } (\lambda x. f x \ominus g x) S$ 
(is ?X S)
proof(rule finite-induct)
from fS show finite S .

show ?X {} by(simp)

fix x and F
assume fF: finite F and xniF:  $x \notin F$ 
and IH: ?X F
have  $f x + \text{sum } f F \ominus g x + \text{sum } g F \leq$ 
 $(f x \ominus g x) + (\text{sum } f F \ominus \text{sum } g F)$ 
by(rule tminus-add-mono)
also from IH have  $\dots \leq (f x \ominus g x) + (\sum_{x \in F} f x \ominus g x)$ 
by(rule add-left-mono)
finally show ?X (insert x F)
by(simp add:sum.insert[OF fF xniF])
qed

lemma tminus-nneg[simp,intro]:
 $0 \leq a \ominus b$ 
by(cases  $b \leq a$ , auto)

lemma tminus-right-antimono:
assumes clb:  $c \leq b$ 
shows  $a \ominus b \leq a \ominus c$ 
proof(cases  $b \leq a$ )

```

case *True*
moreover with *clb* **have** $c \leq a$ **by**(*auto*)
moreover note *clb*
ultimately show *?thesis* **by**(*simp*)
next
case *False* **then show** *?thesis* **by**(*simp*)
qed

lemma *min-tminus-distrib*:
 $\min a b \ominus c = \min (a \ominus c) (b \ominus c)$
unfolding *tminus-def* **by**(*auto*)

end

Bibliography

- David Cock. Verifying probabilistic correctness in Isabelle with pGCL. In *Proceedings of the 7th Systems Software Verification*, pages 1–10, Sydney, Australia, November 2012. doi: 10.4204/EPTCS.102.15.
- David Cock. Practical probability: Applying pGCL to lattice scheduling. In *Proceedings of the 4th International Conference on Interactive Theorem Proving*, pages 1–16, Rennes, France, July 2013. doi: 10.1007/978-3-642-39634-2_23.
- David Cock. From probabilistic operational semantics to information theory - side channels with pGCL in Isabelle. In *Proceedings of the 5th International Conference on Interactive Theorem Proving*, pages 1–15, Vienna, Austria, July 2014a. Springer.
- David Cock. *Leakage in Trustworthy Systems*. PhD thesis, University of New South Wales, 2014b.
- Edsger W. Dijkstra. Guarded commands, nondeterminacy and formal derivation of programs. *Communications of the ACM*, 18(8):453–457, August 1975. ISSN 0001-0782. doi: 10.1145/360933.360975.
- Joe Hurd, Annabelle McIver, and Carroll Morgan. Probabilistic guarded commands mechanized in HOL. *Theoretical Computer Science*, 346(1):96 – 112, 2005. ISSN 0304-3975. doi: 10.1016/j.tcs.2005.08.005. URL <http://www.sciencedirect.com/science/article/pii/S0304397505004767>.
- Annabelle McIver and Carroll Morgan. *Abstraction, Refinement and Proof for Probabilistic Systems*. Springer, 2004.
- Steve Selvin. A problem in probability (letter to the editor). *American Statistician*, 29(1):67, Feb 1975.