Hoare Logics for Time Bounds

Maximilian P. L. Haslbeck   Tobias Nipkow*

September 1, 2025

Abstract

We study three different Hoare logics for reasoning about time bounds of imperative programs and formalize them in Isabelle/HOL: a classical Hoare like logic due to Nielson, a logic with potentials due to Carbonneaux *et al.* and a *separation logic* following work by Atkey, Chaguérand and Pottier. These logics are formally shown to be sound and complete. Verification condition generators are developed and are shown sound and complete too. We also consider variants of the systems where we abstract from multiplicative constants in the running time bounds, thus supporting a big-O style of reasoning. Finally we compare the expressive power of the three systems.

An informal description is found in an accompanying report [HN18].

Contents

*Supported by DFG GRK 1480 (PUMA) and Koselleck Grant NI 491/16-1

1 Arithmetic and Boolean Expressions

theory *AExp* **imports** *Main* **begin**

1.1 Arithmetic Expressions

type_synonym *vname* = *string*
type_synonym *val* = *int*
type_synonym *state* = *vname* ⇒ *val*

datatype *aexp* = *N int* | *V vname* | *Plus aexp aexp* | *Times aexp aexp* | *Div aexp aexp*

fun *aval* :: *aexp* ⇒ *state* ⇒ *val* **where**
aval (*N n*) *s* = *n* |
aval (*V x*) *s* = *s x* |
aval (*Plus* a_1 a_2) *s* = *aval* a_1 *s* + *aval* a_2 *s*|
aval (*Times* a_1 a_2) *s* = *aval* a_1 *s* ∗ *aval* a_2 *s*|
aval (*Div* a_1 a_2) *s* = *aval* a_1 *s div aval* a_2 *s*

value *aval* (*Plus* (*V* ''*x*'') (*N 5*)) (λ*x*. *if x* = ''*x*'' *then 7 else 0*)

The same state more concisely:

value *aval* (*Plus* (*V* ''*x*'') (*N 5*)) ((λ*x*. *0*) (''*x*'':= *7*))

A little syntax magic to write larger states compactly:

definition *null_state* (‹<>›) **where**
null_state ≡ λ*x*. *0*
syntax
State :: *updbinds* => ′*a* (‹<>›)
translations
_State ms == *_Update* <> *ms*
_State (*_updbinds b bs*) <= *_Update* (*_State b*) *bs*

end
theory *BExp* **imports** *AExp* **begin**

1.2 Boolean Expressions

datatype *bexp* = *Bc bool* | *Not bexp* | *And bexp bexp* | *Less aexp aexp*

fun *bval* :: *bexp* ⇒ *state* ⇒ *bool* **where**
bval (*Bc v*) *s* = *v* |
bval (*Not b*) *s* = (¬ *bval b s*) |

$bval\ (And\ b_1\ b_2)\ s = (bval\ b_1\ s \wedge bval\ b_2\ s)\ |$
$bval\ (Less\ a_1\ a_2)\ s = (aval\ a_1\ s < aval\ a_2\ s)$

value $bval\ (Less\ (V\ ''x'')\ (Plus\ (N\ 3)\ (V\ ''y'')))$
$\quad <''x'' := 3,\ ''y'' := 1>$

end

2 IMP — A Simple Imperative Language

theory *Com* **imports** *BExp* **begin**

datatype
$com = SKIP$
$\quad |\ Assign\ vname\ aexp \qquad (\langle\_ ::= \_\rangle\ [1000,\ 61]\ 61)$
$\quad |\ Seq\ \ com\ \ com \qquad (\langle\_;;/\ \_\rangle\ \ [60,\ 61]\ 60)$
$\quad |\ If\ \ bexp\ com\ com \qquad (\langle(IF\ \_/\ THEN\ \_/\ ELSE\ \_)\rangle\ \ [0,\ 0,\ 61]\ 61)$
$\quad |\ While\ \ bexp\ com \qquad (\langle(WHILE\ \_/\ DO\ \_)\rangle\ \ [0,\ 61]\ 61)$

end

theory *Big_Step* **imports** *Com* **begin**

2.1 Big-Step Semantics of Commands

The big-step semantics is a straight-forward inductive definition with concrete syntax. Note that the first parameter is a tuple, so the syntax becomes $(c,s) \Rightarrow s'$.

inductive
$big\_step :: com \times state \Rightarrow state \Rightarrow bool$ (**infix** $\langle\Rightarrow\rangle$ 55)
where
$Skip$: $(SKIP,s) \Rightarrow s\ |$
$Assign$: $(x ::= a,s) \Rightarrow s(x := aval\ a\ s)\ |$
Seq: $[\![\ (c_1,s_1) \Rightarrow s_2;\ \ (c_2,s_2) \Rightarrow s_3\ ]\!] \Longrightarrow (c_1;;c_2,\ s_1) \Rightarrow s_3\ |$
$IfTrue$: $[\![\ bval\ b\ s;\ \ (c_1,s) \Rightarrow t\ ]\!] \Longrightarrow (IF\ b\ THEN\ c_1\ ELSE\ c_2,\ s) \Rightarrow t\ |$
$IfFalse$: $[\![\ \neg bval\ b\ s;\ \ (c_2,s) \Rightarrow t\ ]\!] \Longrightarrow (IF\ b\ THEN\ c_1\ ELSE\ c_2,\ s) \Rightarrow t\ |$
$WhileFalse$: $\neg bval\ b\ s \Longrightarrow (WHILE\ b\ DO\ c,s) \Rightarrow s\ |$
$WhileTrue$:
$[\![\ bval\ b\ s_1;\ \ (c,s_1) \Rightarrow s_2;\ \ (WHILE\ b\ DO\ c,\ s_2) \Rightarrow s_3\ ]\!]$
$\Longrightarrow (WHILE\ b\ DO\ c,\ s_1) \Rightarrow s_3$

We want to execute the big-step rules:

code_pred *big_step* **.**

For inductive definitions we need command `values` instead of `value`.

values $\{t.\ (SKIP,\ \lambda\_.\ 0) \Rightarrow t\}$

We need to translate the result state into a list to display it.

values $\{map\ t\ [''x'']\ |t.\ (SKIP,\ <''x'' := 42>) \Rightarrow t\}$

values $\{map\ t\ [''x'']\ |t.\ (''x'' ::= N\ 2,\ <''x'' := 42>) \Rightarrow t\}$

values $\{map\ t\ [''x'',''y'']\ |t.$
$(WHILE\ Less\ (V\ ''x'')\ (V\ ''y'')\ DO\ (''x'' ::= Plus\ (V\ ''x'')\ (N\ 5)),$
$<''x'' := 0,\ ''y'' := 13>) \Rightarrow t\}$

Proof automation:

The introduction rules are good for automatically construction small program executions. The recursive cases may require backtracking, so we declare the set as unsafe intro rules.

declare *big_step.intros* [*intro*]

The standard induction rule

$[\![x1 \Rightarrow x2;\ \bigwedge s.\ P\ (SKIP,\ s)\ s;\ \bigwedge x\ a\ s.\ P\ (x ::= a,\ s)\ (s(x := aval\ a\ s));$
$\bigwedge c_1\ s_1\ s_2\ c_2\ s_3.$
$[\![(c_1,\ s_1) \Rightarrow s_2;\ P\ (c_1,\ s_1)\ s_2;\ (c_2,\ s_2) \Rightarrow s_3;\ P\ (c_2,\ s_2)\ s_3]\!]$
$\Longrightarrow P\ (c_1;;\ c_2,\ s_1)\ s_3;$
$\bigwedge b\ s\ c_1\ t\ c_2.$
$[\![bval\ b\ s;\ (c_1,\ s) \Rightarrow t;\ P\ (c_1,\ s)\ t]\!] \Longrightarrow P\ (IF\ b\ THEN\ c_1\ ELSE\ c_2,\ s)\ t;$
$\bigwedge b\ s\ c_2\ t\ c_1.$
$[\![\neg\ bval\ b\ s;\ (c_2,\ s) \Rightarrow t;\ P\ (c_2,\ s)\ t]\!] \Longrightarrow P\ (IF\ b\ THEN\ c_1\ ELSE\ c_2,\ s)$
$t;$
$\bigwedge b\ s\ c.\ \neg\ bval\ b\ s \Longrightarrow P\ (WHILE\ b\ DO\ c,\ s)\ s;$
$\bigwedge b\ s_1\ c\ s_2\ s_3.$
$[\![bval\ b\ s_1;\ (c,\ s_1) \Rightarrow s_2;\ P\ (c,\ s_1)\ s_2;\ (WHILE\ b\ DO\ c,\ s_2) \Rightarrow s_3;$
$P\ (WHILE\ b\ DO\ c,\ s_2)\ s_3]\!]$
$\Longrightarrow P\ (WHILE\ b\ DO\ c,\ s_1)\ s_3]\!]$
$\Longrightarrow P\ x1\ x2$

thm *big_step.induct*

This induction schema is almost perfect for our purposes, but our trick for reusing the tuple syntax means that the induction schema has two parameters instead of the c, s, and s' that we are likely to encounter. Splitting the tuple parameter fixes this:

lemmas *big_step_induct* = *big_step.induct*[*split_format*(*complete*)]
thm *big_step_induct*

$[\![(x1a, x1b) \Rightarrow x2a; \bigwedge s.\ P\ SKIP\ s\ s; \bigwedge x\ a\ s.\ P\ (x ::= a)\ s\ (s(x := aval\ a\ s));$
$\bigwedge c_1\ s_1\ s_2\ c_2\ s_3.$
$\quad [\![(c_1, s_1) \Rightarrow s_2; P\ c_1\ s_1\ s_2; (c_2, s_2) \Rightarrow s_3; P\ c_2\ s_2\ s_3]\!]$
$\quad \Longrightarrow P\ (c_1;; c_2)\ s_1\ s_3;$
$\bigwedge b\ s\ c_1\ t\ c_2.$
$\quad [\![bval\ b\ s; (c_1, s) \Rightarrow t; P\ c_1\ s\ t]\!] \Longrightarrow P\ (IF\ b\ THEN\ c_1\ ELSE\ c_2)\ s\ t;$
$\bigwedge b\ s\ c_2\ t\ c_1.$
$\quad [\![\neg\ bval\ b\ s; (c_2, s) \Rightarrow t; P\ c_2\ s\ t]\!] \Longrightarrow P\ (IF\ b\ THEN\ c_1\ ELSE\ c_2)\ s\ t;$
$\bigwedge b\ s\ c.\ \neg\ bval\ b\ s \Longrightarrow P\ (WHILE\ b\ DO\ c)\ s\ s;$
$\bigwedge b\ s_1\ c\ s_2\ s_3.$
$\quad [\![bval\ b\ s_1; (c, s_1) \Rightarrow s_2; P\ c\ s_1\ s_2; (WHILE\ b\ DO\ c, s_2) \Rightarrow s_3;$
$\quad\ P\ (WHILE\ b\ DO\ c)\ s_2\ s_3]\!]$
$\quad \Longrightarrow P\ (WHILE\ b\ DO\ c)\ s_1\ s_3]\!]$
$\Longrightarrow P\ x1a\ x1b\ x2a$

2.2 Rule inversion

What can we deduce from $(SKIP, s) \Rightarrow t$? That $s = t$. This is how we can automatically prove it:

inductive_cases *SkipE*[*elim*!]: $(SKIP,s) \Rightarrow t$
thm *SkipE*

This is an *elimination rule*. The [elim] attribute tells auto, blast and friends (but not simp!) to use it automatically; [elim!] means that it is applied eagerly.

Similarly for the other commands:

inductive_cases *AssignE*[*elim*!]: $(x ::= a,s) \Rightarrow t$
thm *AssignE*
inductive_cases *SeqE*[*elim*!]: $(c1;;c2,s1) \Rightarrow s3$
thm *SeqE*
inductive_cases *IfE*[*elim*!]: $(IF\ b\ THEN\ c1\ ELSE\ c2,s) \Rightarrow t$
thm *IfE*

inductive_cases *WhileE*[*elim*]: $(WHILE\ b\ DO\ c,s) \Rightarrow t$
thm *WhileE*

Only [elim]: [elim!] would not terminate.

An automatic example:

lemma $(IF\ b\ THEN\ SKIP\ ELSE\ SKIP,\ s) \Rightarrow t \Longrightarrow t = s$
by *blast*

Rule inversion by hand via the “cases” method:

```
lemma assumes (IF b THEN SKIP ELSE SKIP, s) ⇒ t
shows t = s
proof-
  from assms show ?thesis
  proof cases — inverting assms
    case IfTrue thm IfTrue
    thus ?thesis by blast
  next
    case IfFalse thus ?thesis by blast
  qed
qed
```

```
lemma assign_simp:
  (x ::= a,s) ⇒ s' ⟷ (s' = s(x := aval a s))
  by auto
```

An example combining rule inversion and derivations

```
lemma Seq_assoc:
  (c1;; c2;; c3, s) ⇒ s' ⟷ (c1;; (c2;; c3), s) ⇒ s'
proof
  assume (c1;; c2;; c3, s) ⇒ s'
  then obtain s1 s2 where
    c1: (c1, s) ⇒ s1 and
    c2: (c2, s1) ⇒ s2 and
    c3: (c3, s2) ⇒ s' by auto
  from c2 c3
  have (c2;; c3, s1) ⇒ s' by (rule Seq)
  with c1
  show (c1;; (c2;; c3), s) ⇒ s' by (rule Seq)
next
  — The other direction is analogous
  assume (c1;; (c2;; c3), s) ⇒ s'
  thus (c1;; c2;; c3, s) ⇒ s' by auto
qed
```

2.3 Command Equivalence

We call two statements c and c' equivalent wrt. the big-step semantics when *c started in s terminates in s' iff c' started in the same s also terminates in the same s'*. Formally:

```
abbreviation
  equiv_c :: com ⇒ com ⇒ bool (infix ‹∼› 50) where
  c ∼ c' ≡ (∀ s t. (c,s) ⇒ t  =  (c',s) ⇒ t)
```

Warning: $\sim$ is the symbol written `\ < s i m >` (without spaces).

As an example, we show that loop unfolding is an equivalence transformation on programs:

lemma *unfold_while*:
($WHILE\ b\ DO\ c$) $\sim$ ($IF\ b\ THEN\ c$;; $WHILE\ b\ DO\ c\ ELSE\ SKIP$) (**is** $?w \sim ?iw$)
proof –
— to show the equivalence, we look at the derivation tree for
— each side and from that construct a derivation tree for the other side
{ fix s t **assume** $(?w,\ s) \Rightarrow t$
— as a first thing we note that, if b is *False* in state s,
— then both statements do nothing:
{ assume $\neg bval\ b\ s$
hence $t = s$ **using** ‹$(?w,s) \Rightarrow t$› **by** *blast*
hence $(?iw,\ s) \Rightarrow t$ **using** ‹$\neg bval\ b\ s$› **by** *blast*
}
moreover
— on the other hand, if b is *True* in state s,
— then only the *WhileTrue* rule can have been used to derive $(?w,\ s) \Rightarrow t$
{ assume $bval\ b\ s$
with ‹$(?w,\ s) \Rightarrow t$› **obtain** s' **where**
$(c,\ s) \Rightarrow s'$ **and** $(?w,\ s') \Rightarrow t$ **by** *auto*
— now we can build a derivation tree for the *IF*
— first, the body of the True-branch:
hence $(c;;\ ?w,\ s) \Rightarrow t$ **by** (*rule Seq*)
— then the whole *IF*
with ‹$bval\ b\ s$› **have** $(?iw,\ s) \Rightarrow t$ **by** (*rule IfTrue*)
}
ultimately
— both cases together give us what we want:
have $(?iw,\ s) \Rightarrow t$ **by** *blast*
}
moreover
— now the other direction:
{ fix s t **assume** $(?iw,\ s) \Rightarrow t$
— again, if b is *False* in state s, then the False-branch
— of the *IF* is executed, and both statements do nothing:
{ assume $\neg bval\ b\ s$
hence $s = t$ **using** ‹$(?iw,\ s) \Rightarrow t$› **by** *blast*
hence $(?w,\ s) \Rightarrow t$ **using** ‹$\neg bval\ b\ s$› **by** *blast*
}
moreover

— on the other hand, if b is *True* in state s,
— then this time only the *IfTrue* rule can have be used
{ assume *bval b s*
with ‹(*?iw*, s) ⇒ t› **have** (c;; *?w*, s) ⇒ t **by** *auto*
— and for this, only the Seq-rule is applicable:
then obtain s' **where**
(c, s) ⇒ s' **and** (*?w*, s') ⇒ t **by** *auto*
— with this information, we can build a derivation tree for the *WHILE*
with ‹*bval b s*›
have (*?w*, s) ⇒ t **by** (*rule WhileTrue*)
}
ultimately
— both cases together again give us what we want:
have (*?w*, s) ⇒ t **by** *blast*
}
ultimately
show *?thesis* **by** *blast*
qed

Luckily, such lengthy proofs are seldom necessary. Isabelle can prove many such facts automatically.

lemma *while_unfold*:
(*WHILE b DO c*) ∼ (*IF b THEN c*;; *WHILE b DO c ELSE SKIP*)
by *blast*

lemma *triv_if*:
(*IF b THEN c ELSE c*) ∼ c
by *blast*

lemma *commute_if*:
(*IF b1 THEN* (*IF b2 THEN c11 ELSE c12*) *ELSE c2*)
∼
(*IF b2 THEN* (*IF b1 THEN c11 ELSE c2*) *ELSE* (*IF b1 THEN c12 ELSE c2*))
by *blast*

lemma *sim_while_cong_aux*:
(*WHILE b DO c*,s) ⇒ t ⟹ $c \sim c'$ ⟹ (*WHILE b DO* c',s) ⇒ t
apply(*induction WHILE b DO c s t arbitrary*: *b c rule*: *big_step_induct*)
apply *blast*
apply *blast*
done

lemma *sim_while_cong*: $c \sim c'$ ⟹ *WHILE b DO c* ∼ *WHILE b DO* c'

by (*metis sim_while_cong_aux*)

Command equivalence is an equivalence relation, i.e. it is reflexive, symmetric, and transitive. Because we used an abbreviation above, Isabelle derives this automatically.

lemma *sim_refl*: $c \sim c$ **by** *simp*
lemma *sim_sym*: $(c \sim c') = (c' \sim c)$ **by** *auto*
lemma *sim_trans*: $c \sim c' \Longrightarrow c' \sim c'' \Longrightarrow c \sim c''$ **by** *auto*

2.4 Execution is deterministic

This proof is automatic.

theorem *big_step_determ*: $[\![\ (c,s) \Rightarrow t;\ (c,s) \Rightarrow u\ ]\!] \Longrightarrow u = t$
by (*induction arbitrary*: *u rule*: *big_step.induct*) *blast+*

This is the proof as you might present it in a lecture. The remaining cases are simple enough to be proved automatically:

theorem
$(c,s) \Rightarrow t \ \Longrightarrow\ (c,s) \Rightarrow t' \ \Longrightarrow\ t' = t$
proof (*induction arbitrary*: t' *rule*: *big_step.induct*)
— the only interesting case, *WhileTrue*:
fix $b\ c\ s\ s_1\ t\ t'$
— The assumptions of the rule:
assume *bval b s* **and** $(c,s) \Rightarrow s_1$ **and** $(WHILE\ b\ DO\ c,s_1) \Rightarrow t$
— Ind.Hyp; note the $\bigwedge$ because of arbitrary:
assume *IHc*: $\bigwedge t'.\ (c,s) \Rightarrow t' \Longrightarrow t' = s_1$
assume *IHw*: $\bigwedge t'.\ (WHILE\ b\ DO\ c,s_1) \Rightarrow t' \Longrightarrow t' = t$
— Premise of implication:
assume $(WHILE\ b\ DO\ c,s) \Rightarrow t'$
with ‹*bval b s*› **obtain** ${s_1}'$ **where**
c: $(c,s) \Rightarrow {s_1}'$ **and**
w: $(WHILE\ b\ DO\ c,{s_1}') \Rightarrow t'$
by *auto*
from *c IHc* **have** ${s_1}' = s_1$ **by** *blast*
with *w IHw* **show** $t' = t$ **by** *blast*
qed *blast+* — prove the rest automatically

end

3 Big Step Semantics with Time

theory *Big_StepT* **imports** *Big_Step* **begin**

3.1 Big-Step with Time Semantics of Commands

inductive
  $big\_step\_t :: com \times state \Rightarrow nat \Rightarrow state \Rightarrow bool$ $(\langle\_ \Rightarrow \_ \Downarrow \_\rangle\ 55)$
where
$Skip$: $(SKIP,s) \Rightarrow Suc\ 0 \Downarrow s$ |
$Assign$: $(x ::= a,s) \Rightarrow Suc\ 0 \Downarrow s(x := aval\ a\ s)$ |
Seq: $[\![\ (c1,s1) \Rightarrow x \Downarrow s2;\ \ (c2,s2) \Rightarrow y \Downarrow s3\ ;\ z=x+y\ ]\!] \Longrightarrow (c1;;c2,\ s1) \Rightarrow z \Downarrow s3$ |
$IfTrue$: $[\![\ bval\ b\ s;\ \ (c1,s) \Rightarrow x \Downarrow t;\ y=x+1\ ]\!] \Longrightarrow (IF\ b\ THEN\ c1\ ELSE\ c2,\ s) \Rightarrow y \Downarrow t$ |
$IfFalse$: $[\![\ \neg bval\ b\ s;\ \ (c2,s) \Rightarrow x \Downarrow t;\ y=x+1\ \ ]\!] \Longrightarrow (IF\ b\ THEN\ c1\ ELSE\ c2,\ s) \Rightarrow y \Downarrow t$ |
$WhileFalse$: $[\![\ \neg bval\ b\ s\ ]\!] \Longrightarrow (WHILE\ b\ DO\ c,s) \Rightarrow Suc\ 0 \Downarrow s$ |
$WhileTrue$: $[\![\ bval\ b\ s1;\ \ (c,s1) \Rightarrow x \Downarrow s2;\ \ (WHILE\ b\ DO\ c,\ s2) \Rightarrow y \Downarrow s3;\ 1+x+y=z\ \ ]\!]$
  $\Longrightarrow (WHILE\ b\ DO\ c,\ s1) \Rightarrow z \Downarrow s3$

We want to execute the big-step rules:

code_pred big_step_t **.**

For inductive definitions we need command `values` instead of `value`.

values $\{(t,\ x).\ (SKIP,\ \lambda\_.\ 0) \Rightarrow x \Downarrow t\}$

We need to translate the result state into a list to display it.

values $\{map\ t\ [''x'']\ |t\ x.\ (SKIP,\ <''x'' := 42>) \Rightarrow x \Downarrow t\}$

values $\{map\ t\ [''x'']\ |t\ x.\ (''x'' ::= N\ 2,\ <''x'' := 42>) \Rightarrow x \Downarrow t\}$

values $\{map\ t\ [''x'',''y'']\ |t\ x.$
  $(WHILE\ Less\ (V\ ''x'')\ (V\ ''y'')\ DO\ (''x'' ::= Plus\ (V\ ''x'')\ (N\ 5)),$
   $<''x'' := 0,\ ''y'' := 13>) \Rightarrow x \Downarrow t\}$

Proof automation:

declare $big\_step\_t.intros$ [$intro$]

lemmas $big\_step\_t\_induct = big\_step\_t.induct[split\_format(complete)]$

3.2 Rule inversion

What can we deduce from $(SKIP,\ s) \Rightarrow x \Downarrow t$? That $s = t$. This is how we can automatically prove it:

inductive_cases $Skip\_tE[elim!]$: $(SKIP,s) \Rightarrow x \Downarrow t$
thm $Skip\_tE$

This is an *elimination rule*. The [elim] attribute tells auto, blast and friends (but not simp!) to use it automatically; [elim!] means that it is applied eagerly.

Similarly for the other commands:

inductive_cases *Assign_tE*[*elim*!]: $(x ::= a,s) \Rightarrow p \Downarrow t$
thm *Assign_tE*
inductive_cases *Seq_tE*[*elim*!]: $(c1;;c2,s1) \Rightarrow p \Downarrow s3$
thm *Seq_tE*
inductive_cases *If_tE*[*elim*!]: $(IF\ b\ THEN\ c1\ ELSE\ c2,s) \Rightarrow x \Downarrow t$
thm *If_tE*

inductive_cases *While_tE*[*elim*]: $(WHILE\ b\ DO\ c,s) \Rightarrow x \Downarrow t$
thm *While_tE*

Only [elim]: [elim!] would not terminate.

An automatic example:

lemma $(IF\ b\ THEN\ SKIP\ ELSE\ SKIP,\ s) \Rightarrow x \Downarrow t \Longrightarrow t = s$
by *blast*

Rule inversion by hand via the "cases" method:

lemma assumes $(IF\ b\ THEN\ SKIP\ ELSE\ SKIP,\ s) \Rightarrow x \Downarrow t$
shows $t = s$
proof−
  from *assms* **show** *?thesis*
  proof *cases* — inverting assms
    case *IfTrue*
    thus *?thesis* **by** *blast*
  next
    case *IfFalse* **thus** *?thesis* **by** *blast*
  qed
qed

lemma *assign_t_simp*:
  $(x ::= a,s) \Rightarrow Suc\ 0 \Downarrow\ s' \longleftrightarrow (s' = s(x := aval\ a\ s))$
  by (*auto*)

An example combining rule inversion and derivations

lemma *Seq_t_assoc*:
  $((c1;;\ c2;;\ c3,\ s) \Rightarrow p \Downarrow\ s') \longleftrightarrow ((c1;;\ (c2;;\ c3),\ s) \Rightarrow p \Downarrow s')$
proof
  assume $(c1;;\ c2;;\ c3,\ s) \Rightarrow p \Downarrow s'$
  then obtain *s1 s2 p1 p2 p3* **where**

$c1$: $(c1, s) \Rightarrow p1 \Downarrow s1$ **and**
$c2$: $(c2, s1) \Rightarrow p2 \Downarrow s2$ **and**
$c3$: $(c3, s2) \Rightarrow p3 \Downarrow s'$ **and**
p: $p = p1 + (p2 + p3)$ **by** *auto*
from *c2 c3*
have $(c2;; c3, s1) \Rightarrow p2 + p3 \Downarrow s'$ **apply** (*rule Seq*) **by** *simp*
with *c1*
show $(c1;; (c2;; c3), s) \Rightarrow p \Downarrow s'$ **unfolding** p **apply** (*rule Seq*) **by** *simp*
next
— The other direction is analogous
assume $(c1;; (c2;; c3), s) \Rightarrow p \Downarrow s'$
then obtain *s1 s2 p1 p2 p3* **where**
$c1$: $(c1, s) \Rightarrow p1 \Downarrow s1$ **and**
$c2$: $(c2, s1) \Rightarrow p2 \Downarrow s2$ **and**
$c3$: $(c3, s2) \Rightarrow p3 \Downarrow s'$ **and**
p: $p = (p1 + p2) + p3$ **by** *auto*
from *c1 c2*
have $(c1;; c2, s) \Rightarrow p1 + p2 \Downarrow s2$ **apply** (*rule Seq*) **by** *simp*
from *this c3*
show $(c1;; c2;; c3, s) \Rightarrow p \Downarrow s'$ **unfolding** p **apply** (*rule Seq*) **by** *simp*
qed

3.3 Relation to Big-Step Semantics

lemma $(\exists p. ((c, s) \Rightarrow p \Downarrow s')) = ((c, s) \Rightarrow s')$
proof
assume $\exists p. (c, s) \Rightarrow p \Downarrow s'$
then obtain p **where** $(c, s) \Rightarrow p \Downarrow s'$
by *blast*
then show $((c, s) \Rightarrow s')$
apply(*induct c s p s' rule*: *big_step_t_induct*)
prefer *2* **apply**(*rule Big_Step.Assign*)
apply(*auto*) **done**
next
assume $((c, s) \Rightarrow s')$
then show $(\exists p. ((c, s) \Rightarrow p \Downarrow s'))$
apply(*induct c s s' rule*: *big_step_induct*)
by *blast+*
qed

3.4 Execution is deterministic

This proof is automatic.

theorem *big_step_t_determ*: $[\![(c,s) \Rightarrow p \Downarrow t; (c,s) \Rightarrow q \Downarrow u]\!] \Longrightarrow u = t$

apply (*induction arbitrary*: *u q rule*: *big_step_t.induct*)
apply *blast*+ **done**

theorem *big_step_t_determ2*: $[\![$ (*c*,*s*) $\Rightarrow$ *p* $\Downarrow$ *t*; (*c*,*s*) $\Rightarrow$ *q* $\Downarrow$ *u* $]\!]$ $\Longrightarrow$ (*u* = *t* $\wedge$ *p*=*q*)
apply (*induction arbitrary*: *u q rule*: *big_step_t_induct*)
apply(*elim Skip_tE*) **apply**(*simp*)
apply(*elim Assign_tE*) **apply**(*simp*)
apply *blast*
apply(*elim If_tE*) **apply**(*simp*) **apply** *blast*
apply(*elim If_tE*) **apply** *blast* **apply**(*simp*)
apply(*erule While_tE*) **apply**(*simp*) **apply** *blast*
proof (*goal_cases*)
case *1*
from *1*(*7*) **show** *?case* **apply**(*safe*)
apply(*erule While_tE*)
using *1*(*1*−*6*) **apply** *fast*
using *1*(*1*−*6*) **apply** (*simp*)
apply(*erule While_tE*)
using *1*(*1*−*6*) **apply** *fast*
using *1*(*1*−*6*) **by** (*simp*)
qed

lemma *bigstep_det*: (*c1*, *s*) $\Rightarrow$ *p1* $\Downarrow$ *t1* $\Longrightarrow$ (*c1*, *s*) $\Rightarrow$ *p* $\Downarrow$ *t* $\Longrightarrow$ *p1*=*p* $\wedge$ *t1*=*t*
using *big_step_t_determ2* **by** *simp*

lemma *bigstep_progress*: (*c*, *s*) $\Rightarrow$ *p* $\Downarrow$ *t* $\Longrightarrow$ *p* > *0*
apply(*induct rule*: *big_step_t.induct*, *auto*) **done**

abbreviation *terminates* (‹$\downarrow$›) **where** *terminates cs* $\equiv$ ($\exists$ *n a*. (*cs* $\Rightarrow$ *n* $\Downarrow$ *a*))
abbreviation *thestate* (‹$\downarrow_s$›) **where** *thestate cs* $\equiv$ (*THE a*. $\exists$ *n*. (*cs* $\Rightarrow$ *n* $\Downarrow$ *a*))
abbreviation *thetime* (‹$\downarrow_t$›) **where** *thetime cs* $\equiv$ (*THE n*. $\exists$ *a*. (*cs* $\Rightarrow$ *n* $\Downarrow$ *a*))

lemma *bigstepT_the_cost*: (*c*, *s*) $\Rightarrow$ *t* $\Downarrow$ *s'* $\Longrightarrow$ $\downarrow_t$(*c*, *s*) = *t*
using *bigstep_det* **by** *blast*

lemma *bigstepT_the_state*: $(c,\ s) \Rightarrow t \Downarrow s' \Longrightarrow \downarrow_s(c,\ s) = s'$
  using *bigstep_det* **by** *blast*

lemma *SKIPnot*: $(\neg\ (SKIP,\ s) \Rightarrow p \Downarrow t) = (s{\neq}t \lor p{\neq}Suc\ 0)$ **by** *blast*

lemma *SKIPp*: $\downarrow_t(SKIP,s) = Suc\ 0$
  apply(*rule the_equality*)
  apply *fast*
  apply *auto* **done**

lemma *SKIPt*: $\downarrow_s(SKIP,s) = s$
  apply(*rule the_equality*)
  apply *fast*
  apply *auto* **done**

lemma *ASSp*: $(THE\ p.\ Ex\ (big\_step\_t\ (x ::= e,\ s)\ p)) = Suc\ 0$
  apply(*rule the_equality*)
  apply *fast*
  apply *auto* **done**

lemma *ASSt*: $(THE\ t.\ \exists\, p.\ (x ::= e,\ s) \Rightarrow p \Downarrow t) = s(x := aval\ e\ s)$
  apply(*rule the_equality*)
  apply *fast*
  apply *auto* **done**

lemma *ASSnot*: $(\ \neg\ (x ::= e,\ s) \Rightarrow p \Downarrow t\ ) = (p{\neq}Suc\ 0 \lor t{\neq}s(x := aval\ e\ s))$
  apply *auto* **done**

lemma *If_THE_True*: $Suc\ (THE\ n.\ \exists\, a.\ (c1,\ s) \Rightarrow n \Downarrow a) = \ (THE\ n.\ \exists\, a.\ (IF\ b\ THEN\ c1\ ELSE\ c2,\ s) \Rightarrow n \Downarrow a)$
    if *T*: *bval b s* **and** *c1_t*: *terminates* $(c1,s)$ **for** *s l*
proof –
  from *c1_t* **obtain** *p t* **where** *a*: $(c1,\ s) \Rightarrow p \Downarrow t$ **by** *blast*
  with *T* **have** *b*: $(IF\ b\ THEN\ c1\ ELSE\ c2,\ s) \Rightarrow p{+}1 \Downarrow t$ **using** *IfTrue* **by** *simp*
  from *a bigstepT_the_cost* **have** $(THE\ n.\ \exists\, a.\ (c1,\ s) \Rightarrow n \Downarrow a) = p$ **by** *simp*
moreover

from *b bigstepT_the_cost* **have** (*THE n*. ∃ *a*. (*IF b THEN c1 ELSE c2*, *s*) ⇒ *n* ⇓ *a*) = *p+1* **by** *simp*
ultimately
show *?thesis* **by** *simp*
qed

lemma *If_THE_False*: *Suc* (*THE n*. ∃ *a*. (*c2*, *s*) ⇒ *n* ⇓ *a*) = (*THE n*. ∃ *a*. (*IF b THEN c1 ELSE c2*, *s*) ⇒ *n* ⇓ *a*)
if *T*: ¬*bval b s* **and** *c2_t*: ↓ (*c2*,*s*) **for** *s l*
proof −
from *c2_t* **obtain** *p t* **where** *a*: (*c2*, *s*) ⇒ *p* ⇓ *t* **by** *blast*
with *T* **have** *b*: (*IF b THEN c1 ELSE c2*, *s*) ⇒ *p+1* ⇓ *t* **using** *IfFalse* **by** *simp*
from *a bigstepT_the_cost* **have** (*THE n*. ∃ *a*. (*c2*, *s*) ⇒ *n* ⇓ *a*) = *p* **by** *simp*
moreover
from *b bigstepT_the_cost* **have** (*THE n*. ∃ *a*. (*IF b THEN c1 ELSE c2*, *s*) ⇒ *n* ⇓ *a*) = *p+1* **by** *simp*
ultimately
show *?thesis* **by** *simp*
qed

end
theory *Nielson_Hoare*
imports *Big_StepT*
begin

4 Nielson Style Hoare Logic with logical variables

abbreviation *eq a b* == (*And* (*Not* (*Less a b*)) (*Not* (*Less b a*)))

type_synonym *lvname* = *string*
type_synonym *assn2* = (*lvname* ⇒ *nat*) ⇒ *state* ⇒ *bool*
type_synonym *tbd* = *state* ⇒ *nat*

4.1 The support of an assn2

definition *support* :: *assn2* ⇒ *string set* **where**
support P = {*x*. ∃ *l1 l2 s*. (∀ *y*. *y* ≠ *x* ⟶ *l1 y* = *l2 y*) ∧ *P l1 s* ≠ *P l2 s*}

lemma *support_and*: *support* (λ*l s*. *P l s* ∧ *Q l s*) ⊆ *support P* ∪ *support Q*

unfolding *support_def* **by** *blast*

lemma *support_impl*: *support* $(\lambda l\ s.\ P\ s \longrightarrow Q\ l\ s) \subseteq$ *support Q*
unfolding *support_def* **by** *blast*

lemma *support_exist*: *support* $(\lambda l\ s.\ \exists\ z{::}nat.\ Q\ z\ l\ s) \subseteq (UN\ n.\ support\ (Q\ n))$
unfolding *support_def* **apply**(*auto*)
by *blast+*

lemma *support_all*: *support* $(\lambda l\ s.\ \forall z.\ Q\ z\ l\ s) \subseteq (UN\ n.\ support\ (Q\ n))$
unfolding *support_def* **apply**(*auto*)
by *blast+*

lemma *support_single*: *support* $(\lambda l\ s.\ P\ (l\ a)\ s) \subseteq \{a\}$
unfolding *support_def* **by** *fastforce*

lemma *support_inv*: $\bigwedge P.\ support\ (\lambda l\ s.\ P\ s) = \{\}$
unfolding *support_def* **by** *blast*

lemma *assn2_lupd*: $x \notin support\ Q \Longrightarrow Q\ (l(x{:=}n)) = Q\ l$
by(*simp add*: *support_def fun_upd_other fun_eq_iff*)
(*metis* (*no_types*, *lifting*) *fun_upd_def*)

4.2 Validity

abbreviation *state_subst* :: *state* ⇒ *aexp* ⇒ *vname* ⇒ *state*
(‹_[_'/_]› [*1000*,*0*,*0*] *999*)
where $s[a/x] == s(x := aval\ a\ s)$

definition *hoare1_valid* :: *assn2* ⇒ *com* ⇒ *tbd* ⇒ *assn2* ⇒ *bool*
(‹$\models_1$ {(*1*_)}/ (_)/ { _ ⇓(*1*_)}› *50*) **where**
$\models_1 \{P\}\ c\ \{q \Downarrow Q\} \longleftrightarrow (\exists\, k{>}0.\ (\forall\, l\ s.\ P\ l\ s \longrightarrow (\exists\, t\ p.\ ((c,s) \Rightarrow p \Downarrow t) \wedge p \leq k * (q\ s) \wedge Q\ l\ t)))$

4.3 Hoare rules

inductive
hoare1 :: *assn2* ⇒ *com* ⇒ *tbd* ⇒ *assn2* ⇒ *bool* (‹$\vdash_1$ ({(*1*_)}/ (_)/ { _ ⇓ (*1*_)})› *50*)
where

Skip: $\vdash_1 \{P\}\ SKIP\ \{\ (\%s.\ Suc\ 0) \Downarrow P\}$ |

Assign: $\vdash_1 \{\lambda l\ s.\ P\ l\ (s[a/x])\}\ x::=a\ \{\ \ (\%s.\ Suc\ 0) \Downarrow P\}$ |

If: $[\![\vdash_1 \{\lambda l\ s.\ P\ l\ s \wedge bval\ b\ s\}\ c_1\ \{\ e1 \Downarrow Q\};$
$\vdash_1 \{\lambda l\ s.\ P\ l\ s \wedge \neg\ bval\ b\ s\}\ c_2\ \{\ e1 \Downarrow Q\}\ ]\!]$
$\Longrightarrow \vdash_1 \{P\}\ IF\ b\ THEN\ c_1\ ELSE\ c_2\ \{\ (\lambda s.\ e1\ s + Suc\ 0\ ) \Downarrow Q\}$ |

Seq: $[\![\vdash_1 \{\ (\%l\ s.\ P_1\ l\ s \wedge l\ x = e2'\ s\ )\ \}\ c_1\ \{\ e1 \Downarrow (\%l\ s.\ P_2\ l\ s \wedge\ \ e2\ s \leq l\ x\ )\};$
$\vdash_1 \{P_2\}\ c_2\ \{\ e2 \Downarrow P_3\};\ x \notin support\ P_1;\ x \notin support\ P_2;$
$\bigwedge l\ s.\ P_1\ l\ s \Longrightarrow e1\ s + e2'\ s \leq e\ s]\!]$
$\Longrightarrow \vdash_1 \{P_1\}\ c_1;;c_2\ \{\ e \Downarrow P_3\}$ |

While:
$[\![\vdash_1 \{\lambda l\ s.\ P\ l\ s \wedge bval\ b\ s \wedge e'\ s = l\ y\}\ c\ \{\ \ e'' \Downarrow \lambda l\ s.\ P\ l\ s \wedge e\ s \leq l\ y\};$
$\forall l\ s.\ bval\ b\ s \wedge P\ l\ s \longrightarrow\ \ e\ s \geq 1\ +\ \ e'\ s + e''\ s\ \ ;$
$\forall l\ s.\ {}^{\sim}\ bval\ b\ s \wedge P\ l\ s \longrightarrow 1 \leq e\ s;$
$y \notin support\ P\ \ ]\!]$
$\Longrightarrow \vdash_1 \{P\}\ WHILE\ b\ DO\ c\ \{\ e \Downarrow \lambda l\ s.\ P\ l\ s \wedge \neg\ bval\ b\ s\}$ |

conseq: $[\![\exists k{>}0.\ \forall l\ s.\ P'\ l\ s \longrightarrow (\ e\ s \leq k * (e'\ s) \wedge(\forall t.\ \exists l'.\ P\ l'\ s \wedge (\ Q\ l'\ t \longrightarrow Q'\ l\ t)\ ));$
$\vdash_1 \{P\}c\{\ e \Downarrow Q\}\ \ \ ]\!] \Longrightarrow$
$\vdash_1 \{P'\}c\{e' \Downarrow Q'\}$

Derived Rules:

lemma *conseq_old*: $[\![\exists k{>}0.\ \forall l\ s.\ P'\ l\ s \longrightarrow (P\ l\ s \wedge (\ e'\ s \leq\ \ k * (e\ s)));$
$\vdash_1 \{P\}c\{\ e' \Downarrow\ \ Q\};\ \forall l\ s.\ Q\ l\ s \longrightarrow Q'\ l\ s\ \ ]\!] \Longrightarrow$
$\vdash_1 \{P'\}c\{e \Downarrow Q'\}$
using *conseq* **apply**(*metis*) **done**

lemma *If2*: $[\![\vdash_1 \{\lambda l\ s.\ P\ l\ s \wedge bval\ b\ s\}\ c_1\ \{\ e \Downarrow Q\};\ \vdash_1 \{\lambda l\ s.\ P\ l\ s \wedge \neg\ bval\ b\ s\}\ c_2\ \{\ e \Downarrow Q\};$
$\bigwedge l\ s.\ P\ l\ s \Longrightarrow e\ s + 1 = e'\ s\ ]\!]$
$\Longrightarrow \vdash_1 \{P\}\ IF\ b\ THEN\ c_1\ ELSE\ c_2\ \{\ e' \Downarrow Q\}$
apply(*rule conseq*[*OF _ If*, **where** $P{=}P$ **and** $P'{=}P$]) **by**(*auto*)

lemma *strengthen_pre*:
$[\![\forall l\ s.\ P'\ l\ s \longrightarrow P\ l\ s\ ;\ \ \vdash_1 \{P\}\ c\ \{\ e \Downarrow Q\}\ ]\!] \Longrightarrow \vdash_1 \{P'\}\ c\ \{\ e \Downarrow Q\}$
apply(*rule conseq_old*[**where** $e'{=}e$ **and** $Q{=}Q$ **and** $P{=}P$])
by(*auto*)

lemma *weaken_post*:

$[\![\vdash_1 \{P\}\ c\ \{e \Downarrow Q\};\ \ \forall\, l\ s.\ Q\ l\ s \longrightarrow Q'\ l\ s\ ]\!] \Longrightarrow\ \vdash_1 \{P\}\ c\ \{e \Downarrow Q'\}$
apply(*rule conseq_old*[**where** $e'=e$ **and** $Q=Q$ **and** $P=P$])
by(*auto*)

lemma *ub_cost*:
$[\![(\exists\, k{>}0.\ \forall\, l\ s.\ P\ l\ s \longrightarrow e'\ s \leq k * (e\ s));\ \vdash_1 \{P\}\ c\ \{e' \Downarrow Q\}\ ]\!] \Longrightarrow\ \vdash_1 \{P\}\ c\ \{e \Downarrow Q\}$
apply(*rule conseq_old*[**where** $e'=e'$ **and** $Q=Q$ **and** $P=P$])
by(*auto*)

lemma *Assign'*: $\forall\, l\ s.\ P\ l\ s \longrightarrow Q\ l\ (s[a/x]) \Longrightarrow (\vdash_1 \{P\}\ x ::= a\ \{\ (\%s.\ 1) \Downarrow Q\})$
using *strengthen_pre*[*OF _ Assign*]
by (*simp*)

4.4 Soundness

The soundness theorem:

theorem *hoare1_sound*: $\vdash_1 \{P\}c\{e \Downarrow Q\}\ \Longrightarrow\ \models_1 \{P\}c\{e \Downarrow Q\}$
apply(*unfold hoare1_valid_def*)
proof(*induction rule*: *hoare1.induct*)
case (*Skip P*)
show *?case* **by** *fastforce*
next
case (*Assign P a x*)
show *?case* **by** *fastforce*
next
case (*Seq P1 x e2' c1 e1 P2 e2 c2 P3 e*)
from *Seq*(*6*) **obtain** *k* **where** *k*: $k{>}0$ **and** *S6*: $\forall\, l\ s.\ P1\ l\ s \wedge l\ x = e2'\ s \longrightarrow (\exists\, t\ p.\ (c1,\ s) \Rightarrow p \Downarrow t \wedge p \leq k * e1\ s \wedge P2\ l\ t \wedge e2\ t \leq l\ x)$ **by** *auto*
from *Seq*(*7*) **obtain** *k'* **where** *k'*: $k'{>}0$ **and** *S7*: $\forall\, l\ s.\ P2\ l\ s \longrightarrow (\exists\, t\ p.\ (c2,\ s) \Rightarrow p \Downarrow t \wedge p \leq k' * e2\ s \wedge P3\ l\ t)$ **by** *auto*
from *k k'* **have** $0 < max\ k\ k'$ **by** *auto*
show *?case*
proof (*rule exI*[**where** $x{=}max\ k\ k'$], *safe*)
fix *l s*
have *x_supp*: $x \notin support\ P1$ **by** *fact*
have *x_supp2*: $x \notin support\ P2$ **by** *fact*

from *S6* **have** *S*: $P1\ (l(x := e2'\ s))\ s \wedge (l(x := e2'\ s))\ x = e2'\ s \longrightarrow (\exists\, t\ p.\ (c1,\ s) \Rightarrow p \Downarrow t \wedge\ p \leq k * e1\ s \wedge P2\ (l(x := e2'\ s))\ t \wedge e2\ t \leq (l(x := e2'\ s))\ x)$
by *blast*

assume *a*: *P1 l s*

with *Seq*(*5*) **have** *1*: *e1 s* + *e2′ s* ≤ *e s* **by** *simp*
with *a S assn2_lupd*[*OF x_supp*] **have** (∃ *t p*. (*c1*, *s*) ⇒ *p* ⇓ *t* ∧ *p* ≤ *k* ∗ *e1 s* ∧ *P2* (*l*(*x* := *e2′ s*)) *t* ∧ *e2 t* ≤ (*l*(*x* := *e2′ s*)) *x*) **by** *simp*
then obtain *t p* **where** *c1*: (*c1*, *s*) ⇒ *p* ⇓ *t* **and** *cost1*: *p* ≤ *k* ∗ *e1 s* **and** *P2′*: *P2* (*l*(*x* := *e2′ s*)) *t* **and** *31*: *e2 t* ≤ (*l*(*x* := *e2′ s*)) *x* **by** *blast*
from *P2′ assn2_lupd*[*OF x_supp2*] **have** *P2*: *P2 l t* **by** *auto*
from *31* **have** *3*: *e2 t* ≤ *e2′ s* **by** *simp*
from *S7 P2* **have** (∃ *t′ p′*. ((*c2*, *t*) ⇒ *p′* ⇓ *t′*) ∧ *p′* ≤ *k′* ∗ *e2 t* ∧ *P3 l t′*) **by** *blast*
then obtain *t′ p′* **where** *c2*: (*c2*, *t*) ⇒ *p′* ⇓ *t′* **and** *cost2*: *p′* ≤ *k′* ∗ (*e2 t*) **and** *P3*: *P3 l t′* **by** *blast*

from *c1 c2* **have** *weg*: (*c1*;; *c2*, *s*) ⇒ *p* + *p′* ⇓ *t′*
apply (*rule Big_StepT.Seq*) **by** *simp*
from *cost1 cost2 3* **have** (*p*+*p′*) ≤ *k* ∗ (*e1 s*) + *k′* ∗ (*e2′ s*)
by (*meson add_mono mult_le_mono2 order_subst1*)
also have ... ≤ (*max k k′*) ∗ (*e1 s*) + (*max k k′*) ∗ (*e2′ s*)
by (*simp add*: *add_mono*)
also have ... ≤ (*max k k′*) ∗ (*e1 s* + *e2′ s*)
by (*simp add*: *add_mult_distrib2*)
also have ... ≤ (*max k k′*) ∗ (*e s*) **using** *1* **by** *simp*
finally
have *cost*: (*p* + *p′*) ≤ (*max k k′*) ∗ (*e s*) **.**

from *weg cost P3*
have (*c1*;; *c2*, *s*) ⇒ *p*+*p′* ⇓ *t′* ∧ (*p*+*p′*) ≤ (*max k k′*) ∗ (*e s*) ∧ *P3 l t′* **by** *blast*
then show (∃ *t p*. (*c1*;; *c2*, *s*) ⇒ *p* ⇓ *t* ∧ *p* ≤ (*max k k′*) ∗ (*e s*) ∧ *P3 l t*) **by** *metis*
qed *fact*
next
case (*If P b c1 e Q c2*)
from *If*(*3*) **obtain** *k1* **where** *k1*: *k1*>*0* **and** *If1*: ∀ *l s*. *P l s* ∧ *bval b s* ⟶ (∃ *t p*. (*c1*, *s*) ⇒ *p* ⇓ *t* ∧ *p* ≤ *k1* ∗ *e s* ∧ *Q l t*) **by** *auto*
from *If*(*4*) **obtain** *k2* **where** *k2*: *k2*>*0* **and** *If2*: ∀ *l s*. *P l s* ∧ ¬ *bval b s* ⟶ (∃ *t p*. (*c2*, *s*) ⇒ *p* ⇓ *t* ∧ *p* ≤ *k2* ∗ *e s* ∧ *Q l t*) **by** *auto*
let *?k′* = *max* (*k1*+*1*) (*k2*+*1*)
have *?k′*>*0* **by** *auto*
show *?case*
proof (*rule exI*[**where** *x*=*?k′*], *safe*)
fix *l s*
assume *P1*: *P l s*

show $\exists\, t\ p.\ (IF\ b\ THEN\ c1\ ELSE\ c2,\ s) \Rightarrow p \Downarrow t \wedge\ \ p \leq ?k' * (e\ s + Suc\ 0) \wedge Q\ l\ t$
proof (*cases bval b s*)
case *True*
with *If1 P1* **obtain** *t p* **where** $(c1,\ s) \Rightarrow p \Downarrow t\ p \leq k1 * (e\ s)\ Q\ l\ t$ **by** *blast*
with *True* **have** *1*: $(IF\ b\ THEN\ c1\ ELSE\ c2,\ s) \Rightarrow p{+}1 \Downarrow t\ (p + 1) \leq (k1{+}1) * (e\ s\ + Suc\ 0)$
$Q\ l\ t$
by *auto*
have $(k1{+}1) * (e\ s\ + Suc\ 0) \leq ?k' * (e\ s\ + Suc\ 0)$
by (*simp add*: *nat_mult_max_left*)
with *1* **have** *2*: $p{+}1\ \ \leq ?k' * (e\ s\ + Suc\ 0)$
by *linarith*
from *1 2* **show** $\exists\, t\ p.\ (IF\ b\ THEN\ c1\ ELSE\ c2,\ s) \Rightarrow p \Downarrow t \wedge p \leq ?k' * (e\ s + Suc\ 0) \wedge Q\ l\ t$ **by** *metis*
next
case *False*
with *If2 P1* **obtain** *t p* **where** $(c2,\ s) \Rightarrow p \Downarrow t\ p \leq k2 * (e\ s)\ Q\ l\ t$ **by** *blast*
with *False* **have** *1*: $(IF\ b\ THEN\ c1\ ELSE\ c2,\ s) \Rightarrow p{+}1 \Downarrow t\ (p + 1) \leq (k2{+}1) * (\ e\ s\ + Suc\ 0)$
$Q\ l\ t$
by *auto*
have $(k2{+}1) * (e\ s\ + Suc\ 0) \leq ?k' * (e\ s\ + Suc\ 0)$
by (*simp add*: *nat_mult_max_left*)
with *1* **have** *2*: $p{+}1\ \ \leq ?k' * (e\ s\ + Suc\ 0)$
by *linarith*
from *1 2* **show** $\exists\, t\ p.\ (IF\ b\ THEN\ c1\ ELSE\ c2,\ s) \Rightarrow p \Downarrow t \wedge\ \ p \leq ?k' * (e\ s + Suc\ 0) \wedge Q\ l\ t$ **by** *metis*
qed
qed *fact*
next
case (*conseq* $P'\ e\ e'\ P\ Q\ Q'\ c$)
from *conseq*(*1*) **obtain** *k1* **where** *k1*: $k1{>}0$ **and** *c1*: $\forall\, l\ s.\ P'\ l\ s \longrightarrow e\ s \leq k1 * e'\ s \wedge (\forall\, t.\ \exists\, l'.\ P\ l'\ s \wedge (Q\ l'\ t \longrightarrow Q'\ l\ t))$ **by** *auto*
then have *c1'*: $\bigwedge l\ s.\ P'\ l\ s \Longrightarrow e\ s \leq k1 * e'\ s \wedge (\forall\, t.\ \exists\, l'.\ P\ l'\ s \wedge (Q\ l'\ t \longrightarrow Q'\ l\ t))$
by *auto*
from *conseq*(*3*) **obtain** *k2* **where** *k2*: $k2{>}0$ **and** *c2*: $\forall\, l\ s.\ P\ l\ s \longrightarrow (\exists\, t\ p.\ (c,\ s) \Rightarrow p \Downarrow t \wedge p \leq k2 * e\ s \wedge Q\ l\ t)$ **by** *auto*
then have *c2'*: $\bigwedge l\ s.\ \ P\ l\ s \Longrightarrow (\exists\, t\ p.\ (c,\ s) \Rightarrow p \Downarrow t \wedge p \leq k2 * e\ s \wedge Q\ l\ t)$ **by** *auto*
have $k2{*}k1\ >\ 0$ **using** *k1 k2* **by** *auto*

show *?case*
proof (*rule exI*[**where** *x*=*k2*∗*k1*], *safe*)
fix *l s*
assume P' *l s*
with *c1′* **have** *A*: $e\ s \leq k1 * e'\ s$ **and** $\bigwedge t.\ \exists l'.\ P\ l'\ s \wedge (Q\ l'\ t \longrightarrow Q'\ l\ t)$ **by** *auto*
then obtain *fl* **where** $\bigwedge t.\ P\ (fl\ t)\ s$ **and** *B*: $\bigwedge t.\ Q\ (fl\ t)\ t \longrightarrow Q'\ l\ t$ **by** *metis*
with *c2′* **obtain** *ft fp* **where** *i*: $\bigwedge t.\ (c,\ s) \Rightarrow (fp\ t) \Downarrow (ft\ t)$ **and** *ii*: $\bigwedge t.\ (fp\ t) \leq k2 * e\ s$
and *iii*: $\bigwedge t.\ Q\ (fl\ t)\ (ft\ t)$
by *meson*
from *i* **obtain** *t p* **where** *tt*: $\bigwedge x.\ ft\ x = t\ \bigwedge x.\ fp\ x = p$ **using** *big_step_t_determ2*
by *meson*
with *i* **have** *c*: $(c,\ s) \Rightarrow p \Downarrow t$ **by** *simp*
from *tt ii iii* **have** *p*: $p \leq k2 * e\ s$ **and** *Q*: $\bigwedge x.\ Q\ (fl\ x)\ t$ **by** *auto*
have *p*: $p \leq k2 * k1 * e'\ s$ **using** *p A*
by (*metis le_trans mult.assoc mult_le_mono2*)
from *B Q* **have** *q*: $Q'\ l\ t$ **by** *fast*

from *c p q*
show $\exists t\ p.\ (c,\ s) \Rightarrow p \Downarrow t \wedge p \leq k2 * k1 * e'\ s \wedge Q'\ l\ t$
by *blast*
qed *fact*
next
case (*While INV b* e' *y c* e'' *e*)
from *While*(*5*) **obtain** *k* **where** *W6*: $\forall l\ s.\ INV\ l\ s \wedge bval\ b\ s \wedge e'\ s = l\ y \longrightarrow (\exists t\ p.\ (c,\ s) \Rightarrow p \Downarrow t \wedge p \leq k * e''\ s \wedge INV\ l\ t \wedge e\ t \leq l\ y)$ **by** *auto*
let $?k' = k+1$
{
fix *n l s*
have $[\![\ e\ s = n;\ INV\ l\ s\ ]\!] \Longrightarrow \exists t\ p.\ (WHILE\ b\ DO\ c,\ s) \Rightarrow p \Downarrow t \wedge p \leq ?k' * e\ s \wedge INV\ l\ t \wedge \neg\ bval\ b\ t$
proof(*induction n arbitrary*: *l s rule*: *less_induct*)
case (*less x*)

show *?case*
proof (*cases bval b s*)
case *False*
with *less*(*2*,*3*) *While*(*3*) **have** *b*: $1 \leq e\ s$ **by** *auto*

show *?thesis*
apply(*rule exI*[**where** *x*=*s*])

apply(*rule exI*[**where** *x=1*]) **apply** *safe*
subgoal using *WhileFalse*[*OF False*] **by** *simp*
subgoal using *b* **by** *auto*
subgoal using *less* **by** *auto*
subgoal using *False* **by** *auto*
done

next
case *True*
with *less*(*2*,*3*) *While*(*2*) **have** *bT*: *bval b s* **and** *cost1*: $1 + e'\ s + e''\ s \leq e\ s$ **by** *auto*
let $?l' = l(y := e'\ s)$

have *y_supp*: $y \notin$ *support INV* **by** *fact*

from *cost1* **have** *Z*: $e'\ s < x$ **using** *less*(*2*) **by** *auto*
from *W6*
have *INV* $?l'\ s \wedge$ *bval b s* $\wedge\ e'\ s = ?l'\ y$
$\longrightarrow (\exists\, t\ p.\ (c,\ s) \Rightarrow p \Downarrow t \wedge\ \ p \leq k * e''\ s\ \wedge$ *INV* $?l'\ t \wedge e\ t \leq ?l'\ y)$
by *blast*
with *less*(*3*) *bT*
have $(\exists\, t\ p.\ ((c,\ s) \Rightarrow p \Downarrow t) \wedge\ \ p \leq k * e''\ s\ \wedge$ *INV* $?l'\ t \wedge e\ t \leq e'\ s)$
using *assn2_lupd*[*OF y_supp*]
by(*auto*)
then obtain *t p* **where** *ceff*: $(c,\ s) \Rightarrow p \Downarrow t$ **and** *cost2*: $p \leq\ k * e''\ s$
and *INVz'*: *INV* $?l'\ t$ **and** *cost3*: $e\ t \leq\ e'\ s$
by *blast*

from *INVz'* **have** *INVz*: *INV l t* **using** *assn2_lupd*[*OF y_supp*] **by** *auto*
have $e\ t < x$ **using** *Z cost3* **by** *auto*
with *less*(*1*)[*OF _ _ INVz*, *of e t*] **obtain** $t'\ p'$
where *weff*: (*WHILE b DO c*, *t*) $\Rightarrow p' \Downarrow t'$ **and** *cost4*: $p' \leq ?k' * e\ t$ **and** *INV0*: *INV l* t'
and *nb*: $\neg$ *bval b* t'
by *fastforce*

have (*WHILE b DO c*, *s*) $\Rightarrow 1 + p + p'\ \Downarrow t'$
apply(*rule WhileTrue*)
apply *fact*
apply (*fact ceff*)

apply (*fact weff*) **by** *simp*
moreover
note *INV0 nb*
moreover
{
have $(1 + p + p') \leq 1 + k * e'' s + ?k' * e\ t$ **using** *cost2 cost4* **by** *linarith*
also have $\ldots \leq\ 1 + k * e'' s + ?k' * e'\ s$ **using** *cost3*
using *add_left_mono mult_le_mono2* **by** *blast*
also have $\ldots \leq ?k'*1 + ?k'* e'' s + ?k' * e'\ s$ **by** *force*
also have $\ldots = ?k' * (1+ e'\ s + e''\ s)$ **by** *algebra*
also have $\ldots \leq ?k' * e\ s$ **using** *cost1*
using *mult_le_mono2* **by** *blast*
finally have $(1 + p + p') \leq ?k' * e\ s$ **.**
}
ultimately
show *?thesis* **by** *metis*
qed
qed
}
then have *erg*: $\bigwedge l\ s.\ INV\ l\ s \Longrightarrow \exists\, t\ p.\ (WHILE\ b\ DO\ c,\ s) \Rightarrow p \Downarrow t \wedge p \leq (k + 1) * e\ s \wedge INV\ l\ t \wedge \neg\ bval\ b\ t$ **by** *auto*
show *?case* **apply**(*rule exI*[**where** *x=?k'*]) **using** *erg* **by** *fastforce*
qed

4.5 Completeness

definition *wp1* :: *com* ⇒ *assn2* ⇒ *assn2* (‹wp_1›) **where**
$wp_1\ c\ Q\ =\ (\lambda l\ s.\ \exists\, t\ p.\ (c,s) \Rightarrow p \Downarrow t \wedge Q\ l\ t)$

lemma *support_wpt*: $support\ (wp_1\ c\ Q) \subseteq support\ Q$
by(*simp add*: *support_def wp1_def*) *blast*

lemma *wp1_terminates*: $wp_1\ c\ Q\ l\ s \Longrightarrow \downarrow (c,\ s)$ **unfolding** *wp1_def* **by** *auto*

lemma *wp1_SKIP*[*simp*]: $wp_1\ SKIP\ Q = Q$ **by**(*auto intro*!: *ext simp*: *wp1_def*)

lemma *wp1_Assign*[*simp*]: $wp_1\ (x ::= e)\ Q = (\lambda l\ s.\ Q\ l\ (s(x := aval\ e\ s)))$
by(*auto intro*!: *ext simp*: *wp1_def*)

lemma *wp1_Seq*[*simp*]: wp_1 $(c_1;;c_2)$ Q = wp_1 c_1 $(wp_1$ c_2 $Q)$ **by** (*auto simp*: *wp1_def fun_eq_iff*)

lemma *wp1_If*[*simp*]: wp_1 (*IF b THEN* c_1 *ELSE* c_2) Q = (λl s. wp_1 (*if bval b s then* c_1 *else* c_2) Q l s) **by** (*auto simp*: *wp1_def fun_eq_iff*)

definition *prec c E* == %*s*. *E* (*THE t*. ($\exists$ *p*. (*c*,*s*) $\Rightarrow$ *p* $\Downarrow$ *t*))

lemma *wp1_prec_Seq_correct*: wp_1 (*c1*;;*c2*) *Q l s* $\Longrightarrow$ $\downarrow_t$ (*c1*, *s*) + *prec c1* (λs. $\downarrow_t$ (*c2*, *s*)) *s* $\leq$ $\downarrow_t$ (*c1*;; *c2*, *s*)
proof −
assume wp_1 (*c1*;;*c2*) *Q l s*
then have *wp*: wp_1 *c1* (wp_1 *c2 Q*) *l s* **by** *simp*
then obtain *t p* **where** *c1_term*: (*c1*, *s*) $\Rightarrow$ *p* $\Downarrow$ *t* **and** ($\exists$ *ta p*. (*c2*, *t*) $\Rightarrow$ *p* $\Downarrow$ *ta* $\wedge$ *Q l ta*) **unfolding** *wp1_def* **by** *blast*
then obtain *t′ p′* **where** *c2_term*: (*c2*, *t*) $\Rightarrow$ *p′* $\Downarrow$ *t′* **and** *Q l t′* **by** *blast*

have *p*: $\downarrow_t$ (*c1*, *s*) = *p* **using** *c1_term bigstepT_the_cost* **by** *simp*
have $\downarrow_t$ (*c2*, *t*) = *p′* **using** *c2_term bigstepT_the_cost* **by** *simp*

have *f*: (*THE t*. $\exists$ *p*. (*c1*, *s*) $\Rightarrow$ *p* $\Downarrow$ *t*) = *t* **using** *c1_term bigstepT_the_state* **by** *simp*

have *prec c1* (λs. $\downarrow_t$ (*c2*, *s*)) *s* = *p′* **unfolding** *prec_def f* **using** *c2_term bigstep_det* **by** *blast*
then have *p′*: *prec c1* (λs. (*THE n*. $\exists$ *a*. (*c2*, *s*) $\Rightarrow$ *n* $\Downarrow$ *a*)) *s*
= *p′* **unfolding** *prec_def* **by** *blast*

from *wp* **have** wp_1 (*c1*;;*c2*) *Q l s* **by** *simp*
then obtain *T P* **where** *c12_term*: (*c1*;;*c2*, *s*) $\Rightarrow$ *P* $\Downarrow$ *T* **and** *Q l T* **unfolding** *wp1_def* **by** *blast*

have *P*: (*THE n*. ($\exists$ *a*. (*c1*;;*c2*, *s*) $\Rightarrow$ *n* $\Downarrow$ *a*)) = *P* **using** *c12_term bigstepT_the_cost* **by** *simp*

from *c12_term* **have** *Ppp′*: *P* = *p* + *p′*
apply(*elim Seq_tE*)
using *c1_term bigstep_det c2_term* **by** *blast*

have (*THE n*. $\exists$ *a*. (*c1*, *s*) $\Rightarrow$ *n* $\Downarrow$ *a*) + *prec c1* (λs. (*THE n*. $\exists$ *a*. (*c2*, *s*) $\Rightarrow$ *n* $\Downarrow$ *a*)) *s*
= *p* + *p′* **using** *p p′* **by** *auto*
also have ... = *P* **using** *Ppp′* **by** *auto*

also have ... = $(THE\ n.\ (\exists\, a.\ (c1;;c2,\ s) \Rightarrow n \Downarrow a))$ **using** *P* **by** *auto*
finally
show $\downarrow_t (c1,\ s) + prec\ c1\ (\lambda s.\ \downarrow_t (c2,\ s))\ s \leq \downarrow_t (c1;;\ c2,\ s)$
by *simp*
qed

abbreviation $new\ Q \equiv SOME\ x.\ x \notin support\ Q$

lemma *bigstep_det*: $(c1,\ s) \Rightarrow p1 \Downarrow t1 \Longrightarrow (c1,\ s) \Rightarrow p \Downarrow t \Longrightarrow p1{=}p \wedge t1{=}t$
using *big_step_t_determ2* **by** *simp*

lemma *bigstepT_the_cost*: $(c,\ s) \Rightarrow P \Downarrow T \Longrightarrow (THE\ n.\ \exists\, a.\ (c,\ s) \Rightarrow n \Downarrow a) = P$
using *bigstep_det* **by** *blast*

lemma *bigstepT_the_state*: $(c,\ s) \Rightarrow P \Downarrow T \Longrightarrow (THE\ a.\ \exists\, n.\ (c,\ s) \Rightarrow n \Downarrow a) = T$
using *bigstep_det* **by** *blast*

lemma assumes *b*: *bval b s*
shows *wp1WhileTrue'*: $wp_1\ (WHILE\ b\ DO\ c)\ Q\ l\ s = wp_1\ c\ (wp_1\ (WHILE\ b\ DO\ c)\ Q)\ l\ s$
proof
assume $wp_1\ c\ (wp_1\ (WHILE\ b\ DO\ c)\ Q)\ l\ s$
from *this*[*unfolded wp1_def*]
obtain $t\ s'\ t'\ s''$ **where** $(c,\ s) \Rightarrow t \Downarrow s'\ (WHILE\ b\ DO\ c,\ s') \Rightarrow t' \Downarrow s''$ **and** *Q*: $Q\ l\ s''$ **by** *blast*
with *b* **have** $(WHILE\ b\ DO\ c,\ s) \Rightarrow 1{+}t{+}t' \Downarrow s''$ **by** *auto*
with *Q* **show** $wp_1\ (WHILE\ b\ DO\ c)\ Q\ l\ s$ **unfolding** *wp1_def* **by** *auto*
next
assume $wp_1\ (WHILE\ b\ DO\ c)\ Q\ l\ s$
from *this*[*unfolded wp1_def*]
obtain $t\ s''$ **where** $(WHILE\ b\ DO\ c,\ s) \Rightarrow t \Downarrow s''$ **and** *Q*: $Q\ l\ s''$ **by** *blast*
with *b* **obtain** $t1\ t2\ s'$ **where** $(c,\ s) \Rightarrow t1 \Downarrow s'\ (WHILE\ b\ DO\ c,\ s') \Rightarrow t2 \Downarrow s''$ **by** *auto*
with *Q* **show** $wp_1\ c\ (wp_1\ (WHILE\ b\ DO\ c)\ Q)\ l\ s$ **unfolding** *wp1_def* **by** *auto*
qed

lemma assumes *b*: ~ *bval b s*
shows *wp1WhileFalse'*: $wp_1\ (WHILE\ b\ DO\ c)\ Q\ l\ s = Q\ l\ s$

proof
  assume wp_1 (*WHILE b DO c*) *Q l s*
  from *this*[*unfolded wp1_def*]
  obtain *t s'* **where** (*WHILE b DO c*, *s*) ⇒ *t* ⇓ *s'* **and** *Q*: *Q l s'* **by** *blast*
  with *b* **have** *s=s'* **by** *auto*
  with *Q* **show** *Q l s* **by** *auto*
next
  assume *Q l s*
  with *b* **show** wp_1 (*WHILE b DO c*) *Q l s* **unfolding** *wp1_def* **by** *auto*
qed

lemma *wp1While*: wp_1 (*WHILE b DO c*) *Q l s* = (*if bval b s then* wp_1 *c* (wp_1 (*WHILE b DO c*) *Q*) *l s else Q l s*)
  apply(*cases bval b s*)
  using *wp1WhileTrue'* **apply** *simp*
  using *wp1WhileFalse'* **apply** *simp* **done**

lemma *wp1_prec2*: **fixes** *e*::*tbd*
  shows (wp_1 *c1 Q l s* ∧
    l x = *prec c1 e s*) = wp_1 *c1* (λ*l s*. *Q l s* ∧ *e s* = *l x*) *l s*
  by (*metis* (*mono_tags*, *lifting*) *Big_StepT.bigstepT_the_state prec_def wp1_def*)

lemma *wp1_prec*: **fixes** *e*::*tbd*
  shows wp_1 *c1 Q l s* ⟹
    l x = *prec c1 e s* ⟹ wp_1 *c1* (λ*l s*. *Q l s* ∧ *e s* = *l x*) *l s*
unfolding *wp1_def prec_def* **apply**(*auto*)
proof −
  fix *p t*
  assume *l x* = *e* (*THE t*. ∃ *p*. (*c1*, *s*) ⇒ *p* ⇓ *t*)
  assume *2*: *Q l t*
  assume *1*: (*c1*, *s*) ⇒ *p* ⇓ *t*
  show ∃ *t*. (∃ *p*. (*c1*, *s*) ⇒ *p* ⇓ *t*) ∧ *Q l t* ∧ *e t* = *e* (*THE t*. ∃ *p*. (*c1*, *s*) ⇒ *p* ⇓ *t*)
    apply(*rule exI*[**where** *x=t*])
    apply(*safe*)
      apply(*rule exI*[**where** *x=p*]) **using** *1* **apply** *simp*
      apply(*fact*)
      using *1* **by**(*simp add*: *bigstepT_the_state*)
qed

lemma *wp1_is_pre*: *finite* (*support Q*) ⟹ $\vdash_1$ {wp_1 *c Q*} *c* { λ*s*. $\downarrow_t$ (*c*, *s*)

$\Downarrow$ *Q*}
proof (*induction c arbitrary*: *Q*)
  case *SKIP*
  have *ff*: $\bigwedge s\ n.\ (\exists a.\ (SKIP,\ s) \Rightarrow n \Downarrow a) = (n{=}Suc\ 0)$ **by** *blast*
  show *?case* **apply** (*auto intro*:*hoare1.Skip simp add*: *ff*) **using** *ff* **done**
next
  have *gg*: $\bigwedge x1\ x2\ s\ n.\ (\exists a.\ (x1 ::= x2,\ s) \Rightarrow n \Downarrow a) = (n{=}Suc\ 0)$ **by** *blast*
  case *Assign* **show** *?case* **apply** (*auto intro*:*hoare1.Assign simp add*: *gg*)
done
next
  case (*Seq c1 c2*)
  — choose a fresh logical variable x
  let *?x* = *new Q*
  have $\exists x.\ x \notin$ *support Q* **using** *Seq.prems infinite_UNIV_listI*
    using *ex_new_if_finite* **by** *blast*
  hence *?x* $\notin$ *support Q* **by** (*rule someI_ex*)
  then have *x2*: *?x* $\notin$ *support* (wp_1 *c2 Q*) **using** *support_wpt* **by** (*fast*)
  then have *x12*: *?x* $\notin$ *support* (wp_1 (*c1*;;*c2*) *Q*) **apply** *simp* **using** *support_wpt* **by** *fast*

  — assemble a postcondition Q1 that ensures the weakest precondition of Q before c2 and saves the running time of c2 into the logical variable x
  let *?Q1* = ($\lambda l\ s.$ (wp_1 *c2 Q*) *l s* $\wedge$ $\downarrow_t$ (*c2*, *s*) = *l ?x*)
  have *finite* (*support ?Q1*) **apply**(*rule rev_finite_subset*[*OF _ support_and*])
    apply(*rule finite_UnI*)
    apply(*rule rev_finite_subset*[*OF _ support_wpt*]) **apply**(*fact*)
    apply(*rule rev_finite_subset*[*OF _ support_single*]) **by** *simp*
  — we can now specify this Q1 in the first Induction Hypothesis
  then have *pre*: $\bigwedge u.\ \vdash_1$ {wp_1 *c1 ?Q1* } *c1* { $\lambda s.\ \downarrow_t$ (*c1*, *s*) $\Downarrow$ *?Q1* }
    using *Seq*(*1*) **by** *simp*

  — we can rewrite this into the form we need for the Seq rule
  have *A*: $\vdash_1$ {$\lambda l\ s.\ wp_1$ (*c1*;;*c2*) *Q l s* $\wedge$ *l ?x* = (*prec c1* (%*s*. $\downarrow_t$ (*c2*, *s*))) *s*} *c1* { $\lambda s.\ \downarrow_t$ (*c1*, *s*) $\Downarrow$ $\lambda l\ s.\ wp_1$ *c2 Q l s* $\wedge$ $\downarrow_t$ (*c2*, *s*) $\leq$ *l ?x*}
    apply(*rule conseq_old*[*OF _ pre*])
    by(*auto simp add*: *wp1_prec*)

  — we can now apply the Seq rule with the first IH (in the right shape A) and the second IH
  show $\vdash_1$ {wp_1 (*c1*;; *c2*) *Q*} *c1*;; *c2* { $\lambda s.\ \downarrow_t$ (*c1*;; *c2*, *s*) $\Downarrow$ *Q*}
    apply(*rule hoare1.Seq*[*OF A Seq*(*2*)])
      — finally some side conditions have to be proven
      using *Seq*(*3*) *x12 x2 wp1_prec_Seq_correct* **.**
next

case (*If b c1 c2*)

show *?case* **apply**(*simp*)
apply(*rule If2*[**where** *e=%s. if bval b s then* $\downarrow_t$ (*c1*, *s*) *else* $\downarrow_t$ (*c2*, *s*)])
apply(*simp_all cong*:*rev_conj_cong*)
apply(*rule conseq_old*[**where** *Q*=*Q* **and** Q'=*Q*])
prefer *2*
apply(*rule If.IH*(*1*)) **apply**(*fact*)
apply(*simp_all*) **apply**(*auto*)[*1*]
apply(*rule conseq_old*[**where** *Q*=*Q* **and** Q'=*Q*])
prefer *2*
apply(*rule If.IH*(*2*)) **apply**(*fact*)
apply(*simp_all*) **apply**(*auto*)[*1*]
apply (*blast intro*: *If_THE_True wp1_terminates If_THE_False*)
done

next
case (*While b c*)

let *?y* = (*new* (wp_1 (*WHILE b DO c*) *Q*))
have *finite* (*support* (wp_1 (*WHILE b DO c*) *Q*))
apply(*rule finite_subset*[*OF support_wpt*]) **apply** *fact* **done**
then have $\exists$ *x*. *x* $\notin$ *support* (wp_1 (*WHILE b DO c*) *Q*) **using** *infinite_UNIV_listI*
using *ex_new_if_finite* **by** *blast*
hence *yQx*: *?y* $\notin$ *support* (wp_1 (*WHILE b DO c*) *Q*) **by** (*rule someI_ex*)

show *?case*
proof (*rule conseq_old*[*OF _ hoare1.While*], *safe*)
show $\exists$ *k*>*0*. $\forall$ *l s*. wp_1 (*WHILE b DO c*) *Q l s* $\longrightarrow$ wp_1 (*WHILE b DO c*) *Q l s* $\wedge$ $\downarrow_t$ (*WHILE b DO c*, *s*) $\leq$ *k* $*$ $\downarrow_t$ (*WHILE b DO c*, *s*)
apply *auto* **done**
next
fix *l s*
assume wp_1 (*WHILE b DO c*) *Q l s* $\neg$ *bval b s*
then show *Q l s* **by** (*simp add*: *wp1While*)
next
fix *l s*
assume wp_1 (*WHILE b DO c*) *Q l s*
from *this*[*unfolded wp1_def*] **obtain** *t* s' **where** *t*: (*WHILE b DO c*, *s*) $\Rightarrow$ *t* $\Downarrow$ s' **and** *Q l* s' **by** *blast*
then have *r*: $\downarrow_t$ (*WHILE b DO c*, *s*) = *t* **using** *Nielson_Hoare.bigstepT_the_cost* **by** *auto*
assume $\neg$ *bval b s*

with *r t* **have** *t2*: *t=1* **by** *auto*
from *r t2* **show** *1* ≤ ↓$_t$ (*WHILE b DO c*, *s*) **by** *auto*
next
fix *l s*
assume *wp*$_1$ (*WHILE b DO c*) *Q l s*
from *this*[*unfolded wp1_def*] **obtain** *t s''* **where** *t*: (*WHILE b DO c*, *s*) ⇒ *t* ⇓ *s''* *Q l s''* **by** *blast*
then have *r*: ↓$_t$ (*WHILE b DO c*, *s*) = *t* **using** *Nielson_Hoare.bigstepT_the_cost* **by** *auto*
assume *bval b s*
with *t* **obtain** *t1 t2 s'* **where** *1*: (*c*,*s*) ⇒ *t1* ⇓ *s'* **and** *2*: (*WHILE b DO c*, *s'*) ⇒ *t2* ⇓ *s''* **and** *sum*: *t=t1+t2+1* **and** *bval b s* **by** *auto*
from *1* **have** *A*: ↓$_t$ (*c*,*s*) = *t1* **and** *s'*: ↓$_s$ (*c*,*s*) = *s'* **using** *Nielson_Hoare.bigstepT_the_cost bigstepT_the_state* **by** *auto*
from *2 s'* **have** *B*: ↓$_t$ (*WHILE b DO c*, ↓$_s$(*c*,*s*)) = *t2* **using** *Nielson_Hoare.bigstepT_the_cost* **by** *auto*

show *1* + (%*s*. ↓$_t$ (*WHILE b DO c*, ↓$_s$(*c*,*s*))) *s* + (%*s*. ↓$_t$ (*c*,*s*)) *s* ≤ ↓$_t$ (*WHILE b DO c*, *s*)
apply(*simp add*: *r A B sum*) **done**
next

show ⊢$_1$ {λ*l s*. *wp*$_1$ (*WHILE b DO c*) *Q l s* ∧ *bval b s* ∧ ↓$_t$ (*WHILE b DO c*, ↓$_s$ (*c*, *s*)) = *l ?y*} *c*
{ λ*s*. ↓$_t$ (*c*, *s*) ⇓ λ*l s*. *wp*$_1$ (*WHILE b DO c*) *Q l s* ∧ ↓$_t$ (*WHILE b DO c*, *s*) ≤ *l ?y*}
apply(*rule conseq_old*[*OF _ While*(*1*), *of _ %l s*. *wp*$_1$ (*WHILE b DO c*) *Q l s* ∧ ↓$_t$ (*WHILE b DO c*, *s*) = *l ?y*])
apply(*rule exI*[**where** *x=1*]) **apply** *simp*
subgoal apply *safe*
apply(*subst* (*asm*) *wp1While*) **apply** *simp*
proof − **fix** *l s*
assume *1*: *wp*$_1$ *c* (*wp*$_1$ (*WHILE b DO c*) *Q*) *l s*
assume *2*: ↓$_t$ (*WHILE b DO c*, ↓$_s$ (*c*, *s*)) = *l ?y*
then have *l ?y* = *prec c* (%*s*. ↓$_t$ (*WHILE b DO c*, *s*)) *s* **unfolding** *prec_def* **by** *auto*
with *1 wp1_prec2*[*of c* (*wp*$_1$ (*WHILE b DO c*) *Q*) *l s _* (λ*s*. ↓$_t$ (*WHILE b DO c*, *s*))]
show *wp*$_1$ *c* (λ*l s*. *wp*$_1$ (*WHILE b DO c*) *Q l s* ∧ ↓$_t$ (*WHILE b DO c*, *s*) = *l ?y*) *l s* **by** *auto*
qed
subgoal apply(*rule finite_subset*[*OF support_and*]) **apply** *auto*
apply(*rule finite_subset*[*OF support_wpt*]) **apply** *fact*

apply(*rule finite_subset*) **apply**(*rule support_single*) **by** *auto*
apply *auto* **done**
next
assume *new* (wp_1 (*WHILE b DO c*) *Q*) ∈ *support* (wp_1 (*WHILE b DO c*) *Q*)
with *yQx* **show** *False*
by *blast*

qed
qed

lemma *valid_wp*: $\models_1$ $\{P\}c\{p \Downarrow Q\} \longleftrightarrow (\exists k>0.\ (\forall l\ s.\ P\ l\ s \longrightarrow (wp_1\ c\ Q\ l\ s \wedge ((THE\ n.\ (\exists\ t.\ ((c,s) \Rightarrow n \Downarrow t)))) \leq k * p\ s)))$
apply(*rule*)
apply(*auto simp*: *hoare1_valid_def wp1_def*)
subgoal for *k* **apply**(*rule exI*[**where** *x*=*k*]) **using** *bigstepT_the_cost* **by** *fast*
subgoal for *k* **apply**(*rule exI*[**where** *x*=*k*]) **using** *bigstepT_the_cost* **by** *fast*
done

theorem *hoare1_complete*: *finite* (*support Q*) $\Longrightarrow \models_1 \{P\}c\{p \Downarrow Q\} \Longrightarrow \vdash_1 \{P\}c\{p \Downarrow Q\}$
apply(*rule conseq_old*[*OF _ wp1_is_pre*, **where** Q'=*Q* **and** *Q*=*Q*, *simplified*])
by (*auto simp*: *valid_wp*)

corollary *hoare1_sound_complete*: *finite* (*support Q*) $\Longrightarrow \vdash_1 \{P\}c\{p \Downarrow Q\} \longleftrightarrow \models_1 \{P\}c\{p \Downarrow Q\}$
by (*metis hoare1_sound hoare1_complete*)

end

theory *Nielson_VCG* **imports** *Nielson_Hoare* **begin**

4.6 Verification Condition Generator

Annotated commands: commands where loops are annotated with invariants.

datatype *acom* =

Askip (‹*SKIP*›) |
Aassign vname aexp (‹(_ ::= _)› [*1000*, *61*] *61*) |
Aseq acom acom (‹_;;/ _› [*60*, *61*] *60*) |
Aif bexp acom acom (‹(*IF* _/ *THEN* _/ *ELSE* _)› [*0*, *0*, *61*] *61*) |
Aconseq assn2 assn2 tbd acom
(‹({_'/_'/_}/ *CONSEQ* _)› [*0*, *0*, *0*, *61*] *61*)|
Awhile (*assn2*)∗((*state*⇒*state*)∗(*tbd*)) *bexp acom* (‹({_}/ *WHILE* _/ *DO* _)› [*0*, *0*, *61*] *61*)

notation *com.SKIP* (‹*SKIP*›)

Strip annotations:

fun *strip* :: *acom* ⇒ *com* **where**
strip SKIP = *SKIP* |
strip (*x* ::= *a*) = (*x* ::= *a*) |
strip (C_1;; C_2) = (*strip* C_1;; *strip* C_2) |
strip (*IF b THEN* C_1 *ELSE* C_2) = (*IF b THEN strip* C_1 *ELSE strip* C_2) |
strip ({_/_/_} *CONSEQ C*) = *strip C* |
strip ({_} *WHILE b DO C*) = (*WHILE b DO strip C*)

support of an expression

4.6.1 support and supportE

definition *supportE* :: ((*char list* ⇒ *nat*) ⇒ (*char list* ⇒ *int*) ⇒ *nat*) ⇒ *string set* **where**
supportE P = {*x*. ∃ *l1 l2 s*. (∀ *y*. *y* ≠ *x* ⟶ *l1 y* = *l2 y*) ∧ *P l1 s* ≠ *P l2 s*}

lemma *expr_lupd*: *x* ∉ *supportE Q* ⟹ *Q* (*l*(*x*:=*n*)) = *Q l*
by(*simp add*: *supportE_def fun_upd_other fun_eq_iff*)
(*metis* (*no_types*, *lifting*) *fun_upd_def*)

lemma *supportE_if*: *supportE* (λ*l s*. *if b s then A l s else B l s*)
⊆ *supportE A* ∪ *supportE B*
unfolding *supportE_def* **apply**(*auto*)
by *metis*+

lemma *support_eq*: *support* (λ*l s*. *l x* = *E l s*) ⊆ *supportE E* ∪ {*x*}
unfolding *support_def supportE_def*
apply(*auto*)

apply *blast*
by *metis*

lemma *support_impl_in*: $G\ e \longrightarrow support\ (\lambda l\ s.\ \ H\ e\ l\ s) \subseteq T$
$\Longrightarrow support\ (\lambda l\ s.\ G\ e \longrightarrow\ \ H\ e\ l\ s) \subseteq T$
unfolding *support_def* **apply**(*auto*)
apply *blast*+ **done**

lemma *support_supportE*: $\bigwedge P\ e.\ support\ (\lambda l\ s.\ \ P\ (e\ l)\ s) \subseteq supportE\ e$
unfolding *support_def supportE_def*
apply(*rule subsetI*)
apply(*simp*)
proof (*clarify*, *goal_cases*)
case (*1 P e x l1 l2 s*)
have *P*: $\forall s.\ e\ l1\ s = e\ l2\ s \Longrightarrow e\ l1 = e\ l2$ **by** *fast*
show $\exists l1\ l2.\ (\forall y.\ y \neq x \longrightarrow l1\ y = l2\ y) \wedge (\exists s.\ e\ l1\ s \neq e\ l2\ s)$
apply(*rule exI*[**where** *x*=*l1*])
apply(*rule exI*[**where** *x*=*l2*])
apply(*safe*)
using *1* **apply** *blast*
apply(*rule ccontr*)
apply(*simp*)
using *1*(*2*) *P* **by** *force*
qed

— collects the logical variables in the Invariants and Loop Bodies as well as the annotated assertions at CONSEQs of an annotated command
fun *varacom* :: *acom* ⇒ *lvname set* **where**
$varacom\ (C_1;;\ C_2) = varacom\ C_1 \cup varacom\ C_2$
$|\ varacom\ (IF\ b\ THEN\ C_1\ ELSE\ C_2) = varacom\ C_1 \cup varacom\ C_2$
| *varacom* ({*P*/*Qannot*/_} *CONSEQ C*)= *support P* ∪ *varacom C* ∪ *support Qannot*
| *varacom* ({(*I*,(*S*,(*E*)))} *WHILE b DO C*) = *support I* ∪ *varacom C*
| *varacom* _ = {}

Weakest precondition from annotated commands:

fun *preT* :: *acom* ⇒ *tbd* ⇒ *tbd* **where**
preT SKIP e = *e* |
$preT\ (x ::= a)\ e = (\lambda s.\ e(s(x := aval\ a\ s)))$ |
$preT\ (C_1;;\ C_2)\ e = preT\ C_1\ (preT\ C_2\ e)$ |
preT ({_/_/_} *CONSEQ C*) *e* = *preT C e* |
$preT\ (IF\ b\ THEN\ C_1\ ELSE\ C_2)\ e =$
$(\lambda s.\ if\ bval\ b\ s\ then\ preT\ C_1\ e\ s\ else\ preT\ C_2\ e\ s)$ |

```
  preT ({(_,(S,_))} WHILE b DO C) e = e o S


lemma preT_linear: preT C (%s. k * e s) = (%s. k * preT C e s)
by (induct C arbitrary: e, auto)

fun postQ :: acom ⇒ state ⇒ state where
  postQ SKIP s = s |
  postQ (x ::= a) s =  s(x := aval a s) |
  postQ (C1;; C2) s = postQ C2 (postQ C1 s) |
  postQ ({_/_/_} CONSEQ C) s = postQ C s |
  postQ (IF b THEN C1 ELSE C2) s =
  (if bval b s then postQ C1 s else postQ C2 s) |
  postQ ({(_,(S,_))} WHILE b DO C) s = S s

lemma TQ: preT C e s = e (postQ C s)
  apply(induct C arbitrary: e s) by (auto)


function (domintros) times :: state ⇒ bexp ⇒ acom ⇒ nat where
  times s b C = (if bval b s then Suc (times (postQ C s) b C) else 0)
   apply(auto) done




lemma assumes I: I z s and
  i:   ⋀s z. I (Suc z) s ⟹ bval b s ∧ I z (postQ C s)
  and  ii:  ⋀s. I 0 s ⟹ ~ bval b s
shows times_z: times s b C = z
proof -
  have I z s ⟹ times_dom (s, b, C) ∧ times s b C = z
  proof(induct z arbitrary: s)
    case 0
    have A: times_dom (s, b, C)
      apply(rule times.domintros)
      apply(simp add:  ii[OF 0] ) done
    have B: times s b C = 0
      using times.psimps[OF A] by(simp add:  ii[OF 0])

    show ?case using A B by simp
  next
    case (Suc z)
    from i[OF Suc(2)] have bv: bval b s
```

and *g*: *I z* (*postQ C s*) **by** *simp_all*
from *Suc*(*1*)[*OF g*] **have** *p1*: *times_dom* (*postQ C s*, *b*, *C*)
and *p2*: *times* (*postQ C s*) *b C* = *z* **by** *simp_all*
have *A*: *times_dom* (*s*, *b*, *C*)
apply(*rule times.domintros*) **apply**(*rule p1*) **done**
have *B*: *times s b C* = *Suc z*
using *times.psimps*[*OF A*] *bv p2* **by** *simp*
show *?case* **using** *A B* **by** *simp*
qed

then show *times s b C* = *z* **using** *I* **by** *simp*
qed

function (*domintros*) *postQs* :: *acom* ⇒ *bexp* ⇒ *state* ⇒ *state* **where**
postQs C b s = (*if bval b s then* (*postQs C b* (*postQ C s*)) *else s*)
apply(*auto*) **done**

fun *postQz* :: *acom* ⇒ *state* ⇒ *nat* ⇒ *state* **where**
postQz C s 0 = *s* |
postQz C s (*Suc n*) = (*postQz C* (*postQ C s*) *n*)

fun *preTz* :: *acom* ⇒ *tbd* ⇒ *nat* ⇒ *tbd* **where**
preTz C e 0 = *e* |
preTz C e (*Suc n*) = *preT C* (*preTz C e n*)

lemma *TzQ*: *preTz C e n s* = *e* (*postQz C s n*)
by (*induct n arbitrary*: *s*, *simp_all add*: *TQ*)

4.6.2 Weakest precondition from annotated commands:

fun *pre* :: *acom* ⇒ *assn2* ⇒ *assn2* **where**
pre SKIP Q = *Q* |
pre (*x* ::= *a*) *Q* = (λ*l s*. *Q l* (*s*(*x* := *aval a s*))) |
pre (C_1;; C_2) *Q* = *pre* C_1 (*pre* C_2 *Q*) |
pre ({*P'*/_/_} *CONSEQ C*) *Q* = *P'* |
pre (*IF b THEN* C_1 *ELSE* C_2) *Q* =
(λ*l s*. *if bval b s then pre* C_1 *Q l s else pre* C_2 *Q l s*) |
pre ({(*I*,(*S*,(*E*)))} *WHILE b DO C*) *Q* = *I*

lemma *supportE_preT*: *supportE* (%*l*. *preT* *C* (*e* *l*)) ⊆ *supportE* *e*
proof(*induct* *C* *arbitrary*: *e*)
  case (*Aif* *b* *C1* *C2* *e*)
  show *?case*
    apply(*simp*)
    apply(*rule* *subset_trans*[*OF* *supportE_if*])
    using *Aif* **by** *fast*
next
  case (*Awhile* *A* *y* *C* *e*)
  obtain *I* *S* *E* **where** *A*: *A*= (*I*,*S*,*E*) **using** *prod_cases3* **by** *blast*
  show *?case* **using** *A* **apply**(*simp*) **unfolding** *supportE_def*
    by *blast*
next
  case (*Aseq*)
  then show *?case* **by** *force*
qed (*simp_all* *add*: *supportE_def*, *blast*)

lemma *supportE_twicepreT*: *supportE* (%*l*. *preT* *C1* (*preT* *C2* (*e* *l*))) ⊆ *supportE* *e*
  by (*rule* *subset_trans*[*OF* *supportE_preT* *supportE_preT*])

lemma *supportE_preTz*: *supportE* (%*l*. *preTz* *C* (*e* *l*) *n*) ⊆ *supportE* *e*
proof (*induct* *n*)
  case (*Suc* *n*)
  show *?case*
    apply(*simp*)
    apply(*rule* *subset_trans*[*OF* *supportE_preT*])
    by *fact*
qed *simp*

lemma *supportE_preTz_Un*:
  supportE (λ*l*. *preTz* *C* (*e* *l*) (*l* *x*)) ⊆ *insert* *x* (*UN* *n*. *supportE* (λ*l*. *preTz* *C* (*e* *l*) *n*))
  apply(*auto* *simp* *add*: *supportE_def* *subset_iff*)
  apply *metis*
  done

lemma *supportE_preTz2*: *supportE* (%*l*. *preTz* *C* (*e* *l*) (*l* *x*)) ⊆ *insert* *x* (*supportE* *e*)

apply(*rule subset_trans*[*OF supportE_preTz_Un*])
using *supportE_preTz* **by** *blast*

lemma *pff*: $\bigwedge n.\ support\ (\lambda l.\ I\ (l(x := n))) \subseteq support\ I - \{x\}$
unfolding *support_def* **apply**(*auto*) **using** *fun_upd_apply* **apply** *smt*
apply (*smt fun_upd_apply*) **oops**

lemma *pff*: $\bigwedge n.\ support\ (\lambda l.\ I\ (l(x := n))) \subseteq support\ I$
unfolding *support_def* **apply**(*auto*) **using** *fun_upd_apply* **apply** *smt*
by (*smt fun_upd_apply*)

lemma *supportAB*: $support\ (\lambda l\ s.\ A\ l\ s \wedge B\ s) \subseteq support\ A$
apply(*rule subset_trans*[*OF support_and*])
by (*simp add*: *support_inv*)

lemma $support\ (pre\ (\{(I,(S,(E\ )))\}\ WHILE\ b\ DO\ C)\ Q) \subseteq support\ I$
by (*simp add*: *supportAB*)

lemma *support_pre*: $support\ (pre\ C\ Q) \subseteq support\ Q \cup varacom\ C$
proof (*induct C arbitrary*: *Q*)
case (*Awhile A b C Q*)
obtain *I S E* **where** *A*: $A= (I,(S,(E\ )))$ **using** *prod_cases3* **by** *blast*
have *support_inv*: $\bigwedge P.\ support\ (\lambda l\ s.\ P\ s) = \{\}$
unfolding *support_def* **by** *blast*
show *?case* **unfolding** *A* **apply**(*simp*) **using** *supportAB* **by** *fast*
next
case (*Aseq C1 C2*)
then show *?case* **by**(*auto*)
next
case (*Aif x C1 C2 Q*)
have *s1*: $support\ (\lambda l\ s.\ bval\ x\ s \longrightarrow pre\ C1\ Q\ l\ s) \subseteq support\ Q \cup varacom\ C1$
apply(*rule subset_trans*[*OF support_impl*]) **by**(*rule Aif*)
have *s2*: $support\ (\lambda l\ s.\ \sim\ bval\ x\ s \longrightarrow pre\ C2\ Q\ l\ s) \subseteq support\ Q \cup varacom\ C2$
apply(*rule subset_trans*[*OF support_impl*]) **by**(*rule Aif*)

show *?case* **apply**(*simp*)
apply(*rule subset_trans*[*OF support_and*])
using *s1 s2* **by** *blast*
next

case (*Aconseq x1 x2 x3 C*)
then show *?case* **by**(*auto*)
qed (*auto simp add*: *support_def*)

lemma *finite_support_pre*[*simp*]: *finite* (*support Q*) $\Longrightarrow$ *finite* (*varacom C*) $\Longrightarrow$ *finite* (*support* (*pre C Q*))
using *finite_subset support_pre finite_UnI* **by** *metis*

fun *time* :: *acom* $\Rightarrow$ *tbd* **where**
time SKIP = (%*s*. *Suc 0*) |
time (*x* ::= *a*) = (%*s*. *Suc 0*) |
time (C_1;; C_2) = (%*s*. *time* C_1 *s* + *preT* C_1 (*time* C_2) *s*) |
time ({_/_/*e*} *CONSEQ C*) = *e* |
time (*IF b THEN* C_1 *ELSE* C_2) =
(λ*s*. *if bval b s then 1* + *time* C_1 *s else 1* + *time* C_2 *s*) |
time ({(_,(*E'*,(*E*)))} *WHILE b DO C*) = *E*

lemma *supportE_single*: *supportE* (λ*l s*. *P*) = {}
unfolding *supportE_def* **by** *blast*

lemma *supportE_plus*: *supportE* (λ*l s*. *e1 l s* + *e2 l s*) $\subseteq$ *supportE e1* $\cup$ *supportE e2*
unfolding *supportE_def* **apply**(*auto*)
by *metis*

lemma *supportE_Suc*: *supportE* (λ*l s*. *Suc* (*e1 l s*)) = *supportE e1*
unfolding *supportE_def* **by** (*auto*)

lemma *supportE_single2*: *supportE* (λ*l* . *P*) = {}
unfolding *supportE_def* **by** *blast*

lemma *supportE_time*: *supportE* (λ*l*. *time C*) = {}
using *supportE_single2* **by** *simp*

lemma $\bigwedge$*s*. ($\forall$ *l*. *I* (*l*(*x*:=*0*)) *s*) = ($\forall$ *l*. *l x* = *0* $\longrightarrow$ *I l s*)
apply(*auto*)
by (*metis fun_upd_triv*)

lemma $\bigwedge$*s*. ($\forall$ *l*. *I* (*l*(*x*:=*Suc* (*l x*))) *s*) = ($\forall$ *l*. ($\exists$ *n*. *l x* = *Suc n*) $\longrightarrow$ *I l s*)
apply(*auto*)

proof (*goal_cases*)
  case (*1 s l n*)
  then have $\bigwedge l.\ I\ (l(x := Suc\ (l\ x)))\ s$ **by** *simp*
  from *this*[**where** $l{=}l(x{:=}n)$]
  have $I\ ((l(x{:=}n))(x := Suc\ ((l(x{:=}n))\ x)))\ s$ **by** *simp*
  then show *?case* **using** *1(2)* **apply**(*simp*)
    by (*metis fun_upd_triv*)
qed

Verification condition:

fun $vc :: acom \Rightarrow assn2\ \Rightarrow bool$ **where**
  $vc\ SKIP\ Q = True$ |
  $vc\ (x ::= a)\ Q = True$ |
  $vc\ (C_1\ ;;\ C_2)\ Q = ((vc\ C_1\ (pre\ C_2\ Q)) \wedge (vc\ C_2\ Q)\ )$ |
  $vc\ (IF\ b\ THEN\ C_1\ ELSE\ C_2)\ Q = (vc\ C_1\ Q \wedge vc\ C_2\ Q)$ |
  $vc\ (\{P'/Q/e'\}\ CONSEQ\ C)\ Q' = (vc\ C\ Q \wedge (\exists k{>}0.\ (\forall l\ s.\ P'\ l\ s \longrightarrow time\ C\ s \leq k * e'\ s\ \wedge (\forall t.\ \exists l'.\ (pre\ C\ Q)\ l'\ s \wedge (\ Q\ l'\ t \longrightarrow Q'\ l\ t)\ ))))$ |

  $vc\ (\{(I,(S,(E)))\}\ WHILE\ b\ DO\ C)\ Q =$
  $((\forall l\ s.\ (I\ l\ s \wedge bval\ b\ s \longrightarrow\ pre\ C\ I\ l\ s \wedge\ \ E\ s \geq 1 + preT\ C\ E\ s + time\ C\ s$
  $\wedge\ S\ s = S\ (postQ\ C\ s)) \wedge$
  $(I\ l\ s \wedge \neg\ bval\ b\ s \longrightarrow Q\ l\ s \wedge E\ s \geq 1\ \wedge S\ s = s)\ ) \wedge$
  $vc\ C\ I)$

lemma *pre_mono*:
  $(\forall l\ s.\ P\ l\ s \longrightarrow P'\ l\ s) \Longrightarrow pre\ C\ P\ l\ s \Longrightarrow pre\ C\ P'\ l\ s$
proof (*induction C arbitrary*: $P\ P'\ l\ s$)
  case (*Aseq C1 C2*)
  then have *A*: *pre C1* (*pre C2 P*) *l s* **by**(*simp*)
  from *Aseq*(*2*)[*OF Aseq*(*3*)] *Aseq*(*1*)[*OF _ A*]
  show *?case* **by** *simp*
next
  case (*Awhile A b C*)
  then obtain *I S E* **where** *A*: $A = (I,S,E\ )$ **using** *prod_cases3* **by** *blast*
  from *Awhile* **show** *?case* **unfolding** *A* **by** *simp*
qed *simp_all*

lemma *vc_mono*: $(\forall l\ s.\ P\ l\ s \longrightarrow P'\ l\ s) \Longrightarrow vc\ C\ P \Longrightarrow vc\ C\ P'$
  apply (*induct C arbitrary*: $P\ P'$)
      apply *auto*
  subgoal using *pre_mono* **by** *metis*
  subgoal using *pre_mono* **by** *metis*

done

4.6.3 Soundness:

abbreviation *preSet U C l s* == (*Ball U* (%*u. case u of* (*x*,*e*) ⇒ *l x* = *preT C e s*))
abbreviation *postSet U l s* == (*Ball U* (%*u. case u of* (*x*,*e*) ⇒ *l x* = *e s*))

fun *ListUpdate* **where**
ListUpdate f [] *l* = *f*
| *ListUpdate f* ((*x*,*e*)#*xs*) *q* = (*ListUpdate f xs q*)(*x*:=*q e x*)

lemma *allg*:
assumes *U2*: $\bigwedge$*l s n x. x*∈ *fst* ' *upds* ⟹ *A* (*l*(*x* := *n*)) = *A l*
shows
fst ' *set xs* ⊆ *fst* ' *upds* ⟹ *A* (*ListUpdate l'' xs q*) = *A l''*
proof (*induct xs*)
case (*Cons a xs*)
obtain *x e* **where** *axe*: *a* = (*x*,*e*) **by** *fastforce*
have *A* (*ListUpdate l''* (*a* # *xs*) *q*)
= *A* ((*ListUpdate l'' xs q*)(*x* := *q e x*)) **unfolding** *axe* **by**(*simp*)
also have
... = *A* (*ListUpdate l'' xs q*)
apply(*rule U2*)
using *Cons axe* **by** *force*
also have ... = *A l''*
using *Cons* **by** *force*
finally show *?case* **.**
qed *simp*

fun *ListUpdateE* **where**
ListUpdateE f [] = *f*
| *ListUpdateE f* ((*x*,*v*)#*xs*) = (*ListUpdateE f xs*)(*x*:=*v*)

lemma *ListUpdate_E*: *ListUpdateE f xs* = *ListUpdate f xs* (%*e x. e*)
apply(*induct xs*) **apply**(*simp_all*)
subgoal for *a xs* **apply**(*cases a*) **apply**(*simp*) **done**
done
lemma *allg_E*: **fixes** *A*::*assn2*
assumes
($\bigwedge$*l s n x. x* ∈ *fst* ' *upds* ⟹ *A* (*l*(*x* := *n*)) = *A l*) *fst* ' *set xs* ⊆ *fst* ' *upds*
shows *A* (*ListUpdateE f xs*) = *A f*
proof −

```
  have  A (ListUpdate f xs (%e x. e)) = A f
    apply(rule allg)
    apply fact+ done
  then show ?thesis by(simp only: ListUpdate_E)
qed

lemma ListUpdateE_updates: distinct (map fst xs) ⟹ x ∈ set xs ⟹
ListUpdateE l'' xs (fst x) = snd x
proof (induct xs)
  case Nil
  then show ?case apply(simp) done
next
  case (Cons a xs)
  show ?case
  proof (cases fst a = fst x)
    case True
    then obtain y e where a: a=(y,e) by fastforce
    with True have fstx: fst x=y by simp
    from Cons(2,3) fstx  a have a2: x=a
      by force
    show ?thesis unfolding a2 a by(simp)
  next
    case False
    with Cons(3) have A: x∈set xs by auto
    obtain y e where a: a=(y,e) by fastforce
    from Cons(2) have B: distinct (map fst xs) by simp
    from Cons(1)[OF B A] False
      show ?thesis unfolding a by(simp)
  qed
qed


lemma ListUpdate_updates: x ∈ fst ` (set xs) ⟹ ListUpdate l'' xs (%e. l)
x = l x
proof(induct xs)
  case Nil
  then show ?case by(simp)
next
  case (Cons a xs)
  obtain q p where axe: a = (p,q) by fastforce
  from Cons show ?case unfolding axe
    apply(cases x=p)
    by(simp_all)
qed
```

```
abbreviation lesvars xs == fst ` (set xs)

fun preList where
  preList [] C l s = True
| preList ((x,e)#xs) C l s = (l x = preT C e s ∧ preList xs C l s)

lemma preList_Seq: preList upds (C1;; C2) l s = preList (map (λ(x, e).
(x, preT C2 e)) upds) C1 l s
proof (induct upds)
  case Nil
  then show ?case by simp
next
  case (Cons a xs)
  obtain y e where a: a=(y,e) by fastforce
  from Cons show ?case unfolding a by (simp)
qed

lemma support_True[simp]: support (λa b. True) = {}
  unfolding support_def
  by fast

lemma support_preList: support (preList upds C1) ⊆ lesvars upds
proof (induct upds)
  case Nil
  then show ?case by simp
next
  case (Cons a upds)
  obtain y e where a: a=(y,e) by fastforce
  from Cons show ?case unfolding a apply (simp)
    apply(rule subset_trans[OF support_and])
    apply(rule Un_least)
    subgoal apply(rule subset_trans[OF support_eq])
      using supportE_twicepreT subset_trans  supportE_single2 by simp
    subgoal by auto
    done
qed


lemma preListpreSet: preSet (set xs) C l s ⟹ preList xs C l s
proof (induct xs)
  case Nil
  then show ?case by simp
next
```

case (*Cons a xs*)
obtain *y e* **where** *a*: *a*=(*y*,*e*) **by** *fastforce*
from *Cons* **show** *?case* **unfolding** *a* **by** (*simp*)
qed

lemma *preSetpreList*: *preList xs C l s* $\Longrightarrow$ *preSet* (*set xs*) *C l s*
proof (*induct xs*)
case (*Cons a xs*)
obtain *y e* **where** *a*: *a*=(*y*,*e*) **by** *fastforce*
from *Cons* **show** *?case* **unfolding** *a*
by(*simp*)
qed *simp*

lemma *preSetpreList_eq*: *preList xs C l s* = *preSet* (*set xs*) *C l s*
proof (*induct xs*)
case (*Cons a xs*)
obtain *y e* **where** *a*: *a*=(*y*,*e*) **by** *fastforce*
from *Cons* **show** *?case* **unfolding** *a*
by(*simp*)
qed *simp*

fun *postList* **where**
postList [] *l s* = *True*
| *postList* ((*x*,*e*)#*xs*) *l s* = (*l x* = *e s* ∧ *postList xs l s*)

lemma *support_postList*: *support* (*postList xs*) ⊆ *lesvars xs*
proof (*induct xs*)
case (*Cons a xs*)
obtain *y e* **where** *a*: *a*=(*y*,*e*) **by** *fastforce*
from *Cons* **show** *?case* **unfolding** *a*
apply(*simp*) **apply**(*rule subset_trans*[*OF support_and*])
apply(*rule Un_least*)
subgoal apply(*rule subset_trans*[*OF support_eq*])
using *supportE_twicepreT subset_trans supportE_single2* **by** *simp*
subgoal by(*auto*)
done
qed *simp*

lemma *postpreList_inv*: **assumes** *S s* = *S* (*postQ C s*)
shows *postList* (*map* (λ(*x*, *e*). (*x*, λ*s*. *e* (*S s*))) *upds*) *l s* = *preList* (*map*

$(\lambda(x,\ e).\ (x,\ \lambda s.\ e\ (S\ s)))$ *upds*) *C l s*
proof (*induct upds*)
  case (*Cons a upds*)
  obtain *y e* **where** *axe*: $a = (y,e)$ **by** *fastforce*

  from *Cons* **show** *?case* **unfolding** *axe* **apply**(*simp*)
    apply(*simp only*: *TQ*) **using** *assms* **by** *auto*
qed *simp*

lemma *postList_preList*: *postList* (*map* $(\lambda(x,\ e).\ (x,\ preT\ C\ e))$ *upds*) *l s*
= *preList upds C l s*
proof (*induct upds*)
  case (*Cons a xs*)
  obtain *y e* **where** *a*: $a=(y,e)$ **by** *fastforce*
  from *Cons* **show** *?case* **unfolding** *a*
   by(*simp*)
qed *simp*

lemma *postSetpostList*: *postList xs l s* $\Longrightarrow$ *postSet* (*set xs*) *l s*
proof (*induct xs*)
  case (*Cons a xs*)
  obtain *y e* **where** *a*: $a=(y,e)$ **by** *fastforce*
  from *Cons* **show** *?case* **unfolding** *a*
   by(*simp*)
qed *simp*

lemma *postListpostSet*: *postSet* (*set xs*) *l s* $\Longrightarrow$ *postList xs l s*
proof (*induct xs*)
  case (*Cons a xs*)
  obtain *y e* **where** *a*: $a=(y,e)$ **by** *fastforce*
  from *Cons* **show** *?case* **unfolding** *a*
   by(*simp*)
qed *simp*

lemma *ListAskip*: *preList xs Askip l s* = *postList xs l s*
  apply(*induct xs*)
  apply(*simp*) **by** *force*

lemma *SetAskip*: *preSet U Askip l s* = *postSet U l s*
by *simp*

lemma *ListAassign*: *preList upds* (*Aassign x1 x2*) *l s* = *postList upds l* (*s*[*x2*/*x1*])
 apply(*induct upds*)
  apply(*simp*) **by** *force*

lemma *SetAassign*: *preSet U* (*Aassign x1 x2*) *l s* = *postSet U l* (*s*[*x2*/*x1*])
by *simp*

lemma *ListAconseq*: *preList upds* (*Aconseq x1 x2 x3 C*) *l s* = *preList upds C l s*
 apply(*induct upds*)
  apply(*simp*) **by** *force*

lemma *SetAconseq*: *preSet U* (*Aconseq x1 x2 x3 C*) *l s* = *preSet U C l s*
by *simp*

lemma *ListAif1*: *bval b s* $\Longrightarrow$ *preList upds* (*IF b THEN C1 ELSE C2*) *l s* = *preList upds C1 l s*
 apply(*induct upds*)
  apply(*simp*) **by** *force*
lemma *SetAif1*: *bval b s* $\Longrightarrow$ *preSet upds* (*IF b THEN C1 ELSE C2*) *l s* = *preSet upds C1 l s*
 apply(*simp*) **done**
lemma *ListAif2*: ~ *bval b s* $\Longrightarrow$ *preList upds* (*IF b THEN C1 ELSE C2*) *l s* = *preList upds C2 l s*
 apply(*induct upds*)
  apply(*simp*) **by** *force*

lemma *SetAif2*: ~ *bval b s* $\Longrightarrow$ *preSet upds* (*IF b THEN C1 ELSE C2*) *l s* = *preSet upds C2 l s*
 apply(*simp*) **done**

lemma *vc_sound*: *vc C Q* $\Longrightarrow$ *finite* (*support Q*) $\Longrightarrow$ *finite* (*varacom C*)
 $\Longrightarrow$ *fst* ‘ (*set upds*) $\cap$ *varacom C* = {} $\Longrightarrow$ *distinct* (*map fst upds*)
 $\Longrightarrow$ $\vdash_1$ {%*l s*. *pre C Q l s* $\wedge$ *preList upds C l s*} *strip C* { *time C* $\Downarrow$ %*l s*. *Q l s* $\wedge$ *postList upds l s*}
 $\wedge$ ($\forall$ *l s*. *pre C Q l s* $\longrightarrow$ *Q l* (*postQ C s*))
proof(*induction C arbitrary*: *Q upds*)
 case (*Askip Q upds*)
 then show *?case*

apply(*auto*)
apply(*rule weaken_post*[**where** *Q*=%*l s. Q l s* ∧ *preList upds Askip l s*])
apply(*simp add*: *Skip*) **using** *ListAskip*
by *fast*
next
case (*Aassign x1 x2 Q upds*)
then show *?case* **apply**(*safe*) **apply**(*auto simp add*: *Assign*)[*1*]
apply(*rule weaken_post*[**where** *Q*=%*l s. Q l s* ∧ *postList upds l s*])
apply(*simp only*: *ListAassign*)
apply(*rule Assign*) **apply** *simp*
apply(*simp only*: *postQ.simps pre.simps*) **done**
next
case (*Aif b C1 C2 Q upds*)
then show *?case* **apply**(*auto simp add*: *Assign*)
apply(*rule If2*[**where** *e*=*λa. if bval b a then time C1 a else time C2 a*])
subgoal
apply(*simp cong*: *rev_conj_cong*)
apply(*rule ub_cost*[**where** *e′*=*time C1*])
apply(*simp*) **apply**(*auto*)[*1*]
apply(*rule strengthen_pre*[**where** *P*=%*l s. pre C1 Q l s* ∧ *preList upds C1 l s*])
using *ListAif1*
apply *fast*
apply(*rule Aif*(*1*)[*THEN conjunct1*])
apply(*auto*)
done
subgoal
apply(*simp cong*: *rev_conj_cong*)
apply(*rule ub_cost*[**where** *e′*=*time C2*])
apply(*simp*) **apply**(*auto*)[*1*]
apply(*rule strengthen_pre*[**where** *P*=%*l s. pre C2 Q l s* ∧ *preList upds C2 l s*])
using *ListAif2*
apply *fast*
apply(*rule Aif*(*2*)[*THEN conjunct1*])
apply(*auto*)
done
apply *auto* **apply** *fast*+ **done**
next
case (*Aconseq P′ Qannot eannot C Q upds*)
then obtain *k* **where** *k*: *k*>*0* **and** *ih1*: *vc C Qannot*
and *ih1′*: (∀ *l s. P′ l s* ⟶ *time C s* ≤ *k* ∗ *eannot s* ∧ (∀ *t.* ∃ *l′. pre C*

$Qannot\ l'\ s \wedge (Qannot\ l'\ t \longrightarrow Q\ l\ t)))$
by *auto*

have $ih2'$: $\forall\, l\ s.\ pre\ C\ Qannot\ l\ s \longrightarrow Qannot\ l\ (postQ\ C\ s)$
apply(*rule Aconseq*(*1*)[*THEN conjunct2*]) **using** *Aconseq*(*2*−*6*) **by** *auto*

have $G1$: $\vdash_1 \{\lambda l\ s.\ P'\ l\ s \wedge preList\ upds\ (\{P'/Qannot/eannot\}\ CONSEQ\ C)\ l\ s\}\ strip\ C$
$\{\ eannot \Downarrow \lambda l\ s.\ Q\ l\ s \wedge postList\ upds\ l\ s\}$
proof (*rule conseq*[*rotated*])
show $\vdash_1 \{\lambda l\ s.\ pre\ C\ Qannot\ l\ s \wedge preList\ upds\ C\ l\ s\}\ strip\ C\ \{\ time\ C \Downarrow \lambda l\ s.\ Qannot\ l\ s \wedge postList\ upds\ l\ s\}$
apply(*rule Aconseq*(*1*)[*THEN conjunct1*])
using *Aconseq*(*2*−*6*) **by** *auto*
next
show $\exists\, k{>}0.\ \forall\, l\ s.\ P'\ l\ s \wedge preList\ upds\ (\{P'/Qannot/eannot\}\ CONSEQ\ C)\ l\ s \longrightarrow$
$time\ C\ s \leq k * eannot\ s \wedge$
$(\forall\, t.\ \exists\, l'.\ (pre\ C\ Qannot\ l'\ s \wedge preList\ upds\ C\ l'\ s) \wedge$
$(Qannot\ l'\ t \wedge postList\ upds\ l'\ t \longrightarrow Q\ l\ t \wedge postList\ upds\ l\ t))$
proof(*rule exI*[**where** *x*=*k*], *safe*)
fix *l s*
assume P': $P'\ l\ s$ **and** *prelist*: $preList\ upds\ (\{P'/Qannot/eannot\}\ CONSEQ\ C)\ l\ s$
then show $time\ C\ s \leq k * eannot\ s$ **using** $ih1'$ **by** *simp*

fix *t*
— we now have to construct a logical environment, that both * satisfies the annotated postcondition Qannot (we obtain it from the first IH) * lets the updates come true (we have to show that resetting these logical variables does not interfere with the other variables)

from $ih1'\ P'$ **have** *satQan*:$(\exists\, l'.\ pre\ C\ Qannot\ l'\ s \wedge (Qannot\ l'\ t \longrightarrow Q\ l\ t))$ **by** *simp*
then obtain l' **where** i': $pre\ C\ Qannot\ l'\ s$ **and** ii': $(Qannot\ l'\ t \longrightarrow Q\ l\ t)$ **by** *blast*

let $?upds' = (map\ (\%(x,e).\ (x,preT\ C\ e\ s))\ upds)$
let $?l'' = (ListUpdateE\ l'\ ?upds')$

{
fix *l s n x*

assume $x \in fst$ ‘ $(set\ upds)$
then have $x \notin support\ (pre\ C\ Qannot)$ **using** $Aconseq(5)\ support\_pre$ **by** *auto*
from $assn2\_lupd[OF\ this]$ **have** $pre\ C\ Qannot\ (l(x := n)) = pre\ C\ Qannot\ l$ **.**
} **note** *U2=this*
{
fix $l\ s\ n\ x$
assume $x \in fst$ ‘ $(set\ upds)$
then have $x \notin support\ Qannot$ **using** $Aconseq(5)$ **by** *auto*
from $assn2\_lupd[OF\ this]$ **have** $Qannot\ (l(x := n)) = Qannot\ l$ **.**
} **note** *K2=this*

have $pre\ C\ Qannot\ ?l'' = pre\ C\ Qannot\ l'$
apply(*rule allg_E*[**where** *?upds=set upds*]) **apply**(*rule U2*) **by** *force+*
with i' **have** i'': $pre\ C\ Qannot\ ?l''\ s$ **by** *simp*

have $Qannot\ ?l'' = Qannot\ l'$
apply(*rule allg_E*[**where** *?upds=set upds*]) **apply**(*rule K2*) **by** *force+*
then have K: $(\%l'\ s.\ Qannot\ l'\ t \longrightarrow Q\ l\ t)\ ?l''\ s = (\%l'\ s.\ Qannot\ l'\ t \longrightarrow Q\ l\ t)\ l'\ s$
by *simp*
with ii' **have** ii'': $(Qannot\ ?l''\ t \longrightarrow Q\ l\ t)$ **by** *simp*

have *xs_upds*: $map\ fst\ ?upds' = map\ fst\ upds$
by *auto*
have *resets*: $\bigwedge x.\ x \in set\ ?upds' \Longrightarrow ListUpdateE\ l'\ ?upds'\ (fst\ x) = snd\ x$ **apply**(*rule ListUpdateE_updates*)
apply(*simp only*: *xs_upds*) **using** $Aconseq(6)$ **apply** *simp*
apply(*simp*) **done**

have A: $preList\ upds\ C\ ?l''\ s$
proof (*rule preListpreSet,safe,goal_cases*)
case $(1\ x\ e)$
then have $(x, preT\ C\ e\ s) \in set\ ?upds'$
by *fastforce*
from *resets*[*OF this*, *simplified*]
show *?case* **.**
qed

have B: $Qannot\ ?l''\ t \Longrightarrow postList\ upds\ ?l''\ t \Longrightarrow postList\ upds\ l\ t$
proof (*rule postListpostSet*, *safe*, *goal_cases*)

case (*1 x e*)
from *postSetpostList*[*OF 1*(*2*)] **have** *g*: *postSet* (*set upds*) *?l″ t* **.**
with *1*(*3*) **have** *A*: *?l″ x* = *e t*
by *fast*
from *1*(*3*) *resets*[*of* (*x*,*preT C e s*)] **have** *B*: *?l″ x* = *snd* (*x*, *preT C e s*)
by *fastforce*
from *A B* **have** *X*: *e t* = *preT C e s* **by** *fastforce*
from *preSetpreList*[*OF prelist*] **have** *preSet* (*set upds*) ({*P′*/*Qannot*/*eannot*} *CONSEQ C*) *l s* **.**
with *1*(*3*) **have** *Y*: *l x* = *preT C e s* **apply**(*simp*) **by** *fast*
from *X Y* **show** *?case* **by** *simp*
qed

show $\exists l'$. (*pre C Qannot l′ s* ∧ *preList upds C l′ s*) ∧
(*Qannot l′ t* ∧ *postList upds l′ t* ⟶ *Q l t* ∧ *postList upds l t*)
apply(*rule exI*[**where** *x*=*?l″*], *safe*)
using *i″ A ii″ B* **by** *auto*
qed *fact*
qed

have *G2*: ⋀*l s*. *P′ l s* ⟹ *Q l* (*postQ C s*)
proof −
fix *l s*
assume *P′ l s*
with *ih1′ ih2′* **show** *Q l* (*postQ C s*) **by** *blast*
qed

show *?case* **using** *G1 G2* **by** *auto*
next
case (*Aseq C1 C2 Q upds*)

let *?P* = (λ*l s*. *pre* (*C1*;; *C2*) *Q l s* ∧ *preList upds* (*C1*;;*C2*) *l s*)
let *?P′* = *support Q* ∪ *varacom C1* ∪ *varacom C2* ∪ *lesvars upds*

have *finite_varacom*: *finite* (*varacom* (*C1*;; *C2*)) **by** *fact*
have *sup_L*: *support* (*preList upds* (*C1*;;*C2*)) ⊆ *lesvars upds*
apply(*rule support_preList*) **done**

— choose a fresh logical variable ?y in order to pull through the cost of the second command
let *?y* = *SOME x*. *x* ∉ *?P′*
have *fP′*: *finite* (*?P′*) **using** *finite_varacom Aseq*(*4*,*5*) **apply** *simp* **done**

from *fP'* **have** $\exists x.\ x \notin$ *?P'* **using** *infinite_UNIV_listI*
using *ex_new_if_finite* **by** *metis*
hence *ynP'*: *?y* $\notin$ *?P'* **by** (*rule someI_ex*)
hence *ysupC1*: *?y* $\notin$ *varacom C1* **using** *support_pre* **by** *auto*
have *sup_B*: *support ?P* $\subseteq$ *?P'*
apply(*rule subset_trans*[*OF support_and*]) **apply** *simp* **using** *support_pre sup_L* **by** *blast*

— we show the first goal: we can deduce the desired Hoare Triple
have *C1*: $\vdash_1$ {λ*l s. pre* (*C1*;; *C2*) *Q l s* $\wedge$ *preList upds* (*C1*;; *C2*) *l s*} *strip C1*;; *strip C2*
{ *time* (*C1*;; *C2*) $\Downarrow$ λ*l s. Q l s* $\wedge$ *postList upds l s*}
proof (*rule Seq*[*rotated*])
— start from the back: we can simply use the IH for C2, and solve the side conditions automatically
show $\vdash_1$ {(%*l s. pre C2 Q l s* $\wedge$ *preList upds C2 l s*)} *strip C2* { *time C2* $\Downarrow$ (%*l s. Q l s* $\wedge$ *postList upds l s*)}
apply(*rule Aseq*(*2*)[*THEN conjunct1*])
using *Aseq*(*3*$-$*7*) **by** *auto*
next
— prepare the new updates: pull them through C2 and save the new execution time of C2 in ?y
let *?upds* = *map* (λ*a. case a of* (*x*,*e*) $\Rightarrow$ (*x*, *preT C2 e*)) *upds*
let *?upds'* = (*?y*,*time C2*)#*?upds*

have *dst_upds'*: *distinct* (*map fst ?upds'*)
using *ynP' Aseq*(*7*) **apply** *simp* **apply** *safe*
using *image_iff* **apply** *fastforce* **by** (*simp add*: *case_prod_beta' distinct_conv_nth*)

— now use the first induction hypothesis (specialised with the augmented upds list, and the weakest precondition of Q through C as post condition)
have *IH1s*: $\vdash_1$ {λ*l s. pre C1* (*pre C2 Q*) *l s* $\wedge$ *preList ?upds' C1 l s*} *strip C1*
{ *time C1* $\Downarrow$ λ*l s. pre C2 Q l s* $\wedge$ *postList ?upds' l s*}
apply(*rule Aseq*(*1*)[*THEN conjunct1*])
using *Aseq*(*3*$-$*7*) *ysupC1 dst_upds'* **by** *auto*

— glue it together with a consequence rule, side conditions are automatic
show $\vdash_1$ {λ*l s.* (*pre* (*C1*;; *C2*) *Q l s* $\wedge$ *preList upds* (*C1*;; *C2*) *l s*) $\wedge$ *l ?y* = *preT C1* (*time C2*) *s*} *strip C1*
{ *time C1* $\Downarrow$ λ*l s.* (λ*l s. pre C2 Q l s* $\wedge$ *preList upds C2 l s*) *l s* $\wedge$ *time C2 s* $\leq$ *l ?y*}
apply(*rule conseq_old*[*OF _ IH1s*])

by (*auto simp*: *preList_Seq postList_preList*)
next
— solve some side conditions showing that, ?y is indeed fresh
show *?y* $\notin$ *support ?P*
using *sup_B ynP'* **by** *auto*
have *F*: *support* (*preList upds C2*) $\subseteq$ *lesvars upds*
apply(*rule support_preList*) **done**
have *support* (λl *s. pre C2 Q l s* $\wedge$ *preList upds C2 l s*) $\subseteq$ *?P'*
apply(*rule subset_trans*[*OF support_and*]) **using** *F support_pre* **by** *blast*
with *ynP'*
show *?y* $\notin$ *support* (λl *s. pre C2 Q l s* $\wedge$ *preList upds C2 l s*) **by** *blast*
qed *simp*

— we show the second goal: weakest precondition implies, that Q holds after the execution of C1 and C2
have *C2*: $\bigwedge l$ *s. pre* (*C1*;; *C2*) *Q l s* $\Longrightarrow$ *Q l* (*postQ* (*C1*;; *C2*) *s*)
proof −
fix *l s*
assume *p*: *pre* (*C1*;; *C2*) *Q l s*
have *A*: $\forall$ *l s. pre C1* (*pre C2 Q*) *l s* $\longrightarrow$ *pre C2 Q l* (*postQ C1 s*)
apply(*rule Aseq*(*1*)[**where** *upds*=[], *THEN conjunct2*])
using *Aseq* **by** *auto*
have *B*: ($\forall$ *l s. pre C2 Q l s* $\longrightarrow$ *Q l* (*postQ C2 s*))
apply(*rule Aseq*(*2*)[**where** *upds*=[], *THEN conjunct2*])
using *Aseq* **by** *auto*
from *p A B* **show** *Q l* (*postQ* (*C1*;; *C2*) *s*) **by** *simp*
qed

show *?case* **using** *C1 C2* **by** *simp*
next
case (*Awhile A b C Q upds*)

— Let us first see, what we got from the induction hypothesis:
obtain *I S E* **where** [*simp*]: *A* = (*I*,(*S*,(*E*))) **using** *prod_cases3* **by** *blast*
with ‹*vc* (*Awhile A b C*) *Q*› **have** *vc* (*Awhile* (*I*,*S*,*E*) *b C*) *Q* **by** *blast*
then have *vc*: *vc C I* **and** *pre2*: $\bigwedge l$ *s. I l s* $\Longrightarrow$ $\neg$ *bval b s* $\Longrightarrow$ *Q l s* $\wedge$ *1* $\leq$ *E s* $\wedge$ *S s* = *s*
and *IQ2*: $\bigwedge l$ *s. I l s* $\Longrightarrow$ *bval b s* $\Longrightarrow$
pre C I l s
$\wedge$ *1* + *preT C E s* + *time C s* $\leq$ *E s* $\wedge$ *S s* = *S* (*postQ C s*) **by** *auto*

— the logical variable x represents the number of loop unfoldings

from *IQ2* **have** *IQ_in*: $\bigwedge l\ s.\ I\ l\ s \Longrightarrow\ \ bval\ b\ s \Longrightarrow S\ s = S\ (postQ\ C\ s)$ **by** *auto*

have *inv_impl*: $\bigwedge l\ s.\ \ I\ l\ s \Longrightarrow\ \ bval\ b\ s \Longrightarrow\ \ pre\ C\ I\ \ l\ s$ **using** *IQ2* **by** *auto*

have *yC*: $lesvars\ upds \cap varacom\ C = \{\}$ **using** *Awhile*(*5*) **by** *auto*

let *?upds* = *map* (%(*x*,*e*). (*x*, %*s*. *e* (*S s*))) *upds*
let *?INV* = %*l s*. $I\ l\ s \wedge postList$ *?upds l s*

have $lesvars\ upds \cap support\ I = \{\}$ **using** *Awhile*(*5*) **by** *auto*

— we need a fresh variable ?z to remember the time bound of the tail of the loop
let *?P*=*lesvars upds* $\cup$ *varacom* ({*A*} *WHILE b DO C*)
let *?z*=*SOME z*::*lvname*. *z* $\notin$ *?P*
have *finite ?P* **using** *Awhile* **by** *auto*
hence $\exists$ *z*. *z*$\notin$*?P* **using** *infinite_UNIV_listI*
  using *ex_new_if_finite* **by** *metis*
hence *znP*: *?z* $\notin$ *?P* **by** (*rule someI_ex*)
from *znP* **have** *zny*: *?z* $\notin$ *lesvars upds*
  and *zI*: *?z* $\notin$ *support I*
  and *blb*: *?z* $\notin$ *varacom C* **by** (*simp_all*)

from *Awhile*(*4*,*6*) **have** *23*: *finite* (*varacom C*)
  and *26*: *finite* (*support I*) **by** *auto*

have $\forall\, l\ s.\ \ pre\ C\ I\ \ l\ s \longrightarrow I\ l\ (postQ\ C\ s)$
  apply(*rule Awhile*(*1*)[*THEN conjunct2*]) **by**(*fact*)+
hence *step*: $\bigwedge l\ s.\ pre\ C\ I\ l\ s \Longrightarrow I\ l\ (postQ\ C\ s)$ **by** *simp*

— we adapt the updates, by pulling them through the loop body and remembering the time bound of the tail of the loop
let *?upds* = *map* (λ(*x*, *e*). (*x*, λ*s*. *e* (*S s*))) *upds*
have *fua*: *lesvars ?upds* = *lesvars upds*
  by *force*
let *?upds'* = (*?z*,*E*) # *?upds*

have g: $\bigwedge e.\ e \circ S = (\%s.\ e\ (S\ s))$ **by** *auto*

— show that the Hoare Rule is derivable
have $G1$: $\vdash_1 \{\lambda l\ s.\ I\ l\ s \wedge preList\ upds\ (\{(I,\ S,\ E)\}\ WHILE\ b\ DO\ C)\ l\ s\}\ WHILE\ b\ DO\ strip\ C$
$\{\ E \Downarrow \lambda l\ s.\ Q\ l\ s \wedge postList\ upds\ l\ s\}$
proof(*rule conseq_old*)
show $\vdash_1 \{\lambda l\ s.\ I\ l\ s \wedge postList\ ?upds\ l\ s\}\ WHILE\ b\ DO\ strip\ C$
$\{\ E \Downarrow \lambda l\ s.\ (I\ l\ s \wedge postList\ ?upds\ l\ s) \wedge \neg bval\ b\ s\ \}$
— We use the While Rule and then have to show, that ...
proof(*rule While*, *goal_cases*)
— A) the loop body preserves the loop invariant
have $lesvars\ ?upds' \cap varacom\ C = \{\}$
using *yC blb* **by**(*auto*)

have z: $(fst \circ (\lambda(x,\ e).\ (x,\ \lambda s.\ e\ (S\ s)))) = fst$ **by** *auto*
have $distinct\ (map\ fst\ ?upds')$
using *Awhile*(*6*) *zny* **by** (*auto simp add*: *z*)

— for showing preservation of the invariant, use the consequence rule ...
show $\vdash_1 \{\lambda l\ s.\ (I\ l\ s \wedge postList\ ?upds\ l\ s) \wedge bval\ b\ s \wedge preT\ C\ E\ s = l\ ?z\}$
$strip\ C\ \{\ time\ C \Downarrow \lambda l\ s.\ (I\ l\ s \wedge postList\ ?upds\ l\ s) \wedge E\ s \leq l\ ?z\}$
proof (*rule conseq_old*)
— ... and employ the induction hypothesis, ...
show $\vdash_1 \{\lambda l\ s.\ pre\ C\ I\ l\ s \wedge preList\ ?upds'\ C\ l\ s\}\ strip\ C$
$\{\ time\ C \Downarrow \lambda l\ s.\ I\ l\ s \wedge postList\ ?upds'\ l\ s\}$
apply(*rule Awhile.IH*[*THEN conjunct1*]) **by** *fact+*
next
— finally we have to prove the side condition.
show $\exists k>0.\ \forall l\ s.\ (I\ l\ s \wedge postList\ ?upds\ l\ s) \wedge bval\ b\ s \wedge preT\ C\ E\ s = l\ ?z$
$\longrightarrow (pre\ C\ I\ l\ s \wedge preList\ ?upds'\ C\ l\ s) \wedge time\ C\ s \leq k * time\ C\ s$
apply(*rule exI*[**where** x=*1*]) **apply**(*simp*)
proof (*safe*, *goal_cases*)
case (*2 l s*)
note *upds_invariant=postpreList_inv*[*OF IQ_in*[*OF 2*(*1*)]]
from *2 upds_invariant* **show** *?case* **by** *auto*
next
case (*1 l s*) **then show** *?case* **using** *inv_impl* **by** *auto*
qed
qed *auto*

next

— B) the invariant with number of loop unfoldings greater than 0 implies true loop guard and running time is correctly bounded

show $\forall\, l\ s.\ bval\ b\ s\ \wedge I\ l\ s \wedge postList\ ?upds\ l\ s \longrightarrow 1 + preT\ C\ E\ s + time\ C\ s \leq E\ s$

proof (*clarify*, *goal_cases*)

case (*1 l s*)

show *?case* **using** *IQ2 1*(*1*,*2*) **by** *auto*

qed

next

— C) the invariant with number of loop unfoldings equal to 0 implies false loop guard and running time is correctly bounded

show $\forall\, l\ s.\ \neg\ bval\ b\ s \wedge I\ l\ s \wedge postList\ ?upds\ l\ s \longrightarrow\ 1 \leq E\ s$

proof (*clarify*, *goal_cases*)

case (*1 l s*)

then show *?case*

using *pre2 1*(*2*) **by** *auto*

qed

next

— D) ?z is indeed a fresh variable

have *pff*: $?z \notin lesvars\ ?upds$ **apply**(*simp only*: *fua*) **by** *fact*

have $support\ (\lambda l\ s.\ I\ l\ s \wedge postList\ ?upds\ l\ s) \subseteq support\ I \cup support\ (postList\ ?upds)$

by(*rule support_and*)

also have $support\ (postList\ ?upds) \subseteq lesvars\ ?upds$

apply(*rule support_postList*) **done**

finally

have $support\ (\lambda l\ s.\ I\ l\ s \wedge postList\ ?upds\ l\ s) \subseteq support\ I \cup lesvars\ ?upds$

by *blast*

thus $?z \notin support\ (\lambda l\ s.\ I\ l\ s \wedge postList\ ?upds\ l\ s)$

apply(*rule contra_subsetD*)

using *zI pff* **by**(*simp*)

qed

next

show $\exists\, k{>}0.\ \forall\, l\ s.\ I\ l\ s\ \wedge preList\ upds\ (\{(I,\ S,\ E)\}\ WHILE\ b\ DO\ C)\ l\ s \longrightarrow$ $(I\ l\ s \wedge postList\ (map\ (\lambda(x,\ e).\ (x,\ \lambda s.\ e\ (S\ s)))\ upds)\ l\ s) \wedge E\ s \leq k * E\ s$

apply(*rule exI*[**where** *x=1*]) **apply**(*auto*) **apply**(*simp only*: *postList_preList*[*symmetric*]) **apply** (*auto*)

apply(*simp only*: *g*)

done

next

show $\forall\, l\ s.\ (I\ l\ s \wedge postList\ (map\ (\lambda(x,\ e).\ (x,\ \lambda s.\ e\ (S\ s)))\ upds)\ l\ s) \wedge \neg\ bval\ b\ s\ \longrightarrow Q\ l\ s \wedge postList\ upds\ l\ s$
using *pre2* **by**(*induct upds*, *auto*)
qed

have *G2*: $\bigwedge l\ s.\ pre\ (\{A\}\ WHILE\ b\ DO\ C)\ Q\ l\ s \Longrightarrow Q\ l\ (postQ\ (\{A\}\ WHILE\ b\ DO\ C)\ s)$
proof −
fix *l s*
assume $pre\ (\{A\}\ WHILE\ b\ DO\ C)\ Q\ l\ s$
then have *I*: $I\ l\ s$ **by** *simp*
{ fix *n*
have $E\ s = n \Longrightarrow I\ l\ s \Longrightarrow Q\ l\ (postQ\ (\{A\}\ WHILE\ b\ DO\ C)\ s)$
proof (*induct n arbitrary*: *s l rule*: *less_induct*)
case (*less n*)
then show *?case*
proof (*cases bval b s*)
case *True*
with *less IQ2* **have** $pre\ C\ I\ l\ s$ **and** *S*: $S\ s = S\ (postQ\ C\ s)$ **and** *t*: $1 + preT\ C\ E\ s + time\ C\ s \leq E\ s$ **by** *auto*
with *step* **have** I': $I\ l\ (postQ\ C\ s)$ **and** $1 + E\ (postQ\ C\ s) + time\ C\ s \leq E\ s$ **using** *TQ* **by** *auto*
with *less* **have** $E\ (postQ\ C\ s) < n$ **by** *auto*
with *less*(*1*) I' **have** $Q\ l\ (postQ\ (\{A\}\ WHILE\ b\ DO\ C)\ (postQ\ C\ s))$ **by** *auto*
with *step* **show** *?thesis* **using** *S* **by** *simp*
next
case *False*
with *pre2 less*(*3*) **have** $Q\ l\ s\ S\ s = s$ **by** *auto*
then show *?thesis* **by** *simp*
qed
qed
}
with *I* **show** $Q\ l\ (postQ\ (\{A\}\ WHILE\ b\ DO\ C)\ s)$ **by** *simp*
qed

show *?case* **using** *G1 G2* **by** *auto*
qed

corollary *vc_sound'*:
  assumes *vc C Q*
      finite (*support Q*) *finite* (*varacom C*)
      $\forall$ *l s*. *P l s* $\longrightarrow$ *pre C Q l s*
  shows $\vdash_1$ {*P*} *strip C* {*time C* $\Downarrow$ *Q*}
proof –
  show *?thesis*
    apply(*rule conseq_old*)
      prefer *2* **apply**(*rule vc_sound*[**where** *upds*=[], *OF assms*(*1*−*3*), *THEN conjunct1*])
    using *assms*(*4*) **apply** *auto*
    done
qed

lemma *preT_constant*: *preT C* (%_. *a*) = (%_. *a*)
  apply(*induct C*) **by** (*auto*)

corollary *vc_sound''*:
  ⟦ *vc C Q*; ($\exists$ *k*>*0*. $\forall$ *l s*. *P l s* $\longrightarrow$ *pre C Q l s* $\wedge$ *time C s* $\leq$ *k* $*$ *e s*);
  finite (*support Q*); *finite* (*varacom C*)⟧ $\Longrightarrow$ $\vdash_1$ {*P*} *strip C* {*e* $\Downarrow$ *Q*}
  apply(*rule ub_cost*[**where** *e'*=*time C*])
   apply(*auto*)
  apply(*rule vc_sound'*) **by** *auto*

4.6.4 Completeness:

lemma *vc_complete*:
  $\vdash_1$ {*P*} *c* { *e* $\Downarrow$ *Q*} $\Longrightarrow$ $\exists$ *C*. *strip C* = *c* $\wedge$ *vc C Q*
  $\wedge$ ($\forall$ *l s*. *P l s* $\longrightarrow$ *pre C Q l s* $\wedge$ *Q l* (*postQ C s*))
  $\wedge$ ($\exists$ *k*. $\forall$ *l s*. *P l s* $\longrightarrow$ *time C s* $\leq$ *k* $*$ *e s*)
  (**is** _ $\Longrightarrow$ $\exists$ *C*. *?G P c Q C e*)
proof (*induction rule*: *hoare1.induct*)
  case *Skip*
  show *?case* (**is** $\exists$ *C*. *?C C*)
  proof **show** *?C Askip* **by** *auto*
  qed
next
  case (*Assign P a x*)
  show *?case* (**is** $\exists$ *C*. *?C C*)
   proof **show** *?C*(*Aassign x a*) **apply** (*simp del*: *fun_upd_apply*) **apply**(*auto*) **done** **qed**
next
  case (*Seq P x e2' c1 e1 Q e2 c2 R e*)

```
  from Seq.IH(1)   obtain C1 where ?G (λl s. P l s ∧ l x = e2' s) c1
(λa b. Q a b ∧ e2 b ≤ a x) C1 e1  by blast
  then obtain k where ih1: strip C1 = c1
    vc C1 (λa b. Q a b ∧ e2 b ≤ a x)
    ⋀l s. P l s ⟹ l x = e2' s ⟹ pre C1 (λla sa. (Q la sa ∧ e2 sa ≤ la
x)) l s
    (∀ l s. P l s ∧ l x = e2' s ⟶  time C1 s ≤ k ∗ e1 s)
    ⋀l s.  P l s ⟹ l x = e2' s ⟹ Q l (postQ C1 s) ∧ e2 (postQ C1 s) ≤
l x
    apply auto done

  from Seq.IH(2)   obtain C2 where ih2: ?G Q c2 R C2 e2   by blast
  then obtain k2 where ih2: strip C2 = c2
    vc C2 R
    (⋀l s. Q l s ⟹ pre C2 R l s)
    (∀ l s. Q l s ⟶  time C2 s ≤ k2 ∗ e2 s)
    ⋀l s . Q l s ⟹ R l (postQ C2 s)  apply auto done

  show ?case (is ∃ C. ?C C)
  proof
    show ?C(Aseq (Aconseq P Q (time C1) C1) C2)
    proof (safe, goal_cases)
      case 1
      then show ?case apply(simp add: ih1(1) ih2(1)) done
    next
      case 2
      then show ?case apply(simp) apply(safe)
         subgoal apply(rule vc_mono) prefer 2 apply (rule ih1(2)) ap-
ply(auto) done
        subgoal apply(rule exI[where x=1]) apply safe
          subgoal by(auto)
          subgoal for l s t
            apply(rule exI[where x=l(x:= e2' s)])
            apply(safe)
              subgoal apply(rule pre_mono) prefer 2 apply (rule ih1(3))

                apply(subst assn2_lupd) using Seq(3) by auto
              subgoal apply(rule ih2(3)) using assn2_lupd[OF Seq(4)] by
auto
            done
          done
        subgoal by (rule ih2(2))
        done
    next
```

case (*3 l s*)
then show *?case* **apply**(*simp*) **done**
next

case (*4 l s*)
from *4* **have** *P* (*l*(*x*:=*e2′ s*)) *s* **using** *assn2_lupd*[*OF Seq*(*3*)] **by** *simp*
with *ih1*(*5*)[**where** *l*=*l*(*x*:=*e2′ s*)]
have *Q* (*l*(*x* := *e2′ s*)) (*postQ C1 s*) **by** *simp*
then have *Q l* (*postQ C1 s*) **using** *assn2_lupd*[*OF Seq*(*4*)] **by** *simp*
with *ih2*(*3*) **have** *Q l* (*postQ C1 s*) **by** *simp*
with *ih2*(*5*)
show *?case* **apply**(*auto*) **done**
next
case *5*
from *ih1*(*4*) **have**
gg: $\bigwedge l\ s.\ [\![P\ l\ s;\ e2'\ s = l\ x]\!] \Longrightarrow\ \ time\ C1\ s \leq k * e1\ s$ **by** *auto*

show *?case*
proof (*rule exI*[**where** *x*=(*max k k2*)], *safe*, *goal_cases*)
case (*1 l s*)
have *xnP*: $x \notin$ *support P* **by** *fact*
have *41*: *P* (*l*(*x* := *e2′ s*)) *s*
apply(*subst assn2_lupd*)
apply(*fact xnP*)
apply(*fact 5*) **done**

have *A*: *time C1 s* $\leq$ *k* $*$ *e1 s*
apply(*rule gg*[**where** *l*=*l*(*x*:=*e2′ s*)])
apply(*rule 41*)
apply(*simp*) **done**

have *B*: *preT C1* (*time C2*) *s* $\leq$ *k2* $*$ *e2′ s*
proof −
from *1* **have** *P* (*l*(*x* := *e2′ s*)) *s* **using** *assn2_lupd*[*OF xnP*] **by** *simp*

have *F*: *Q* (*l*(*x*:=*e2′ s*)) (*postQ C1 s*) $\wedge$ *e2* (*postQ C1 s*) $\leq$ (*l*(*x*:=*e2′ s*)) *x*
apply(*rule ih1*(*5*)[**where** *l*=*l*(*x*:=*e2′ s*) **and** *s*=*s*])
apply(*fact*)
apply(*simp*) **done**
then have *time C2* (*postQ C1 s*) $\leq$ *k2* $*$ *e2* (*postQ C1 s*) **using** *ih2*(*4*) **by** *auto*
with *F* **have** *time C2* (*postQ C1 s*) $\leq$ *k2* $*$ *e2′ s*

using *order_subst1* **by** *fastforce*
then show $preT\ C1\ (time\ C2)\ s \leq k2 * e2'\ s$ **using** *TQ* **by** *simp*

qed
have $time\ C1\ s + preT\ C1\ (time\ C2)\ s \leq k\ \ * e1\ s + k2 * e2'\ s$ **using** *A B* **by** *linarith*
also have $\ldots \leq (max\ k\ \ k2) * e1\ s + (max\ k\ \ k2) * e2'\ s$
using *nat_mult_max_left* **by** *auto*
also have $\ldots = (max\ k\ \ k2) * (e1\ s + e2'\ s)$ **by** *algebra*
also have $\ldots \leq (max\ k\ \ k2) * e\ s$ **using** *Seq*(5)[*OF 1*] **by** *auto*
finally
have $time\ C1\ s + preT\ C1\ (time\ C2)\ s \leq (max\ k\ \ k2) * e\ s$ **.**
then show *?case*
by *auto*
qed
qed
qed

next
case (*If P b c1 e1 Q c2*)
from *If.IH*(*1*) **obtain** *C1* **where** $?G\ (\lambda l\ s.\ P\ l\ s \wedge bval\ b\ s)\ c1\ Q\ C1\ e1$
by *blast*
then obtain *k1* **where** *ih1*: $strip\ C1 = c1 \wedge vc\ C1\ Q \wedge (\forall l\ s.\ P\ l\ s \wedge bval\ b\ s \longrightarrow pre\ C1\ Q\ l\ s \wedge Q\ l\ (postQ\ C1\ s)) \wedge (\forall l\ s.\ P\ l\ s \wedge bval\ b\ s \longrightarrow time\ C1\ s \leq k1 * e1\ s)$
by *blast*
from *If.IH*(*2*) **obtain** *C2* **where** $?G\ (\lambda l\ s.\ P\ l\ s \wedge \neg bval\ b\ s)\ c2\ Q\ C2\ e1$
by *blast*
then obtain *k2* **where** *ih2*: $strip\ C2 = c2 \wedge vc\ C2\ Q \wedge (\forall l\ s.\ P\ l\ s \wedge \neg bval\ b\ s \longrightarrow pre\ C2\ Q\ l\ s \wedge Q\ l\ (postQ\ C2\ s)) \wedge (\forall l\ s.\ P\ l\ s \wedge \neg bval\ b\ s \longrightarrow time\ C2\ s \leq k2 * e1\ s\ )$
by *blast*
define k' **where** $k' == max\ (k1{+}1)\ (k2{+}1)$
show *?case* (**is** $\exists C.\ ?C\ C$)
proof
show $?C(Aif\ b\ C1\ C2)$
apply(*safe*)
prefer *5*
apply(*rule exI*[**where** $x{=}k'$]) **apply**(*safe*)
subgoal for *l s* **apply**(*auto*)
proof(*goal_cases*)
case *1*
with *ih1* **have** $time\ C1\ s \leq k1 * e1\ s$ **by** *blast*

```
      then have Suc (time C1 s) ≤ 1 + k1 ∗ e1 s by auto
      also have ... ≤ k′ + k1 ∗ e1 s unfolding k′_def by(auto)
      also have ... ≤ k′ + k′ ∗ e1 s unfolding k′_def
        by (simp add: max_def)
      finally show ?case .
    next
      case 2
      with ih2 have time C2 s ≤ k2 ∗ e1 s by blast
      then have Suc (time C2 s) ≤ 1 + k2 ∗ e1 s by auto
      also have ... ≤ k′ + k2 ∗ e1 s unfolding k′_def by(auto)
      also have ... ≤ k′ + k′ ∗ e1 s unfolding k′_def
        by (simp add: max_def)
      finally show ?case .
    qed
    using ih1 ih2 apply(simp)
    using ih1 ih2 apply(auto)
    done
  qed
next
  case (While P b e′ y c e′′ e)
  have supportPre: support (λl s. P l s ∧ bval b s ∧ e′ s = l y) ⊆ support
P ∪ {y}
  using support_and support_single  by fast
  from  While.IH obtain C where
    ih: ?G (λl s. P l s ∧ bval b s ∧ e′ s = l y) c (λa b. P a b ∧ e b ≤ a y)
C e′′
    using supportPre by blast
  then obtain k where ih2: vc C (λa b. P a b ∧ e b ≤ a y)
    ⋀l s. ⟦ P l s ; bval b s ; e′ s = l y ⟧ ⟹ pre C (λla sa. (P la sa ∧ e sa
≤ la y)) l s
    ⋀l s. ⟦ P l s ; bval b s ; e′ s = l y ⟧ ⟹  time C s ≤ k ∗ e′′ s
    ⋀l s.⟦ P l s ; bval b s ; e′ s = l y⟧ ⟹ P l (postQ C s) ∧ e (postQ C s)
≤ l y
    by fast

  let ?S = postQs C b
  {
    fix l s n
     have e s = n ⟹ P l s ⟹ postQs_dom (C, b, s) ∧ P l (?S s) ∧ ~
bval b (?S s)
    proof (induct n arbitrary: l s rule: less_induct)
      case (less x)
      show ?case
      proof (cases bval b s)
```

case *True*
with *While*(*2*) *less*(*3*) **have** $1 + e'\ s + e''\ s \leq e\ s$ **by** *auto*
then have *e'e*: $e'\ s < e\ s$ **by** *simp*
have *P* (*l*(*y*:=*e' s*)) *s* **using** *less*(*3*) *assn2_lupd*[*OF While*(*4*)] **by** *simp*
from *ih2*(*4*)[*OF this*] *True* **have** *ee'*: $e\ (postQ\ C\ s) \leq e'\ s$ **and** *P'*: *P* (*l*(*y* := *e' s*)) (*postQ C s*) **by** *auto*
from *P'* **have** *P''*: *P l* (*postQ C s*) **using** *less*(*3*) *assn2_lupd*[*OF While*(*4*)] **by** *simp*
from *ee' e'e less*(*2*) **have** $e\ (postQ\ C\ s) < x$ **by** *auto*
from *less*(*1*)[*OF this _ P''*] **have** *d*: *postQs_dom* (*C*, *b*, *postQ C s*)
and *p*: *P l* (*postQs C b* (*postQ C s*))
and *b*: ¬ *bval b* (*postQs C b* (*postQ C s*)) **by** *auto*
have *d'*: *postQs_dom* (*C*, *b*, *s*)
by (*simp add*: *d postQs.domintros*)
have *p'*: *P l* (*postQs C b s*)
using *True d p postQs.domintros postQs.psimps* **by** *fastforce*
have *b'*: ¬ *bval b* (*postQs C b s*)
by (*metis b d postQs.domintros postQs.pelims*)

from *d' p' b'* **show** *?thesis* **by** *auto*
next
case *False*
then have *1*: *postQs_dom* (*C*, *b*, *s*)
using *postQs.domintros* **by** *blast*
then have *2*: *?S s* = *s* **using** *postQs.psimps False* **by** *force*
from *1 2 less*(*3*) *False* **show** *?thesis* **by** *simp*
qed
qed
}
then have *Pdom*: $\bigwedge l\ s.\ P\ l\ s \Longrightarrow postQs\_dom\ (C,\ b,\ s) \wedge P\ l\ (?S\ s) \wedge$ ~ *bval b* (*?S s*) **by** *simp*

have *S1*: $\bigwedge l\ s.\ P\ l\ s \Longrightarrow P\ l\ (?S\ s)$ **using** *Pdom* **by** *simp*
have *S2*: $\bigwedge l\ s.\ P\ l\ s \Longrightarrow$ ~ *bval b* (*?S s*) **using** *Pdom* **by** *simp*
have *S3*: $\bigwedge l\ s.\ P\ l\ s \Longrightarrow bval\ b\ s \Longrightarrow ?S\ s = ?S\ (postQ\ C\ s)$ **using** *postQs.psimps Pdom* **by** *simp*
have *S4*: $\bigwedge l\ s.\ P\ l\ s \Longrightarrow \neg\ bval\ b\ s \Longrightarrow ?S\ s = s$ **using** *postQs.psimps Pdom* **by** *simp*

let *?w* = {(*P*,*?S*,(%*s*. *max k 1* ∗ *e s*))} *WHILE b DO* (*Aconseq* (λ*l s*. *P l s* ∧ *bval b s*) (λ*la sa*. *P la sa* ∧ *e sa* ≤ *la y*) (*time C*) *C*)

show *?case* (**is** ∃ *C*. *?C C*)

```
  proof
    show ?C ?w
    proof (safe, goal_cases)
      case 1
      then show ?case using ih by(simp)
    next
      case 2
      then show ?case
      proof(simp, safe, goal_cases)
        case (1 l s)
        from 2 have z: P (l(y := e′ s)) s
          using 1 assn2_lupd[OF While(4)]  by metis
        from ih2(3)[where l=l(y := e′ s) and s=s]
        have A:  time C s ≤ k ∗ e′′ s  using 1 z by(simp)

        from ih2(4)[where l=l(y := e′ s) and s=s]
      have e (postQ C s) ≤ (l(y := e′ s)) y apply(simp) using 1 z by(simp)

        then have e (postQ C s) ≤ e′ s by simp

        with TQ have B: preT C e s ≤ e′ s by simp
        let ?eskal = (λs. max k (Suc 0) ∗ e s)
         have preT C (λs. max k (Suc 0) ∗ e s) s = max k (Suc 0) ∗ preT
C e s
          using preT_linear by simp
        with B have  B: preT C ?eskal s ≤ max k (Suc 0) ∗ e′ s by auto

       from  While.hyps(2) 1 have C: 1 + e′ s + e′′ s ≤ e s by auto
       have Suc (preT C ?eskal s + time C s) ≤ 1 + (max k 1) ∗ e′ s + k
∗ e′′ s
          using A B by linarith
        also have ... ≤ (max k 1) + (max k 1) ∗ e′ s + (max k 1) ∗ e′′ s
          using nat_mult_max_left by auto
        also have ... = (max k 1) ∗ (1 + e′ s + e′′ s)
          by algebra
        also have ... ≤ (max k 1) ∗ e s
          using C by (metis mult.assoc mult_le_mono2)
       finally have Suc (preT C ?eskal s + time C s) ≤ ((max k 1) ) ∗ e s
.
        thus ?case by auto
      next
        case (3 l s)
        with While.hyps(3) show ?case by auto
      next
```

```
      case 5
      then show ?case
        apply(rule vc_mono)
        prefer 2 apply(fact ih2(1)) by auto
    next
      case 6
      show ?case apply(rule exI[where x=1]) apply(safe)
        subgoal by simp
        subgoal for l s t apply(rule exI[where x=l(y:=e' s)])

        proof (safe)
          assume 8: P l s and b:  bval b s
            then have P (l(y := e' s)) s using assn2_lupd[OF While(4)]
by metis
           with b ih2(2)  show pre C (λla sa. P la sa ∧ e sa ≤ la y) (l(y
:= e' s)) s
            apply(auto) done
          fix t
          assume P (l(y := e' s)) t
          thus P l t using assn2_lupd[OF While(4)] by simp
        qed
        done
    qed (simp_all add: S4 S3)
  next
    case 6
    show ?case apply(rule exI[where x=k+1]) by auto
  qed (simp_all add: S1 S2)
 qed
next
 case (conseq P' e e' P Q Q' c)
 then obtain C k where C: strip C = c
   vc C Q
   (∀ l s  . P l s ⟶ pre C Q  l s )
   (∀ l s . P l s ⟶ Q l (postQ C s))
   (∀ l s. P l s ⟶  time C s ≤ k ∗ e s) by metis
 from conseq(1) obtain k2 where cons:  ∀ l s. P' l s ⟶ e s ≤ k2 ∗ e' s
∧ (∀ t. ∃ l'. P l' s ∧ (Q l' t ⟶ Q' l t)) by auto

 show ?case
   apply(rule exI[where x=Aconseq P' Q (time C) C])
   apply(safe)
   subgoal apply(simp) by(fact)
   subgoal apply(simp)
     apply(safe)
```

subgoal using $C(2)$
apply(*fast*) **done**
subgoal
apply(*rule exI*[**where** $x=k+1$])
apply *auto*
using $C(2)$ *cons* $C(3)$ **by** *blast*
done
subgoal apply(*rule pre_mono*)
prefer *2* **apply**(*simp*) **using** $C(3)$ *conseq*(*1*) **apply** *fast*
done
subgoal
apply(*simp*)
using $C(4)$ *conseq*(*1*,*3*) **apply** *blast* **done**
apply(*rule exI*[**where** $x=k*k2$]) **apply**(*safe*)
subgoal for *l s*
using $C(5)$ *cons* **apply**(*auto*)
proof(*goal_cases*)
case *1*
then have *absch*: $e\ s \le k2 * e'\ s$ *time* $C\ s \le k\ * e\ s$ **by** *blast+*
show *?case*
using *absch order_trans* **by** *fastforce*
qed
done
qed

end

4.7 The Variables in an Expression

theory *Vars* **imports** *Com*
begin

We need to collect the variables in both arithmetic and boolean expressions. For a change we do not introduce two functions, e.g. *avars* and *bvars*, but we overload the name *vars* via a *type class*, a device that originated with Haskell:

class *vars* =
fixes *vars* :: $'a \Rightarrow$ *vname set*

This defines a type class "vars" with a single function of (coincidentally) the same name. Then we define two separated instances of the class, one for *aexp* and one for *bexp*:

instantiation *aexp* :: *vars*
begin

fun *vars_aexp* :: *aexp* ⇒ *vname set* **where**
$vars\ (N\ n) = \{\}\ |$
$vars\ (V\ x) = \{x\}\ |$
$vars\ (Plus\ a_1\ a_2) = vars\ a_1 \cup vars\ a_2\ \ |$
$vars\ (Times\ a_1\ a_2) = vars\ a_1 \cup vars\ a_2\ \ |$
$vars\ (Div\ a_1\ a_2) = vars\ a_1 \cup vars\ a_2$

instance ..

end

value *vars* (*Plus* (*V* ''x'') (*V* ''y''))

instantiation *bexp* :: *vars*
begin

fun *vars_bexp* :: *bexp* ⇒ *vname set* **where**
$vars\ (Bc\ v) = \{\}\ |$
$vars\ (Not\ b) = vars\ b\ |$
$vars\ (And\ b_1\ b_2) = vars\ b_1 \cup vars\ b_2\ |$
$vars\ (Less\ a_1\ a_2) = vars\ a_1 \cup vars\ a_2$

instance ..

end

value *vars* (*Less* (*Plus* (*V* ''z'') (*V* ''y'')) (*V* ''x''))

abbreviation
  eq_on :: ('*a* ⇒ '*b*) ⇒ ('*a* ⇒ '*b*) ⇒ '*a set* ⇒ *bool*
 (‹(_ =/ _/ on _)› [*50*,*0*,*50*] *50*) **where**
$f = g\ on\ X == \forall\ x \in X.\ f\ x = g\ x$

lemma *aval_eq_if_eq_on_vars*[*simp*]:
  $s_1 = s_2\ on\ vars\ a \Longrightarrow aval\ a\ s_1 = aval\ a\ s_2$
apply(*induction a*)
apply *simp_all*
done

lemma *bval_eq_if_eq_on_vars*:
  $s_1 = s_2\ on\ vars\ b \Longrightarrow bval\ b\ s_1 = bval\ b\ s_2$
proof(*induction b*)
  case (*Less a1 a2*)

hence *aval a1* s_1 = *aval a1* s_2 **and** *aval a2* s_1 = *aval a2* s_2 **by** *simp_all*
thus *?case* **by** *simp*
qed *simp_all*

fun *lvars* :: *com* ⇒ *vname set* **where**
lvars SKIP = {} |
lvars (*x*::=*e*) = {*x*} |
lvars (*c1*;;*c2*) = *lvars c1* ∪ *lvars c2* |
lvars (*IF b THEN c1 ELSE c2*) = *lvars c1* ∪ *lvars c2* |
lvars (*WHILE b DO c*) = *lvars c*

fun *rvars* :: *com* ⇒ *vname set* **where**
rvars SKIP = {} |
rvars (*x*::=*e*) = *vars e* |
rvars (*c1*;;*c2*) = *rvars c1* ∪ *rvars c2* |
rvars (*IF b THEN c1 ELSE c2*) = *vars b* ∪ *rvars c1* ∪ *rvars c2* |
rvars (*WHILE b DO c*) = *vars b* ∪ *rvars c*

instantiation *com* :: *vars*
begin

definition *vars_com c* = *lvars c* ∪ *rvars c*

instance ..

end

lemma *vars_com_simps*[*simp*]:
vars SKIP = {}
vars (*x*::=*e*) = {*x*} ∪ *vars e*
vars (*c1*;;*c2*) = *vars c1* ∪ *vars c2*
vars (*IF b THEN c1 ELSE c2*) = *vars b* ∪ *vars c1* ∪ *vars c2*
vars (*WHILE b DO c*) = *vars b* ∪ *vars c*
by(*auto simp*: *vars_com_def*)

end
theory *Nielson_VCGi*
imports *Nielson_Hoare Vars*
begin

4.8 Optimized Verification Condition Generator

Annotated commands: commands where loops are annotated with invariants.

datatype *acom* =
  Askip (‹*SKIP*›) |
  Aassign vname aexp (‹(_ ::= _)› [*1000*, *61*] *61*) |
  Aseq acom acom (‹_;;/ _› [*60*, *61*] *60*) |
  Aif bexp acom acom (‹(*IF* _/ *THEN* _/ *ELSE* _)› [*0*, *0*, *61*] *61*) |
  Aconseq assn2∗(*vname set*) *assn2*∗(*vname set*) *tbd* ∗ (*vname set*) *acom*
  (‹({_′/_′/_}/ *CONSEQ* _)› [*0*, *0*, *0*, *61*] *61*)|
  Awhile (*assn2*∗(*vname set*))∗((*state*⇒*state*)∗(*tbd*∗((*vname set*∗(*vname* ⇒ *vname set*))))) *bexp acom* (‹({_}/ *WHILE* _/ *DO* _)› [*0*, *0*, *61*] *61*)

notation *com.SKIP* (‹*SKIP*›)

Strip annotations:

fun *strip* :: *acom* ⇒ *com* **where**
  strip SKIP = *SKIP* |
  strip (*x* ::= *a*) = (*x* ::= *a*) |
  strip (C_1;; C_2) = (*strip* C_1;; *strip* C_2) |
  strip (*IF b THEN* C_1 *ELSE* C_2) = (*IF b THEN strip* C_1 *ELSE strip* C_2)
|
  strip ({_/_/_} *CONSEQ C*) = *strip C* |
  strip ({_} *WHILE b DO C*) = (*WHILE b DO strip C*)

support of an expression

definition *supportE* :: ((*char list* ⇒ *nat*) ⇒ (*char list* ⇒ *int*) ⇒ *nat*) ⇒ *string set* **where**
  supportE P = {*x*. ∃ *l1 l2 s*. (∀ *y*. *y* ≠ *x* ⟶ *l1 y* = *l2 y*) ∧ *P l1 s* ≠ *P l2 s*}

lemma *expr_lupd*: *x* ∉ *supportE Q* ⟹ *Q* (*l*(*x*:=*n*)) = *Q l*
  by(*simp add*: *supportE_def fun_upd_other fun_eq_iff*)
    (*metis* (*no_types*, *lifting*) *fun_upd_def*)

fun *varacom* :: *acom* ⇒ *lvname set* **where**
  varacom (C_1;; C_2)= *varacom* C_1 ∪ *varacom* C_2
| *varacom* (*IF b THEN* C_1 *ELSE* C_2)= *varacom* C_1 ∪ *varacom* C_2
| *varacom* ({(*P*,_)/(*Qannot*,_)/_} *CONSEQ C*)= *support P* ∪ *varacom C* ∪ *support Qannot*
| *varacom* ({((*I*,_),(*S*,(*E*,*Es*)))} *WHILE b DO C*) = *support I* ∪ *varacom C*

| *varacom* _ = {}

fun *varnewacom* :: *acom* ⇒ *lvname set* **where**
varnewacom (C_1;; C_2)= *varnewacom* C_1 ∪ *varnewacom* C_2
| *varnewacom* (*IF b THEN* C_1 *ELSE* C_2)= *varnewacom* C_1 ∪ *varnewacom* C_2
| *varnewacom* ({_/_/_} *CONSEQ C*)= *varnewacom C*
| *varnewacom* ({(*I*,(*S*,(*E*,*Es*)))} *WHILE b DO C*) = *varnewacom C*
| *varnewacom* _ = {}

lemma *finite_varnewacom*: *finite* (*varnewacom C*)
by (*induct C*) (*auto*)

fun *wf* :: *acom* ⇒ *lvname set* ⇒ *bool* **where**
wf SKIP _ = *True* |
wf (*x* ::= *a*) _ = *True* |
wf (C_1;; C_2) *S* = (*wf* C_1 (*S* ∪ *varnewacom* C_2) ∧ *wf* C_2 *S*) |
wf (*IF b THEN* C_1 *ELSE* C_2) *S* = (*wf* C_1 *S* ∧ *wf* C_2 *S*) |
wf ({_/(*Qannot*,_)/_} *CONSEQ C*) *S* = (*finite* (*support Qannot*) ∧ *wf C S*) |
wf ({(_,(_,(_,*Es*)))} *WHILE b DO C*) *S* = (*wf C S*)

Weakest precondition from annotated commands:

fun *preT* :: *acom* ⇒ *tbd* ⇒ *tbd* **where**
preT SKIP e = *e* |
preT (*x* ::= *a*) *e* = (λ*s*. *e*(*s*(*x* := *aval a s*))) |
preT (C_1;; C_2) *e* = *preT* C_1 (*preT* C_2 *e*) |
preT ({_/_/_} *CONSEQ C*) *e* = *preT C e* |
preT (*IF b THEN* C_1 *ELSE* C_2) *e* =
(λ*s*. *if bval b s then preT* C_1 *e s else preT* C_2 *e s*) |
preT ({(_,(*S*,_))} *WHILE b DO C*) *e* = *e o S*

lemma *preT_constant*: *preT C* (%_. *a*) = (%_. *a*)
by(*induct C*, *auto*)

lemma *preT_linear*: *preT C* (%*s*. *k* ∗ *e s*) = (%*s*. *k* ∗ *preT C e s*)
by (*induct C arbitrary*: *e*, *auto*)

fun *postQ* :: *acom* ⇒ *state* ⇒ *state* **where**
postQ SKIP s = *s* |

$postQ\ (x ::= a)\ s = \ s(x := aval\ a\ s)\ |$
$postQ\ (C_1;;\ C_2)\ s = postQ\ C_2\ (postQ\ C_1\ s)\ |$
$postQ\ (\{\_/\_/\_\}\ CONSEQ\ C)\ s = postQ\ C\ s\ |$
$postQ\ (IF\ b\ THEN\ C_1\ ELSE\ C_2)\ s =$
$(if\ bval\ b\ s\ then\ postQ\ C_1\ s\ else\ postQ\ C_2\ s)\ |$
$postQ\ (\{(\_,(S,\_))\}\ WHILE\ b\ DO\ C)\ s = S\ s$

fun *fune* :: *acom* ⇒ *vname set* ⇒ *vname set* **where**
$fune\ SKIP\ LV = LV\ |$
$fune\ (x ::= a)\ LV = LV \cup vars\ a\ |$
$fune\ (C_1;;\ C_2)\ LV = fune\ C_1\ (fune\ C_2\ LV)\ |$
$fune\ (\{\_/\_/\_\}\ CONSEQ\ C)\ LV = fune\ C\ LV\ |$
$fune\ (IF\ b\ THEN\ C_1\ ELSE\ C_2)\ LV = vars\ b \cup fune\ C_1\ LV \cup fune\ C_2$
$LV\ |$
$fune\ (\{(\_,(S,(E,Es,SS)))\}\ WHILE\ b\ DO\ C)\ LV = (\bigcup x{\in}LV.\ SS\ x)$

lemma *fune_mono*: $A \subseteq B \Longrightarrow fune\ C\ A \subseteq fune\ C\ B$
proof(*induct C arbitrary*: *A B*)
case (*Awhile x1 x2 C*)
obtain *a b c d e f* **where** *a*: *x1* = (*a*,*b*,*c*,*d*,*e*) **using** *prod_cases5* **by** *blast*
from *Awhile* **show** *?case* **unfolding** *a* **by**(*auto*)
qed (*auto simp add*: *le_supI1 le_supI2*)

lemma *TQ*: $preT\ C\ e\ s = e\ (postQ\ C\ s)$
apply(*induct C arbitrary*: *e s*) **by** (*auto*)

function (*domintros*) *times* :: *state* ⇒ *bexp* ⇒ *acom* ⇒ *nat* **where**
$times\ s\ b\ C = (if\ bval\ b\ s\ then\ Suc\ (times\ (postQ\ C\ s)\ b\ C)\ else\ 0)$
apply(*auto*) **done**

lemma assumes *I*: *I z s* **and**
i: $\bigwedge s\ z.\ I\ (Suc\ z)\ s \Longrightarrow bval\ b\ s \wedge I\ z\ (postQ\ C\ s)$
and *ii*: $\bigwedge s.\ I\ 0\ s \Longrightarrow {\sim}\ bval\ b\ s$
shows *times_z*: $times\ s\ b\ C = z$
proof −
have $I\ z\ s \Longrightarrow times\_dom\ (s,\ b,\ C) \wedge times\ s\ b\ C = z$

proof(*induct z arbitrary*: *s*)
  case *0*
  have *A*: *times_dom* (*s*, *b*, *C*)
    apply(*rule times.domintros*)
    apply(*simp add*: *ii*[*OF 0*]) **done**
  have *B*: *times s b C* = *0*
    using *times.psimps*[*OF A*] **by**(*simp add*: *ii*[*OF 0*])

  show *?case* **using** *A B* **by** *simp*
next
  case (*Suc z*)
  from *i*[*OF Suc*(*2*)] **have** *bv*: *bval b s*
    and *g*: *I z* (*postQ C s*) **by** *simp_all*
  from *Suc*(*1*)[*OF g*] **have** *p1*: *times_dom* (*postQ C s*, *b*, *C*)
    and *p2*: *times* (*postQ C s*) *b C* = *z* **by** *simp_all*
  have *A*: *times_dom* (*s*, *b*, *C*)
    apply(*rule times.domintros*) **apply**(*rule p1*) **done**
  have *B*: *times s b C* = *Suc z*
    using *times.psimps*[*OF A*] *bv p2* **by** *simp*
  show *?case* **using** *A B* **by** *simp*
qed

then show *times s b C* = *z* **using** *I* **by** *simp*
qed

fun *postQz* :: *acom* ⇒ *state* ⇒ *nat* ⇒ *state* **where**
  postQz C s 0 = *s* |
  postQz C s (*Suc n*) = (*postQz C* (*postQ C s*) *n*)

fun *preTz* :: *acom* ⇒ *tbd* ⇒ *nat* ⇒ *tbd* **where**
  preTz C e 0 = *e* |
  preTz C e (*Suc n*) = *preT C* (*preTz C e n*)

lemma *TzQ*: *preTz C e n s* = *e* (*postQz C s n*)
  by (*induct n arbitrary*: *s*, *simp_all add*: *TQ*)

Weakest precondition from annotated commands:

fun *pre* :: *acom* ⇒ *assn2* ⇒ *assn2* **where**
  pre SKIP Q = *Q* |
  pre (*x* ::= *a*) *Q* = (λ*l s*. *Q l* (*s*(*x* := *aval a s*))) |
  pre (C_1;; C_2) *Q* = *pre* C_1 (*pre* C_2 *Q*) |
  pre ({(*P'*,*Ps*)/_/_} *CONSEQ C*) *Q* = *P'* |

```
  pre (IF b THEN C₁ ELSE C₂) Q =
  (λl s. if bval b s then pre C₁ Q l s else pre C₂ Q l s) |
  pre ({((I,Is),(S,(E,Es,SS)))} WHILE b DO C) Q = I


fun qdeps :: acom ⇒ vname set ⇒ vname set where
  qdeps SKIP LV  = LV |
  qdeps (x ::= a) LV = LV ∪ vars a |
  qdeps (C₁;; C₂) LV  = qdeps C₁ (qdeps C₂ LV) |
  qdeps ({(P',Ps)/_/_} CONSEQ C) _ = Ps |
  qdeps (IF b THEN C₁ ELSE C₂) LV = vars b ∪ qdeps C₁ LV ∪ qdeps
C₂ LV |
  qdeps ({((I,Is),(S,(E,x,Es)))} WHILE b DO C) _ = Is

lemma qdeps_mono: A ⊆ B ⟹ qdeps C A ⊆ qdeps C B
  by (induct C arbitrary: A B, auto simp: le_supI1 le_supI2)

lemma supportE_if: supportE (λl s. if b s then A l s else B l s)
  ⊆ supportE A ∪ supportE B
  unfolding supportE_def apply(auto)
  by metis+

lemma supportE_preT: supportE (%l. preT C (e l)) ⊆ supportE e
proof(induct C arbitrary: e)
  case (Aif b C1 C2 e)
  show ?case
    apply(simp)
    apply(rule subset_trans[OF supportE_if])
    using Aif by fast
next
  case (Awhile A y C e)
  obtain I S E x where A: A= (I,S,E,x) using prod_cases4 by blast
  show ?case using A apply(simp) unfolding supportE_def
    by blast
next
  case (Aseq)
  then show ?case by force
qed (simp_all add: supportE_def, blast)

lemma supportE_twicepreT: supportE (%l. preT C1 (preT C2 (e l))) ⊆
supportE e
  by (rule subset_trans[OF supportE_preT supportE_preT])
```

lemma *supportE_preTz*: *supportE* (%*l*. *preTz* *C* (*e* *l*) *n*) ⊆ *supportE* *e*
proof (*induct* *n*)
  case (*Suc* *n*)
  show *?case*
    apply(*simp*)
    apply(*rule* *subset_trans*[*OF* *supportE_preT*])
    by *fact*
qed *simp*

lemma *supportE_preTz_Un*:
  supportE (λ*l*. *preTz* *C* (*e* *l*) (*l* *x*)) ⊆ *insert* *x* (*UN* *n*. *supportE* (λ*l*. *preTz* *C* (*e* *l*) *n*))
  apply(*auto* *simp* *add*: *supportE_def* *subset_iff*)
  apply *metis*
  done

lemma *support_eq*: *support* (λ*l* *s*. *l* *x* = *E* *l* *s*) ⊆ *supportE* *E* ∪ {*x*}
  unfolding *support_def* *supportE_def*
  apply(*auto*)
   apply *blast*
  by *metis*

lemma *support_impl_in*: *G* *e* ⟶ *support* (λ*l* *s*. *H* *e* *l* *s*) ⊆ *T*
  ⟹ *support* (λ*l* *s*. *G* *e* ⟶ *H* *e* *l* *s*) ⊆ *T*
  unfolding *support_def* **apply**(*auto*)
   apply *blast*+ **done**

lemma *support_supportE*: ⋀*P* *e*. *support* (λ*l* *s*. *P* (*e* *l*) *s*) ⊆ *supportE* *e*
  unfolding *support_def* *supportE_def*
  apply(*rule* *subsetI*)
  apply(*simp*)
proof (*clarify*, *goal_cases*)
  case (*1* *P* *e* *x* *l1* *l2* *s*)
  have *P*: ∀*s*. *e* *l1* *s* = *e* *l2* *s* ⟹ *e* *l1* = *e* *l2* **by** *fast*
  show ∃*l1* *l2*. (∀*y*. *y* ≠ *x* ⟶ *l1* *y* = *l2* *y*) ∧ (∃*s*. *e* *l1* *s* ≠ *e* *l2* *s*)
    apply(*rule* *exI*[**where** *x*=*l1*])
    apply(*rule* *exI*[**where** *x*=*l2*])
    apply(*safe*)
    using *1* **apply** *blast*
    apply(*rule* *ccontr*)
    apply(*simp*)

```
    using 1(2) P by force
qed

lemma support_pre: support (pre C Q) ⊆ support Q ∪ varacom C
proof (induct C arbitrary: Q)
  case (Awhile A b C Q)
  obtain I2 S E Es SS where A: A= (I2,(S,(E,Es,SS))) using prod_cases5
by blast
  obtain I Is where I2=(I,Is) by fastforce
  note A=this A
  have support_inv: ⋀P. support (λl s. P s) = {}
    unfolding support_def by blast
  show ?case unfolding A  by(auto)
next
  case (Aseq C1 C2)
  then show ?case by(auto)
next
  case (Aif x C1 C2 Q)
  have s1: support (λl s. bval x s ⟶ pre C1 Q l s) ⊆ support Q ∪ varacom
C1
    apply(rule subset_trans[OF support_impl]) by(rule Aif)
   have s2: support (λl s. ~ bval x s ⟶ pre C2 Q l s) ⊆ support Q ∪
varacom C2
    apply(rule subset_trans[OF support_impl]) by(rule Aif)

  show ?case apply(simp)
    apply(rule subset_trans[OF support_and])
    using s1 s2 by blast
next
  case (Aconseq x1 x2 x3 C)
  obtain a b c d e f where x1=(a,b) x2=(c,d) x3=(e,f) by force
  with Aconseq show ?case by auto
qed (auto simp add: support_def)

lemma finite_support_pre: finite (support Q)  ⟹ finite (varacom C) ⟹
finite (support (pre C Q))
  using finite_subset support_pre finite_UnI by metis


fun time :: acom ⇒ tbd where
  time SKIP = (%s. Suc 0) |
  time (x ::= a) = (%s. Suc 0) |
  time (C₁;; C₂) = (%s. time C₁ s + preT C₁ (time C₂) s) |
  time ({_/_/(e,es)} CONSEQ C) = e |
```

time (*IF b THEN* C_1 *ELSE* C_2) =
(λ*s. if bval b s then 1* + *time* C_1 *s else 1* + *time* C_2 *s*) |
time ({(_,(E',(*E*,*x*)))} *WHILE b DO C*) = *E*

fun *kdeps* :: *acom* ⇒ *vname set* **where**
kdeps SKIP = {} |
kdeps (*x* ::= *a*) = {} |
kdeps (C_1;; C_2) = *kdeps* C_1 ∪ *fune* C_1 (*kdeps* C_2) |
kdeps (*IF b THEN* C_1 *ELSE* C_2) = *vars b* ∪ *kdeps* C_1 ∪ *kdeps* C_2 |
kdeps ({(_,(E',(*E*,*Es*,*SS*)))} *WHILE b DO C*) = *Es* |
kdeps ({_/_/(*e*,*es*)} *CONSEQ C*) = *es*

lemma *supportE_single*: *supportE* (λ*l s. P*) = {}
unfolding *supportE_def* **by** *blast*

lemma *supportE_plus*: *supportE* (λ*l s. e1 l s* + *e2 l s*) ⊆ *supportE e1* ∪ *supportE e2*
unfolding *supportE_def* **apply**(*auto*)
by *metis*

lemma *supportE_Suc*: *supportE* (λ*l s. Suc* (*e1 l s*)) = *supportE e1*
unfolding *supportE_def* **by** (*auto*)

lemma *supportE_single2*: *supportE* (λ*l . P*) = {}
unfolding *supportE_def* **by** *blast*

lemma *supportE_time*: *supportE* (λ*l. time C*) = {}
using *supportE_single2* **by** *simp*

lemma ⋀*s*. (∀ *l. I* (*l*(*x*:=*0*)) *s*) = (∀ *l. l x* = *0* ⟶ *I l s*)
apply(*auto*)
by (*metis fun_upd_triv*)

lemma ⋀*s*. (∀ *l. I* (*l*(*x*:=*Suc* (*l x*))) *s*) = (∀ *l*. (∃ *n. l x* = *Suc n*) ⟶ *I l s*)
apply(*auto*)
proof (*goal_cases*)
case (*1 s l n*)
then have ⋀*l. I* (*l*(*x* := *Suc* (*l x*))) *s* **by** *simp*
from *this*[**where** *l*=*l*(*x*:=*n*)]

have I $((l(x{:=}n))(x := Suc\ ((l(x{:=}n))\ x)))\ s$ **by** *simp*
then show *?case* **using** *1(2)* **apply**(*simp*)
by (*metis fun_upd_triv*)
qed

Verification condition:

definition *funStar* **where** $funStar\ f = (\%x.\ \{y.\ (x,y) \in \{(x,y).\ y \in f\ x\}^*\})$

lemma *funStart_prop1*: $x \in (funStar\ f)\ x$ **unfolding** *funStar_def* **by** *auto*
lemma *funStart_prop2*: $f\ x \subseteq (funStar\ f)\ x$ **unfolding** *funStar_def* **by** *auto*

fun $vc :: acom \Rightarrow assn2 \Rightarrow vname\ set \Rightarrow vname\ set \Rightarrow bool$ **where**
$vc\ SKIP\ Q\ \_\ \_ = True$ |
$vc\ (x ::= a)\ Q\ \_\ \_ = True$ |
$vc\ (C_1\ ;;\ C_2)\ Q\ LVQ\ LVE = ((vc\ C_1\ (pre\ C_2\ Q)\ (qdeps\ C_2\ LVQ)\ (fune\ C_2\ LVE \cup kdeps\ C_2)) \wedge (vc\ C_2\ Q\ LVQ\ LVE)\ )$ |
$vc\ (IF\ b\ THEN\ C_1\ ELSE\ C_2)\ Q\ LVQ\ LVE = (vc\ C_1\ Q\ LVQ\ LVE \wedge vc\ C_2\ Q\ LVQ\ LVE)$ |
$vc\ (\{(P',Ps)/(Q,Qs)/(e',es)\}\ CONSEQ\ C)\ Q'\ LVQ\ LVE = (vc\ C\ Q\ Qs\ LVE$ — evtl LV weglassen - glaub eher nicht
$\wedge\ (\forall s1\ s2\ l.\ (\forall x \in Ps.\ s1\ x = s2\ x) \longrightarrow P'\ l\ s1 = P'\ l\ s2)$ — annotation Ps (the set of variables P' depends on) is correct
$\wedge\ (\forall s1\ s2\ l.\ (\forall x \in Qs.\ s1\ x = s2\ x) \longrightarrow Q\ l\ s1 = Q\ l\ s2)$ — annotation Qs (the set of variables Q depends on) is correct
$\wedge\ (\forall s1\ s2.\ (\forall x \in es.\ s1\ x = s2\ x) \longrightarrow e'\ s1 = e'\ s2)$ — annotation es (the set of variables e' depends on) is correct
$\wedge\ (\exists k>0.\ (\forall l\ s.\ P'\ l\ s \longrightarrow time\ C\ s \le k * e'\ s\ \wedge (\forall t.\ \exists l'.\ (pre\ C\ Q)\ l'\ s \wedge (\ Q\ l'\ t \longrightarrow Q'\ l\ t)\ ))))$ |

$vc\ (\{((I,Is),(S,(E,es,SS)))\}\ WHILE\ b\ DO\ C)\ Q\ LVQ\ LVE = ((\forall s1\ s2\ l.\ (\forall x \in Is.\ s1\ x = s2\ x) \longrightarrow I\ l\ s1 = I\ l\ s2)$ — annotation Is is correct
$\wedge\ (\forall y \in LVE \cup LVQ.\ (let\ Ss{=}SS\ y\ in\ (\forall s1\ s2.\ (\forall x \in Ss.\ s1\ x = s2\ x) \longrightarrow (S\ s1)\ y = (S\ s2)\ y)))$ — annotation SS is correct, for only one step
$\wedge\ (\forall s1\ s2.\ (\forall x \in es.\ s1\ x = s2\ x) \longrightarrow E\ s1 = E\ s2)$ — annotation es (the set of variables E depends on) is correct
$\wedge\ (\forall l\ s.\ (I\ l\ s \wedge bval\ b\ s \longrightarrow pre\ C\ I\ l\ s \wedge\ \ E\ s \ge 1 + preT\ C\ E\ s + time\ C\ s$
$\wedge\ (\forall v \in (\bigcup y \in LVE \cup LVQ.\ (funStar\ SS)\ y).\ (S\ s)\ v = (S\ (postQ\ C\ s))\ v)\ ) \wedge$
$(I\ l\ s \wedge \neg\ bval\ b\ s \longrightarrow Q\ l\ s \wedge E\ s \ge 1 \wedge (\forall v \in (\bigcup y \in LVE \cup LVQ.\ (funStar\ SS)\ y).\ (S\ s)\ v\ = s\ v))\ ) \wedge$
$vc\ C\ I\ Is\ (es \cup (\bigcup y \in LVE.\ (funStar\ SS)\ y)))$

4.8.1 Soundness:

```
abbreviation preSet U C l s == (Ball U (%u. case u of (x,e,v) ⇒ l x =
preT C e s))
abbreviation postSet U l s == (Ball U (%u. case u of (x,e,v) ⇒ l x = e
s))


fun ListUpdate where
  ListUpdate f [] l = f
| ListUpdate f ((x,e,v)#xs) q = (ListUpdate f xs q)(x:=q e x)

lemma allg:
  assumes U2: ⋀l s n x. x∈ fst ` upds ⟹ A (l(x := n)) = A l
  shows
    fst ` set xs ⊆ fst ` upds ⟹ A (ListUpdate l'' xs q) = A l''
proof (induct xs)
  case (Cons a xs)
  obtain x e v where axe: a = (x,e,v)
    using prod_cases3 by blast
  have A (ListUpdate l'' (a # xs) q)
    = A ((ListUpdate l'' xs q)(x := q e x))   unfolding axe by(simp)
  also have
    ... = A (ListUpdate l'' xs q)
    apply(rule U2)
    using Cons axe by force
  also have ... = A l''
    using Cons by force
  finally show ?case .
qed simp

fun ListUpdateE where
  ListUpdateE f []   = f
| ListUpdateE f ((x,e,v)#xs) = (ListUpdateE f xs )(x:=e)

lemma ListUpdate_E: ListUpdateE f xs = ListUpdate f xs (%e x. e)
  apply(induct xs) apply(simp_all)
  subgoal for a xs apply(cases a) apply(simp) done
  done
lemma allg_E: fixes A::assn2
    assumes
    (⋀l s n x. x ∈ fst ` upds ⟹ A (l(x := n)) = A l) fst ` set xs ⊆ fst ` upds
   shows A (ListUpdateE f xs) = A f
proof -
```

```
  have A (ListUpdate f xs (%e x. e)) = A f
    apply(rule allg)
    apply fact+ done
  then show ?thesis by(simp only: ListUpdate_E)
qed

lemma ListUpdateE_updates: distinct (map fst xs) ⟹ x ∈ set xs ⟹
ListUpdateE l'' xs (fst x) = fst (snd x)
proof (induct xs)
  case Nil
  then show ?case apply(simp) done
next
  case (Cons a xs)
  show ?case
  proof (cases fst a = fst x)
    case True
    then obtain y e v where a: a=(y,e,v)
      using prod_cases3 by blast
    with True have fstx: fst x=y by simp
    from Cons(2,3) fstx  a have a2: x=a
      by force
    show ?thesis unfolding a2 a by(simp)
  next
    case False
    with Cons(3) have A: x∈set xs by auto
    then obtain y e v where a: a=(y,e,v)
      using prod_cases3 by blast
    from Cons(2) have B: distinct (map fst xs) by simp
    from Cons(1)[OF B A] False
      show ?thesis unfolding a by(simp)
  qed
qed


lemma ListUpdate_updates: x ∈ fst ‘ (set xs) ⟹ ListUpdate l'' xs (%e. l)
x = l x
proof(induct xs)
  case Nil
  then show ?case by(simp)
next
  case (Cons a xs)
  obtain q p v where axe: a = (p,q,v)
      using prod_cases3 by blast
  from Cons show ?case unfolding axe
```

```
    apply(cases x=p)
    by(simp_all)
qed

abbreviation lesvars xs == fst ` (set xs)

fun preList where
  preList [] C l s = True
| preList ((x,(e,v))#xs) C l s = (l x = preT C e s ∧ preList xs C l s)


lemma preList_Seq: preList upds (C1;; C2) l s = preList (map (λ(x, e, v).
(x, preT C2 e, fune C2 v)) upds) C1 l s
proof (induct upds)
  case Nil
  then show ?case by simp
next
  case (Cons a xs)
  obtain y e v where a: a=(y,(e,v))
    using prod_cases3 by blast
  from Cons show ?case unfolding a by (simp)
qed

lemma [simp]: support (λa b. True) = {}
  unfolding support_def
  by fast

lemma support_preList: support (preList upds C1) ⊆ lesvars upds
proof (induct upds)
  case Nil
  then show ?case by simp
next
  case (Cons a upds)
  obtain y e v where a: a=(y,(e,v))
    using prod_cases3 by blast
  from Cons show ?case unfolding a apply (simp)
    apply(rule subset_trans[OF support_and])
    apply(rule Un_least)
    subgoal apply(rule subset_trans[OF support_eq])
      using supportE_twicepreT subset_trans  supportE_single2 by simp
    subgoal by auto
    done
qed
```

lemma *preListpreSet*: *preSet* (*set xs*) *C l s* $\Longrightarrow$ *preList xs C l s*
proof (*induct xs*)
  case *Nil*
  then show *?case* **by** *simp*
next
  case (*Cons a xs*)
  obtain *y e v* **where** *a*: *a*=(*y*,(*e*,*v*))
    using *prod_cases3* **by** *blast*
  from *Cons* **show** *?case* **unfolding** *a* **by** (*simp*)
qed

lemma *preSetpreList*: *preList xs C l s* $\Longrightarrow$ *preSet* (*set xs*) *C l s*
proof (*induct xs*)
  case (*Cons a xs*)
  obtain *y e v* **where** *a*: *a*=(*y*,(*e*,*v*))
    using *prod_cases3* **by** *blast*
  from *Cons* **show** *?case* **unfolding** *a*
    by(*simp*)
qed *simp*

lemma *preSetpreList_eq*: *preList xs C l s* = *preSet* (*set xs*) *C l s*
proof (*induct xs*)
  case (*Cons a xs*)
  obtain *y e v* **where** *a*: *a*=(*y*,(*e*,*v*))
    using *prod_cases3* **by** *blast*
  from *Cons* **show** *?case* **unfolding** *a*
    by(*simp*)
qed *simp*

fun *postList* **where**
  postList [] *l s* = *True*
| *postList* ((*x*,*e*,*v*)#*xs*) *l s* = (*l x* = *e s* ∧ *postList xs l s*)

lemma *postList xs l s* = (*foldr* (λ(*x*,*e*,*v*) *acc l s*. *l x* = *e s* ∧ *acc l s*) *xs* (%*l s*. *True*)) *l s*
apply(*induct xs*) **apply**(*simp*) **by** (*auto*)

lemma *support_postList*: *support* (*postList xs*) ⊆ *lesvars xs*
proof (*induct xs*)
  case (*Cons a xs*)

obtain *y e v* **where** *a*: *a*=(*y*,(*e*,*v*))
  using *prod_cases3* **by** *blast*
from *Cons* **show** *?case* **unfolding** *a*
  apply(*simp*) **apply**(*rule subset_trans*[*OF support_and*])
  apply(*rule Un_least*)
  subgoal apply(*rule subset_trans*[*OF support_eq*])
    using *supportE_twicepreT subset_trans supportE_single2* **by** *simp*
  subgoal by(*auto*)
    done
qed *simp*

lemma *postList_preList*: *postList* (*map* (λ(*x*, *e*, *v*). (*x*, *preT C2 e*, *fune C2 v*)) *upds*) *l s* = *preList upds C2 l s*
proof (*induct upds*)
  case (*Cons a xs*)
  obtain *y e v* **where** *a*: *a*=(*y*,(*e*,*v*))
    using *prod_cases3* **by** *blast*
  from *Cons* **show** *?case* **unfolding** *a*
    by(*simp*)
qed *simp*

lemma *postSetpostList*: *postList xs l s* ⟹ *postSet* (*set xs*) *l s*
proof (*induct xs*)
  case (*Cons a xs*)
  obtain *y e v* **where** *a*: *a*=(*y*,(*e*,*v*))
    using *prod_cases3* **by** *blast*
  from *Cons* **show** *?case* **unfolding** *a*
    by(*simp*)
qed *simp*

lemma *postListpostSet*: *postSet* (*set xs*) *l s* ⟹ *postList xs l s*
proof (*induct xs*)
  case (*Cons a xs*)
  obtain *y e v* **where** *a*: *a*=(*y*,(*e*,*v*))
    using *prod_cases3* **by** *blast*
  from *Cons* **show** *?case* **unfolding** *a*
    by(*simp*)
qed *simp*

lemma *postListpostSet2*: *postList xs l s* = *postSet* (*set xs*) *l s*
  using *postListpostSet postSetpostList* **by** *metis*

lemma *ListAskip*: *preList xs Askip l s* = *postList xs l s*
 apply(*induct xs*)
  apply(*simp*) **by** *force*

lemma *SetAskip*: *preSet U Askip l s* = *postSet U l s*
by *simp*

lemma *ListAassign*: *preList upds* (*Aassign x1 x2*) *l s* = *postList upds l* (*s*[*x2*/*x1*])
 apply(*induct upds*)
  apply(*simp*) **by** *force*

lemma *SetAassign*: *preSet U* (*Aassign x1 x2*) *l s* = *postSet U l* (*s*[*x2*/*x1*])
by *simp*

lemma *ListAconseq*: *preList upds* (*Aconseq x1 x2 x3 C*) *l s* = *preList upds C l s*
 apply(*induct upds*)
  apply(*simp*) **by** *force*

lemma *SetAconseq*: *preSet U* (*Aconseq x1 x2 x3 C*) *l s* = *preSet U C l s*
by *simp*

lemma *ListAif1*: *bval b s* $\Longrightarrow$ *preList upds* (*IF b THEN C1 ELSE C2*) *l s* = *preList upds C1 l s*
 apply(*induct upds*)
  apply(*simp*) **by** *force*
lemma *SetAif1*: *bval b s* $\Longrightarrow$ *preSet upds* (*IF b THEN C1 ELSE C2*) *l s* = *preSet upds C1 l s*
 apply(*simp*) **done**
lemma *ListAif2*: ~ *bval b s* $\Longrightarrow$ *preList upds* (*IF b THEN C1 ELSE C2*) *l s* = *preList upds C2 l s*
 apply(*induct upds*)
  apply(*simp*) **by** *force*

lemma *SetAif2*: ~ *bval b s* $\Longrightarrow$ *preSet upds* (*IF b THEN C1 ELSE C2*) *l s* = *preSet upds C2 l s*
 apply(*simp*) **done**

definition *K* **where** *K C LVQ Q* == ($\forall$ *l s1 s2*. *s1* = *s2 on qdeps C LVQ*

$\longrightarrow$ *pre C Q l s1* = *pre C Q l s2*)

definition *K2* **where** *K2 C e Es Q* == ($\forall$ *s1 s2*. *s1* = *s2 on fune C Es* $\longrightarrow$ *preT C e s1* = *preT C e s2*)

definition *K3* **where** *K3 upds C Q* = ($\forall$ (*a*,*b*,*c*)$\in$*set upds*. *K2 C b c Q*)
definition *K4* **where** *K4 upds LV C Q* = (*K C LV Q* $\wedge$ *K3 upds C Q* $\wedge$ ($\forall$ *s1 s2*. *s1* = *s2 on kdeps C* $\longrightarrow$ *time C s1* = *time C s2*))

lemma *k4If*: *K4 upds LVQ C1 Q* $\Longrightarrow$ *K4 upds LVQ C2 Q* $\Longrightarrow$ *K4 upds LVQ* (*IF b THEN C1 ELSE C2*) *Q*
proof –
  have *fl*: $\bigwedge$*A B s1 s2*. *A* $\subseteq$ *B* $\Longrightarrow$ *s1* = *s2 on B* $\Longrightarrow$ *s1* = *s2 on A* **by** *auto*
  assume *K4 upds LVQ C1 Q K4 upds LVQ C2 Q*
  then show *K4 upds LVQ* (*IF b THEN C1 ELSE C2*) *Q*
   unfolding *K4_def K_def K3_def K2_def* **using** *bval_eq_if_eq_on_vars fl* **apply** *auto*
      apply *blast*+ **done**
qed

4.8.2 Soundness

lemma *vc_sound*: *vc C Q LVQ LVE* $\Longrightarrow$ *finite* (*support Q*)
  $\Longrightarrow$ *fst* ‘ (*set upds*) $\cap$ *varacom C* = {} $\Longrightarrow$ *distinct* (*map fst upds*)
  $\Longrightarrow$ *finite* (*varacom C*)
  $\Longrightarrow$ ($\forall$ *l s1 s2*. *s1* = *s2 on LVQ* $\longrightarrow$ *Q l s1* = *Q l s2*)
  $\Longrightarrow$ ($\forall$ *l s1 s2*. *s1* = *s2 on LVE* $\longrightarrow$ *postList upds l s1* = *postList upds l s2*)
  $\Longrightarrow$ ($\forall$ (*a*,*b*,*c*)$\in$*set upds*. ($\forall$ *s1 s2*. *s1* = *s2 on c* $\longrightarrow$ *b s1* = *b s2*)) — *c* are really the variables *b* depends on
  $\Longrightarrow$ ($\bigcup$ (*a*,*b*,*c*)$\in$*set upds*. *c*) $\subseteq$ *LVE* — in *LV* are all the variables that the expressions in *upds* depend on
  $\Longrightarrow$ $\vdash_1$ {%*l s*. *pre C Q l s* $\wedge$ *preList upds C l s*} *strip C* { *time C* $\Downarrow$ %*l s*. *Q l s* $\wedge$ *postList upds l s*}
  $\wedge$ (($\forall$ *l s*. *pre C Q l s* $\longrightarrow$ *Q l* (*postQ C s*)) $\wedge$ *K4 upds LVQ C Q*)
proof(*induction C arbitrary*: *Q upds LVE LVQ*)
  case (*Askip Q upds*)
  then show *?case* **unfolding** *K4_def K_def K3_def K2_def*
    apply(*auto*)
    apply(*rule weaken_post*[**where** *Q*=%*l s*. *Q l s* $\wedge$ *preList upds Askip l s*])
     apply(*simp add*: *Skip*) **using** *ListAskip*

```
    by fast
next
  case (Aassign x1 x2 Q upds)
  then show ?case unfolding K_def apply(safe) apply(auto simp add:
Assign)[1]
     apply(rule weaken_post[where Q=%l s. Q l s ∧ postList upds l s])
      apply(simp only: ListAassign)
      apply(rule Assign) apply simp
     apply(simp only: postQ.simps pre.simps) apply(auto)
        unfolding K4_def K2_def K3_def K_def by (auto)
next
  case (Aif b C1 C2 Q upds )
  from Aif(3) have 1: vc C1 Q LVQ LVE and 2: vc C2 Q LVQ LVE by
auto
  have T: ⋀l s. pre C1 Q l s ⟹ bval b s ⟹ Q l (postQ C1 s)
      and kT: K4 upds LVQ C1 Q
    using Aif(1)[OF 1 Aif(4) _ Aif(6)] Aif(5−11) by auto
  have F: ⋀l s. pre C2 Q l s ⟹ ¬ bval b s ⟹ Q l (postQ C2 s)
    and kF: K4 upds LVQ C2 Q
    using Aif(2)[OF 2 Aif(4) _ Aif(6)] Aif(5−11) by auto

  show ?case apply(safe)
    subgoal
      apply(simp)
     apply(rule If2[where e=λa. if bval b a then  time C1 a else time C2
a])
    subgoal
      apply(simp cong: rev_conj_cong)
      apply(rule ub_cost[where e′=time C1])
       apply(simp) apply(auto)[1]
     apply(rule strengthen_pre[where P=%l s. pre C1 Q l s ∧ preList upds
C1 l s])
        using ListAif1
       apply fast
        apply(rule Aif(1)[THEN conjunct1])
          using Aif
           apply(auto)
      done
    subgoal
      apply(simp cong: rev_conj_cong)
      apply(rule ub_cost[where e′=time C2])
       apply(simp) apply(auto)[1]
     apply(rule strengthen_pre[where P=%l s. pre C2 Q l s ∧ preList upds
C2 l s])
```

```
        using ListAif2
       apply fast
      apply(rule Aif(2)[THEN conjunct1])
          using Aif
           apply(auto)
          done
        by simp
    using T F kT kF  by (auto intro: k4If)
next
  case (Aconseq P'2 Qannot2 eannot2 C Q upds)
  obtain P' Ps where [simp]: P'2 = (P',Ps) by fastforce
  obtain Qannot Q's where [simp]: Qannot2 = (Qannot,Q's) by fastforce
  obtain eannot es where [simp]: eannot2 = (eannot,es) by fastforce

  have ih0: finite (support Qannot) using Aconseq(3,6) by simp

  from ‹vc ({P'2/Qannot2/eannot2} CONSEQ C) Q LVQ LVE›
  obtain k where k0: k>0 and ih1: vc C Qannot Q's LVE
    and ih2:  (∀ l s. P' l s ⟶  time C s ≤ k * eannot s ∧ (∀ t. ∃ l'. pre C
Qannot l' s ∧ (Qannot l' t ⟶ Q l t)))
    and pc: (∀ s1 s2 l. (∀ x∈Ps. s1 x=s2 x) ⟶ P' l s1 = P' l s2)
     and qc: (∀ s1 s2 l. (∀ x∈Q's. s1 x=s2 x) ⟶ Qannot l s1 = Qannot l
s2)
    and ec: (∀ s1 s2. (∀ x∈es. s1 x=s2 x) ⟶ eannot s1 = eannot s2)
    by auto
  have  k: ⊢₁ {λl s. pre C Qannot l s ∧ preList upds C l s} strip C { time
C ⇓ λl s. Qannot l s ∧ postList upds l s}
    ∧ ((∀ l s. pre C Qannot l s ⟶ Qannot l (postQ C s)) ∧ K4 upds Q's C
Qannot)
    apply(rule Aconseq(1)) using Aconseq(2−10) by auto

  note ih=k[THEN conjunct1] and ihsnd=k[THEN conjunct2]

  show ?case apply(simp, safe)
     apply(rule conseq[where e=time C and P=λl s. pre C Qannot l s ∧
preList upds C l s and Q=%l s. Qannot l s ∧ postList upds l s])
      prefer 2
      apply(rule ih)
    subgoal apply(rule exI[where x=k])
    proof (safe, goal_cases)
      case (1)
      with k0 show ?case by auto
    next
      case (2 l s)
```

then show *?case* **using** *ih2* **by** *simp*
next
case (*3 l s t*)
have *finupds*: *finite* (*set upds*) **by** *simp*
{
fix *l s n x*
assume $x \in fst$ ‘ (*set upds*)
then have $x \notin$ *support* (*pre C Qannot*) **using** *Aconseq*(*4*) *support_pre* **by** *auto*
from *assn2_lupd*[*OF this*] **have** *pre C Qannot* $(l(x := n))$ = *pre C Qannot l* **.**
} **note** *U2=this*
{
fix *l s n x*
assume $x \in fst$ ‘ (*set upds*)
then have $x \notin$ *support Qannot* **using** *Aconseq*(*4*) **by** *auto*
from *assn2_lupd*[*OF this*] **have** *Qannot* $(l(x := n))$ = *Qannot l* **.**
} **note** *K2=this*

from *ih2 3*(*1*) **have** $*$: $(\exists l'.$ *pre C Qannot* l' $s \wedge$ (*Qannot* l' $t \longrightarrow Q\ l\ t$)) **by** *simp*
obtain l' **where** i': *pre C Qannot* l' *s* **and** ii': (*Qannot* l' $t \longrightarrow Q\ l\ t$)
and *lxlx*: $\bigwedge x.\ x \in fst$ ‘ (*set upds*) $\Longrightarrow l'\ x = l\ x$
proof (*goal_cases*)
case *1*
from $*$ **obtain** l'' **where** i': *pre C Qannot* l'' *s* **and** ii': (*Qannot* l'' $t \longrightarrow Q\ l\ t$)
by *blast*

note *allg=allg*[**where** *q=%e x. l x*]

have *pre C Qannot* (*ListUpdate* l'' *upds* ($\lambda e.\ l$)) = *pre C Qannot* l''

apply(*rule allg*[**where** *?upds=set upds*]) **apply**(*rule U2*) **apply** *fast* **by** *fast*
with i' **have** *U*: *pre C Qannot* (*ListUpdate* l'' *upds* ($\lambda e.\ l$)) *s* **by** *simp*

have *Qannot* (*ListUpdate* l'' *upds* ($\lambda e.\ l$)) = *Qannot* l''
apply(*rule allg*[**where** *?upds=set upds*]) **apply**(*rule K2*) **apply** *fast* **by** *fast*

then have *K*: (%l' *s*. *Qannot* l' $t \longrightarrow Q\ l\ t$) (*ListUpdate* l'' *upds* ($\lambda e.\ l$)) s = (%l' *s*. *Qannot* l' $t \longrightarrow Q\ l\ t$) l'' *s*

by *simp*
with *ii'* **have** *K*: (*Qannot* (*ListUpdate* l'' *upds* ($\lambda e.$ *l*)) *t* $\longrightarrow$ *Q l t*)
by *simp*

{
fix *x*
assume *as*: $x \in$ *fst* ‘ (*set upds*)
have *ListUpdate* l'' *upds* ($\lambda e.$ *l*) *x* = *l x*
apply(*rule ListUpdate_updates*)
using *as* **by** *fast*
} **note** *kla=this*

show *thesis*
apply(*rule 1*)
apply(*fact U*)
apply(*fact K*)
apply(*fact kla*)
done
qed

let *?upds'* = *set* (*map* (%(*x*,*e*,*v*). (*x*,*preT C e s*,*fune C v*)) *upds*)
have *finite ?upds'* **by** *simp*
define *xs* **where** *xs* = *map* (%(*x*,*e*,*v*). (*x*,*preT C e s*,*fune C v*)) *upds*
then have *set xs*= *?upds'* **by** *simp*

have *pre C Qannot* (*ListUpdateE l' xs*) = *pre C Qannot l'*
apply(*rule allg_E*[**where** *?upds=?upds'*]) **apply**(*rule U2*)
apply *force* **unfolding** *xs_def* **by** *simp*
with *i'* **have** *U*: *pre C Qannot* (*ListUpdateE l' xs*) *s* **by** *simp*

have *Qannot* (*ListUpdateE l' xs*) = *Qannot l'*
apply(*rule allg_E*[**where** *?upds=?upds'*]) **apply**(*rule K2*) **apply**
force **unfolding** *xs_def* **by** *auto*
then have *K*: (%*l' s. Qannot l' t* $\longrightarrow$ *Q l t*) (*ListUpdateE l' xs*) *s* =
(%*l' s. Qannot l' t* $\longrightarrow$ *Q l t*) *l' s*
by *simp*
with *ii'* **have** *K*: (*Qannot* (*ListUpdateE l' xs*) *t* $\longrightarrow$ *Q l t*) **by** *simp*

have *xs_upds*: *map fst xs* = *map fst upds*
unfolding *xs_def* **by** *auto*

have *grr*: $\bigwedge x.$ $x \in$ *?upds'* $\Longrightarrow$ *ListUpdateE l' xs* (*fst x*) = *fst* (*snd x*)
apply(*rule ListUpdateE_updates*)

apply(*simp only*: *xs_upds*) **using** *Aconseq*(*5*) **apply** *simp*
unfolding *xs_def* **apply**(*simp*) **done**
show *?case*
apply(*rule exI*[**where** *x*=*ListUpdateE l′ xs*])
apply(*safe*)
subgoal by *fact*
subgoal apply(*rule preListpreSet*) **proof** (*safe*,*goal_cases*)
case (*1 x e v*)
then have (*x*, *preT C e s*, *fune C v*) ∈ *?upds′*
by *force*
from *grr*[*OF this*, *simplified*]
show *?case* **.**

qed
subgoal using *K* **apply**(*simp*) **done**
subgoal apply(*rule postListpostSet*)
proof (*safe*, *goal_cases*)
case (*1 x e v*)
with *lxlx*[*of x*] **have** *fF*: *l x* = *l′ x*
by *force*

from *postSetpostList*[*OF 1*(*2*)] **have** *g*: *postSet* (*set upds*) (*ListUpdateE l′ xs*) *t* **.**
with *1*(*3*) **have** *A*: (*ListUpdateE l′ xs*) *x* = *e t*
by *fast*
from *1*(*3*) *grr*[*of* (*x*,*preT C e s*, *fune C v*)] **have** *B*: *ListUpdateE l′ xs x* = *fst* (*snd* (*x*, *preT C e s*, *fune C v*))
by *force*
from *A B* **have** *X*: *e t* = *preT C e s* **by** *fastforce*
from *preSetpreList*[*OF 3*(*2*)] **have** *preSet* (*set upds*) ({*P′2*/*Qannot2*/*eannot2*} *CONSEQ C*) *l s* **apply**(*simp*) **done**
with *1*(*3*) **have** *Y*: *l x* = *preT C e s* **apply**(*simp*) **by** *fast*
from *X Y* **show** *?case* **by** *simp*
qed
done
qed
subgoal using *ihsnd ih2* **by** *blast*
subgoal using *ihsnd*[*THEN conjunct2*] *pc* **unfolding** *K4_def K_def* **apply**(*auto*)
unfolding *K3_def K2_def* **using** *ec* **by** *auto*
done
next
case (*Aseq C1 C2 Q upds*)

let *?P* = (λ*l s. pre C1* (*pre C2 Q*) *l s* $\wedge$ *preList upds* (*C1*;;*C2*) *l s*)
let *?P'* = *support Q* $\cup$ *varacom C1* $\cup$ *varacom C2* $\cup$ *lesvars upds*

have *finite_varacom*: *finite* (*varacom* (*C1*;; *C2*)) **by** *fact*
have *finite_varacomC2*: *finite* (*varacom C2*)
apply(*rule finite_subset*[*OF* _ *finite_varacom*]) **by** *simp*

let *?y* = *SOME x. x* $\notin$ *?P'*
have *sup_L*: *support* (*preList upds* (*C1*;;*C2*)) $\subseteq$ *lesvars upds*
apply(*rule support_preList*) **done**

have *sup_B*: *support ?P* $\subseteq$ *?P'*
apply(*rule subset_trans*[*OF support_and*]) **using** *support_pre sup_L* **by** *blast*
have *fP'*: *finite* (*?P'*) **using** *finite_varacom Aseq*(*3*,*4*,*5*) **apply** *simp* **done**
hence $\exists$ *x. x* $\notin$ *?P'* **using** *infinite_UNIV_listI*
using *ex_new_if_finite* **by** *metis*
hence *ynP'*: *?y* $\notin$ *?P'* **by** (*rule someI_ex*)
hence *ysupPreC2Q*: *?y* $\notin$ *support* (*pre C2 Q*) **and** *ysupC1*: *?y* $\notin$ *varacom C1* **using** *support_pre* **by** *auto*

from *Aseq*(*5*) **have** *lesvars upds* $\cap$ *varacom C2* = {} **by** *auto*

from *Aseq* **show** *?case* **apply**(*auto*)
proof (*rule Seq*, *goal_cases*)
case *2*
show $\vdash_1$ {(%*l s. pre C2 Q l s* $\wedge$ *preList upds C2 l s*)} *strip C2* { *time C2* $\Downarrow$ (%*l s. Q l s* $\wedge$ *postList upds l s*)}
apply(*rule weaken_post*[**where** *Q*=(%*l s. Q l s* $\wedge$ *postList upds l s*)])
apply(*rule 2*(*2*)[*THEN conjunct1*])
apply *fact*
apply (*fact*)+ **using** *2*(*8*) **by** *simp*
next
case *3*
fix *s*
show *time C1 s* + *preT C1* (*time C2*) *s* $\leq$ *time C1 s* + *preT C1* (*time C2*) *s*
by *simp*
next
case *1*

from *ynP'* **have** *yC1*: *?y* $\notin$ *varacom C1* **by** *blast*
have *xC1*: *lesvars upds* $\cap$ *varacom C1* = {} **using** *Aseq*(*5*) **by** *auto*
from *finite_support_pre*[*OF Aseq*(*4*) *finite_varacomC2*]
have *G*: *finite* (*support* (*pre C2 Q*)) **.**

let *?upds* = *map* (λ*a. case a of* (*x*,*e*,*v*) $\Rightarrow$ (*x*, *preT C2 e*, *fune C2 v*)) *upds*
let *?upds'* = (*?y*,*time C2*, *kdeps C2*)#*?upds*

{
have *A*: *lesvars ?upds'* = {*?y*} $\cup$ *lesvars upds* **apply** *simp*
by *force*
from *Aseq*(*5*) **have** *2*: *lesvars upds* $\cap$ *varacom C1* = {} **by** *auto*
have *lesvars ?upds'* $\cap$ *varacom C1* = {}
unfolding *A* **using** *ysupC1 2* **by** *blast*
} **note** *klar*=*this*

have *t*: *fst* $\circ$ (λ(*x*, *e*, *v*). (*x*, *preT C2 e*, *fune C2 v*)) = *fst* **by** *auto*

{
fix *a b c X*
assume *a* $\notin$ *lesvars X* (*a*,*b*,*c*) $\in$ *set X*
then have *False* **by** *force*
} **note** *helper*=*this*

have *dmap*: *distinct* (*map fst ?upds'*)
apply(*auto simp add*: *t*)
subgoal for *e* **apply**(*rule helper*[*of ?y upds e*]) **using** *ynP'* **by** *auto*
subgoal by *fact*
done
note *bla1*=*1*(*1*)[**where** *Q*=*pre C2 Q* **and** *upds*=*?upds'*, *OF 1*(*10*) *G klar dmap*]

note *bla*=*1*(*2*)[*OF 1*(*11*,*3*), *THEN conjunct2*, *THEN conjunct2*]
from *1*(*4*) **have** *kal*: *lesvars upds* $\cap$ *varacom C2* = {} **by** *auto*
from *bla*[*OF kal Aseq.prems*(*4*,*6*,*7*,*8*,*9*)] **have** *bla4*: *K4 upds LVQ C2 Q* **by** *auto*
then **have** *bla*: *K C2 LVQ Q* **unfolding** *K4_def* **by** *auto*

have *A*:
$\vdash_1$ {λ*l s. pre C1* (*pre C2 Q*) *l s* $\wedge$ *preList ?upds' C1 l s*}
strip C1
{ *time C1* $\Downarrow$ λ*l s. pre C2 Q l s* $\wedge$ *postList ?upds' l s*} $\wedge$

(∀ *l s*. *pre C1* (*pre C2 Q*) *l s* ⟶ *pre C2 Q l* (*postQ C1 s*)) ∧ *K4 ?upds′* (*qdeps C2 LVQ*) *C1* (*pre C2 Q*)
apply(*rule 1*(*1*)[**where** *Q*=*pre C2 Q* **and** *upds*=*?upds′*, *OF 1*(*10*) *G klar dmap*])
proof (*goal_cases*)
case *1*
then show *?case* **using** *bla* **unfolding** *K_def* **by** *auto*
next
case *2*
show *?case* **apply**(*rule*,*rule*,*rule*,*rule*) **proof** (*goal_cases*)
case (*1 l s1 s2*)
then show *?case* **using** *bla4* **using** *Aseq.prems*(*9*) **unfolding** *K4_def K3_def K2_def*
apply(*simp*)
proof (*goal_cases*)
case *1*
then have *t*: *time C2 s1* = *time C2 s2* **by** *auto*

have *post*: *postList* (*map* (λ(*x*, *e*, *v*). (*x*, *preT C2 e*, *fune C2 v*)) *upds*) *l s1* = *postList* (*map* (λ(*x*, *e*, *v*). (*x*, *preT C2 e*, *fune C2 v*)) *upds*) *l s2* (**is** *?IH upds*)
using *1*
proof (*induct upds*)
case (*Cons a upds*)
then have *IH*: *?IH upds* **by** *auto*
obtain *x e v* **where** *a*: *a* = (*x*,*e*,*v*) **using** *prod_cases3* **by** *blast*
from *Cons*(*4*) **have** *v* ⊆ *LVE* **unfolding** *a* **by** *auto*
with *Cons*(*2*) **have** *s12v*: *s1* = *s2 on fune C2 v* **unfolding** *a* **using** *fune_mono* **by** *blast*
with *Cons*(*3*) *IH a* **show** *?case* **by** *auto*
qed *auto*

from *post t* **show** *?case* **by** *auto*
qed
qed
next
case *3*
then show *?case* **using** *bla4* **unfolding** *K4_def K3_def K2_def* **by**(*auto*)
next
case *4*
then show *?case* **apply**(*auto*)
proof (*goal_cases*)
case (*1 x a aa b*)

with *Aseq.prems*(*9*) **have** *b* ⊆ *LVE* **by** *auto*
with *fune_mono* **have** *fune C2 b* ⊆ *fune C2 LVE* **by** *auto*
with *1* **show** *?case* **by** *blast*
qed
qed

show $\vdash_1$ {*λl s.* (*pre C1* (*pre C2 Q*) *l s* ∧ *preList upds* (*C1*;; *C2*) *l s*) ∧ *l ?y* = *preT C1* (*time C2*) *s*} *strip C1*
{ *time C1* ⇓ *λl s.* (*pre C2 Q l s* ∧ *preList upds C2 l s*) ∧ *time C2 s* ≤ *l ?y*}
apply(*rule conseq_old*)
prefer *2*
apply(*rule A*[*THEN conjunct1*])
apply(*auto simp*: *preList_Seq postList_preList*) **done**

from *A*[*THEN conjunct2*, *THEN conjunct2*] **have** *A1*: *K C1* (*qdeps C2 LVQ*) (*pre C2 Q*)
and *A2*: *K3 ?upds′ C1* (*pre C2 Q*) **and** *A3*: (∀ *s1 s2*. *s1* = *s2 on kdeps C1* ⟶ *time C1 s1* = *time C1 s2*) **unfolding** *K4_def* **by** *auto*
from *bla4* **have** *B1*: *K C2 LVQ Q* **and** *B2*: *K3 upds C2 Q* **and** *B3*: (∀ *s1 s2*. *s1* = *s2 on kdeps C2* ⟶ *time C2 s1* = *time C2 s2*) **unfolding** *K4_def* **by** *auto*
show *K4 upds LVQ* (*C1*;; *C2*) *Q*
unfolding *K4_def* **apply**(*safe*)
subgoal using *A1 B1* **unfolding** *K_def* **by**(*simp*)
subgoal using *A2 B2* **unfolding** *K3_def K2_def* **apply**(*auto*) **done**
subgoal for *s1 s2* **using** *A3 B3* **apply** *auto*
proof (*goal_cases*)
case *1*
then have *t*: *time C1 s1* = *time C1 s2* **by** *auto*
from *A2* **have** ∀ *s1 s2*. *s1* = *s2 on fune C1* (*kdeps C2*) ⟶ *preT C1* (*time C2*) *s1* = *preT C1* (*time C2*) *s2* **unfolding** *K3_def K2_def* **by** *auto*
then have *p*: *preT C1* (*time C2*) *s1* = *preT C1* (*time C2*) *s2*
using *1*(*1*) **by** *simp*
from *t p* **show** *?case* **by** *auto*
qed
done
next
from *ynP′ sup_B* **show** *?y* ∉ *support ?P* **by** *blast*
have *F*: *support* (*preList upds C2*) ⊆ *lesvars upds*
apply(*rule support_preList*) **done**
have *support* (*λl s. pre C2 Q l s* ∧ *preList upds C2 l s*) ⊆ *?P′*
apply(*rule subset_trans*[*OF support_and*]) **using** *F support_pre* **by**

blast
with *ynP'*
show *?y* $\notin$ *support* (λ*l s. pre C2 Q l s* $\wedge$ *preList upds C2 l s*) **by** *blast*
next
case (*6 l s*)

note *bla=6*(*2*)[*OF 6*(*11*,*3*), *THEN conjunct2*, *THEN conjunct2*]
from *6*(*4*) **have** *kal*: *lesvars upds* $\cap$ *varacom C2* = {} **by** *auto*
from *bla*[*OF kal Aseq.prems*(*4*,*6*,*7*,*8*,*9*)] **have** *bla4*: *K4 upds LVQ C2 Q* **by** *auto*
then **have** *bla*: *K C2 LVQ Q* **unfolding** *K4_def* **by** *auto*

have *11*: *finite* (*support* (*pre C2 Q*))
apply(*rule finite_subset*[*OF support_pre*])
using *6*(*3*,*4*,*10*) *finite_varacomC2* **by** *blast*
have *A*: $\forall$ *l s. pre C1* (*pre C2 Q*) *l s* $\longrightarrow$ *pre C2 Q l* (*postQ C1 s*)
apply(*rule 6*(*1*)[**where** *upds*=[], *THEN conjunct2*, *THEN conjunct1*])
apply(*fact*)+ **apply**(*auto*) **using** *bla* **unfolding** *K_def* **apply** *blast*+ **done**
have *B*: ($\forall$ *l s. pre C2 Q l s* $\longrightarrow$ *Q l* (*postQ C2 s*))
apply(*rule 6*(*2*)[**where** *upds*=[], *THEN conjunct2*, *THEN conjunct1*])
apply(*fact*)+ **apply** *auto* **using** *Aseq.prems*(*6*) **by** *auto*
from *A B 6* **show** *?case* **by** *simp*
qed
next
case (*Awhile A b C Q upds*)
obtain *I2 S E Es SS* **where** *aha*[*simp*]: *A* = (*I2*,(*S*,(*E*,*Es*,*SS*))) **using** *prod_cases5* **by** *blast*
obtain *I Is* **where** *aha2*: *I2* = (*I*, *Is*)
by *fastforce*
let *?LV* =($\bigcup$*y*$\in$*LVE* $\cup$ *LVQ*. (*funStar SS*) *y*)
have *LVE_LVE*: *LVE* $\subseteq$ ($\bigcup$*y*$\in$*LVE*. (*funStar SS*) *y*) **using** *funStart_prop1* **by** *fast*
have *LV_LV*: *LVE* $\cup$ *LVQ* $\subseteq$ *?LV* **using** *funStart_prop1* **by** *fast*
have *LV_LV2*: ($\bigcup$*y*$\in$*LVE* $\cup$ *LVQ*. *SS y*) $\subseteq$ *?LV* **using** *funStart_prop2* **by** *fast*
have *LVE_LV2*: ($\bigcup$*y*$\in$*LVE*. *SS y*) $\subseteq$ ($\bigcup$*y*$\in$*LVE*. (*funStar SS*) *y*) **using** *funStart_prop2* **by** *fast*
note *aha* = *aha2 aha*
with *aha aha2* ‹*vc* (*Awhile A b C*) *Q LVQ LVE*› **have** *vc* (*Awhile* ((*I*,*Is*),*S*,*E*,*Es*,*SS*) *b C*) *Q LVQ LVE* **apply** *auto* **apply** *fast*+ **done**
then
have *vc*: *vc C I Is* (*Es* $\cup$ ($\bigcup$*y*$\in$*LVE*. (*funStar SS*) *y*))

and *IQ*: $\forall\, l\ s.\ (I\ l\ s \wedge bval\ b\ s \longrightarrow pre\ C\ I\ l\ s \wedge\ 1 + preT\ C\ E\ s + time\ C\ s \leq E\ s \wedge S\ s = S\ (postQ\ C\ s)\ on\ ?LV)$ **and**
pre: $\forall\, l\ s.\ (I\ l\ s \wedge \neg\ bval\ b\ s \longrightarrow Q\ l\ s \wedge 1 \leq E\ s \wedge S\ s = s\ on\ ?LV)$
and *Is*: $(\forall\, s1\ s2\ l.\ s1 = s2\ on\ Is \longrightarrow I\ l\ s1 = I\ l\ s2)$
and *Ss*: $(\forall\, y{\in}LVE \cup LVQ.\ let\ Ss = SS\ y\ in\ \forall\, s1\ s2.\ s1 = s2\ on\ Ss \longrightarrow S\ s1\ y = S\ s2\ y)$
and *Es*: $(\forall\, s1\ s2.\ s1 = s2\ on\ Es \longrightarrow E\ s1 = E\ s2)$ **apply** *simp_all* **apply** *auto* **apply** *fast+* **done**

then have *pre2*: $\bigwedge l\ s.\ I\ l\ s \Longrightarrow \neg\ bval\ b\ s \Longrightarrow Q\ l\ s \wedge 1 \leq E\ s \wedge S\ s = s\ on\ ?LV$
and *IQ2*: $\bigwedge l\ s.\ (I\ l\ s \Longrightarrow bval\ b\ s \Longrightarrow pre\ C\ I\ l\ s \wedge\ 1 + preT\ C\ E\ s + time\ C\ s \leq E\ s \wedge S\ s = S\ (postQ\ C\ s)\ on\ ?LV)$
and *Ss2*: $\bigwedge y\ s1\ s2.\ s1 = s2\ on\ (\bigcup y{\in}LVE.\ SS\ y) \Longrightarrow S\ s1 = S\ s2\ on\ LVE$
by *auto*

from *Ss* **have** *Ssc*: $\bigwedge c\ s1\ s2.\ c \subseteq LVE \Longrightarrow\ s1 = s2\ on\ (\bigcup y{\in}c.\ SS\ y) \Longrightarrow S\ s1 = S\ s2\ on\ c$
by *auto*

from *IQ* **have** *IQ_in*: $\bigwedge l\ s.\ I\ l\ s \Longrightarrow bval\ b\ s \Longrightarrow S\ s = S\ (postQ\ C\ s)\ on\ ?LV$ **by** *auto*

have *inv_impl*: $\bigwedge l\ s.\ I\ l\ s \Longrightarrow bval\ b\ s \Longrightarrow pre\ C\ I\ \ l\ s$ **using** *IQ* **by** *auto*

have *yC*: $lesvars\ upds \cap varacom\ C = \{\}$ **using** *Awhile(4) aha* **by** *auto*

let $?upds = map\ (\%(x,e,v).\ (x,\ \%s.\ e\ (S\ s),\ \bigcup x{\in}v.\ SS\ x))\ upds$
let $?INV = \%l\ s.\ I\ l\ s \wedge postList\ ?upds\ l\ s$

have $lesvars\ upds \cap support\ I = \{\}$ **using** *Awhile(4)* **unfolding** *aha* **by** *auto*

let $?P{=}lesvars\ upds \cup varacom\ (\{A\}\ WHILE\ b\ DO\ C)$
let $?z{=}SOME\ z{::}lvname.\ z \notin\ ?P$
have *finite ?P* **apply**(*auto simp del*: *aha*) **by** (*fact Awhile(6)*)
hence $\exists\, z.\ z{\notin}?P$ **using** *infinite_UNIV_listI*
using *ex_new_if_finite* **by** *metis*
hence *znP*: $?z \notin\ ?P$ **by** (*rule someI_ex*)
from *znP* **have**
zny: $?z \notin lesvars\ upds$
and *zI*: $?z \notin support\ I$

and *blb*: *?z* $\notin$ *varacom C* **by** (*simp_all add*: *aha*)

from *Awhile*(*4*,*6*) **have** *23*: *finite* (*varacom C*)
and *26*: *finite* (*support I*) **by** (*auto simp add*: *finite_subset aha*)

have $\forall$ *l s*. *pre C I l s* $\longrightarrow$ *I l* (*postQ C s*)
apply(*rule Awhile*(*1*)[*THEN conjunct2*, *THEN conjunct1*])
apply(*fact*)+ **subgoal using** *Is* **apply** *auto* **done**
subgoal using *Awhile*(*8*) *LVE_LVE* **by** (*metis subsetD sup.cobounded2*)
apply *fact* **using** *Awhile*(*10*) *LVE_LVE* **by** *blast*
hence *step*: $\bigwedge$*l s*. *pre C I l s* $\Longrightarrow$ *I l* (*postQ C s*) **by** *simp*

have *fua*: *lesvars ?upds* = *lesvars upds*
by *force*
let *?upds'* = (*?z*,*E*,*Es*) # *?upds*

show *?case*
proof (*safe*, *goal_cases*)
case (*2 l s*)
from *2* **have** *A*: *I l s* **unfolding** *aha* **by**(*simp*)
then have *I*: *I l s* **by** *simp*

{ fix *n*
have *E s* = *n* $\Longrightarrow$ *I l s* $\Longrightarrow$ *Q l* (*postQ* ({*A*} *WHILE b DO C*) *s*)
proof (*induct n arbitrary*: *s l rule*: *less_induct*)
case (*less n*)
then show *?case*
proof (*cases bval b s*)
case *True*
with *less IQ2* **have** *pre C I l s* **and** *S*: *S s* = *S* (*postQ C s*) *on ?LV* **and** *t*: *1* + *preT C E s* + *time C s* $\leq$ *E s* **by** *auto*
with *step* **have** *I'*: *I l* (*postQ C s*) **and** *1* + *E* (*postQ C s*) + *time C s* $\leq$ *E s* **using** *TQ* **by** *auto*
with *less* **have** *E* (*postQ C s*) < *n* **by** *auto*
with *less*(*1*) *I'* **have** *Q l* (*postQ* ({*A*} *WHILE b DO C*) (*postQ C s*)) **by** *auto*
with *step* **show** *?thesis* **using** *S* **apply** *simp* **using** *Awhile*(*7*)
by (*metis* (*no_types*, *lifting*) *LV_LV SUP_union contra_subsetD sup.boundedE*)
next
case *False*
with *pre2 less*(*3*) **have** *Q l s S s* = *s on ?LV* **by** *auto*
then show *?thesis* **apply** *simp* **using** *Awhile*(*7*)
by (*metis* (*no_types*, *lifting*) *LV_LV SUP_union contra_subsetD*

sup.boundedE)
qed
qed
}
with *I* **show** *Q l* (*postQ* ({*A*} *WHILE b DO C*) *s*) **by** *simp*
next
case *1*
have *g*: $\bigwedge e.\ e \circ S = (\%s.\ e\ (S\ s))$ **by** *auto*

have *lesvars ?upds′* ∩ *varacom C* = {}
using *yC blb* **by**(*auto*)

have *z*: $(fst \circ (\lambda(x,\ e,\ v).\ (x,\ \lambda s.\ e\ (S\ s),\ \bigcup x{\in}v.\ SS\ x))) = fst$ **by**(*auto*)
have *distinct* (*map fst ?upds′*)
using *Awhile*(*5*) *zny* **by** (*auto simp add*: *z*)

have *klae*: $\bigwedge s1\ s2\ A\ B.\ B \subseteq A \Longrightarrow s1 = s2\ on\ A \Longrightarrow s1 = s2\ on\ B$ **by** *auto*
from *Awhile*(*8*) *Awhile*(*9*) **have** *gl*: $\bigwedge a\ b\ c\ s1\ s2.\ (a,b,c) \in set\ upds \Longrightarrow s1 = s2\ on\ c \Longrightarrow b\ s1 = b\ s2$
by *fast*
have *CombALL*: $\vdash_1$ {$\lambda l\ s.\ pre\ C\ I\ l\ s \wedge preList\ ?upds'\ C\ l\ s$}
strip C
{ $time\ C \Downarrow \lambda l\ s.\ I\ l\ s \wedge postList\ ?upds'\ l\ s$} ∧
$(\forall l\ s.\ pre\ C\ I\ l\ s \longrightarrow I\ l\ (postQ\ C\ s)) \wedge K4\ ((SOME\ z.\ z \notin lesvars\ upds \cup varacom\ (\{A\}\ WHILE\ b\ DO\ C),\ E,\ Es)\ \#\ map\ (\lambda(x,\ e,\ v).\ (x,\ \lambda s.\ e\ (S\ s),\ \bigcup x{\in}v.\ SS\ x))\ upds)\ Is\ C\ I$
apply(*rule Awhile.IH*[**where** *upds=?upds′*])
apply (*fact*)+

subgoal apply *safe* **using** *Is* **apply** *blast*
using *Is* **apply** *blast* **done**
subgoal
using *Is Es* **apply** *auto*
apply(*simp_all add*: *postListpostSet2*, *safe*)
proof (*goal_cases*)
case (*1 l s1 s2 x e v*)
from *1*(*5*,*6*) **have** *i*: $l\ x = e\ (S\ s1)$ **by** *auto*
from *Awhile*(*10*) *1*(*6*) **have** *vLC*: $v \subseteq LVE$ **by** *auto*
have *st*: $(\bigcup y{\in}v.\ SS\ y) \subseteq (\bigcup y{\in}LVE.\ SS\ y)$ **using** *vLC* **by** *blast*
also have $\ldots \subseteq (\bigcup y{\in}LVE.\ funStar\ SS\ y)$ **using** *LVE_LV2* **by** *blast*

finally have *st*: $(\bigcup y{\in}v.\ SS\ y) \subseteq\ \ Es \cup (\bigcup y{\in}LVE.\ funStar\ SS\ y)$
by *blast*
have *ii*: $e\ (S\ s1) = e\ (S\ s2)$
apply(*rule gl*)
apply *fact*
apply(*rule Ssc*)
apply *fact*
using *st 1*(*3*) **by** *blast*
from *i ii* **show** *?case* **by** *simp*
next
case (*2 l s1 s2 x e v*)
from *2*(*5*,*6*) **have** *i*: $l\ x = e\ (S\ s2)$ **by** *auto*
from *Awhile*(*10*) *2*(*6*) **have** *vLC*: $v \subseteq LVE$ **by** *auto*
have *st*: $(\bigcup y{\in}v.\ SS\ y) \subseteq (\bigcup y{\in}LVE.\ SS\ y)$ **using** *vLC* **by** *blast*
also have $\ldots \subseteq (\bigcup y{\in}LVE.\ funStar\ SS\ y)$ **using** *LVE_LV2* **by**
blast
finally have *st*: $(\bigcup y{\in}v.\ SS\ y) \subseteq\ \ Es \cup (\bigcup y{\in}LVE.\ funStar\ SS\ y)$
by *blast*
have *ii*: $e\ (S\ s1) = e\ (S\ s2)$
apply(*rule gl*)
apply *fact*
apply(*rule Ssc*)
apply *fact*
using *st 2*(*3*) **by** *blast*
from *i ii* **show** *?case* **by** *simp*
qed **apply**(*auto*)
subgoal using *Es* **by** *auto*
subgoal apply(*rule gl*) **apply**(*simp*) **using** *Ss Awhile*(*10*) **by**
fastforce
subgoal using *Awhile*(*10*) *LVE_LV2* **by** *blast*
done
from *this*[*THEN conjunct2*, *THEN conjunct2*] **have**
K: *K C Is I* **and** *K3*: *K3 ?upds′ C I* **and** *Kt*: $\forall\, s1\ s2.\ s1 = s2\ on$
$kdeps\ C \longrightarrow time\ C\ s1 = time\ C\ s2$ **unfolding** *K4_def* **by** *auto*
show *K4 upds LVQ* ({*A*} *WHILE b DO C*) *Q*
unfolding *K4_def* **apply** *safe*
subgoal using *K* **unfolding** *K_def aha* **using** *Is* **by** *auto*
subgoal using *K3* **unfolding** *K3_def K2_def aha* **apply** *auto*
subgoal for *x e v* **apply** (*rule gl*) **apply** *simp* **apply**(*rule Ssc*)
using *Awhile*(*10*)
apply *fast* **apply** *blast* **done done**
subgoal using *Kt Es* **unfolding** *aha* **by** *auto*
done

show *?case*

apply(*simp add*: *aha*)
apply(*rule conseq_old*[**where** P=*?INV* **and** e'=E **and** Q=$\lambda l\ s.$ *?INV* $l\ s \wedge {}^{\sim}$ *bval b s*])
defer
proof (*goal_cases*)
case *3*
show *?case* **apply**(*rule exI*[**where** x=*1*]) **apply**(*auto*)[*1*] **apply**(*simp only*: *postList_preList*[*symmetric*]) **apply** (*auto*)[*1*]
by(*simp only*: *g*)
next
case *2*
show *?case*
proof (*safe*, *goal_cases*)
case (*1 l s*)
then show *?case* **using** *pre* **by** *auto*
next
case (*2 l s*)
from *Awhile*(*8*) **have** *Aw7*: $\bigwedge l\ s1\ s2.\ s1 = s2$ *on LVE* $\Longrightarrow$ *postList upds l s1* = *postList upds l s2* **by** *auto*
have *postList* (*map* ($\lambda(x, e, v).\ (x, \lambda s.\ e\ (S\ s), \bigcup x{\in}v.\ SS\ x)$) *upds*) $l\ s$ =
postList upds l ($S\ s$) **apply**(*induct upds*) **apply** *auto* **done**
also have ... = *postList upds l s* **using** *Aw7*[*of S s s l*] *pre2 2 LV_LV*
by *fast*
finally show *?case* **using** *2*(*3*) **by** *simp*
qed
next
case *1*
show *?case*
proof(*rule While*, *goal_cases*)
case *1*

note *Comb*=*CombALL*[*THEN conjunct1*]

show $\vdash_1 \{\lambda l\ s.\ (I\ l\ s \wedge$ *postList ?upds l s*$) \wedge$ *bval b s* $\wedge$ *preT C E s* $= l$ *?z*$\}$
strip C $\{$ *time C* $\Downarrow \lambda l\ s.\ (I\ l\ s \wedge$ *postList ?upds l s*$) \wedge E\ s \leq l$ *?z*$\}$
apply(*rule conseq_old*)
apply(*rule exI*[**where** x=*1*]) **apply**(*simp*)
prefer *2*

```
      proof (rule Comb, safe, goal_cases)
        case (2 l s)
        from IQ_in[OF 2(1)] gl Awhile(10,9)
        have y: postList ?upds l s =
              preList ?upds C l s (is ?IH upds)
        proof (induct upds)
          case (Cons a  upds')
         obtain y e v where axe: a = (y,e,v) using prod_cases3 by blast

          have IH: ?IH upds' apply(rule Cons(1))
            using Cons(2−5) by auto
          from Cons(3) axe have ke: ⋀s1 s2. s1 = s2 on v ⟹ e s1 = e
s2
            by fastforce
          have vLC: v ⊆ LVE using axe Cons(4) by simp
            have step: e (S s) = e (S (postQ C s)) apply(rule ke) using
Cons(2)  using vLC LV_LV 2(3)
            by blast
          show ?case unfolding axe using IH step apply(simp)
              apply(simp only: TQ) done
        qed simp
        from 2 show ?case by(simp add: y)
      qed  (auto simp: inv_impl)
    next
      show ∀ l s. bval b s ∧ I l s ∧ postList ?upds l s ⟶  1 + preT C E s
+ time C s ≤   E s
      proof (clarify, goal_cases)
        case (1 l s)
        thus ?case
          using 1 IQ by auto
      qed
    next
      show ∀ l s. ~bval b s ∧ I l s ∧ postList ?upds l s ⟶  1 ≤ E s
      proof (clarify, goal_cases)
        case (1 l s)
        with pre show ?case by auto
      qed
    next
      have pff: ?z ∉ lesvars ?upds apply(simp only: fua) by fact
      have support (λl s. I l s ∧ postList ?upds l s) ⊆ support I ∪ support
(postList ?upds)
        by(rule support_and)
      also
      have support (postList ?upds)
```

```
        ⊆ lesvars ?upds
          apply(rule support_postList) done
      finally
      have support (λl s. I l s ∧ postList ?upds l s) ⊆ support I ∪ lesvars
?upds
        by blast
      thus ?z ∉ support (λl s. I l s ∧ postList ?upds l s)
        apply(rule contra_subsetD)
        using zI pff by(simp)
    qed
  qed

 qed
qed


corollary vc_sound':
 assumes vc C Q Qset {}
        finite (support Q) finite (varacom C)
        ∀ l s. P l s ⟶ pre C Q l s
        ⋀s1 s2 l. s1 = s2 on Qset ⟹ Q l s1 = Q l s2
 shows ⊢₁ {P} strip C {time C ⇓ Q}
proof -
 show ?thesis
  apply(rule conseq_old)
   prefer 2 apply(rule vc_sound[where upds=[], OF assms(1), simpli-
fied, OF assms(2−3), THEN conjunct1])
  using assms(4,5) apply auto
  done
qed

corollary vc_sound'':
 assumes vc C Q Qset {}
        finite (support Q) finite (varacom C)
         (∃ k>0. ∀ l s. P l s ⟶ pre C Q l s ∧ time C s ≤ k ∗ e s)
        ⋀s1 s2 l. s1 = s2 on Qset ⟹ Q l s1 = Q l s2
 shows ⊢₁ {P} strip C {e ⇓ Q}
proof -
 show ?thesis
  apply(rule conseq_old)
   prefer 2 apply(rule vc_sound[where upds=[], OF assms(1), simpli-
fied, OF assms(2−3), THEN conjunct1])
  using assms(4,5) apply auto
  done
```

qed

end
theory *Nielson_VCGi_complete*
imports *Nielson_VCG Nielson_VCGi*
begin

4.8.3 Completeness

As the improved VCG for the Nielson logic is only more liberal in the sense that the S annotation is only checked for "interesting" variables, if we specify the set of interesting variables to be all variables we basically get the same verification conditions as for the normal VCG. In that sense, we can prove the completeness of the improved VCG with the completeness theorem of the normal VCG.

For that, we formulate some translation functions and in the end show completeness of the improved VCG:

fun *transl* :: *Nielson_VCG.acom* ⇒ *Nielson_VCGi.acom* **where**
transl SKIP = *SKIP* |
transl (x ::= a) = (x ::= a) |
transl (C_1;; C_2) = (*transl* C_1;; *transl* C_2) |
transl (*IF b THEN* C_1 *ELSE* C_2) = (*IF b THEN transl* C_1 *ELSE transl* C_2) |
transl ({*A*/*B*/*D*} *CONSEQ C*) = ({(*A*,*UNIV*)/(*B*,*UNIV*)/(*D*,*UNIV*)} *CONSEQ transl C*) |
transl ({(*I*,*S*,*E*)} *WHILE b DO C*) = ({((*I*,*UNIV*),*S*,*E*,*UNIV*,(λ*v*. *UNIV*))} *WHILE b DO transl C*)

lemma *qdeps_UNIV*: *qdeps* (*transl C*) *UNIV* = *UNIV*
apply(*induct C*) **apply** *auto* **done**

lemma *fune_UNIV*: *fune* (*transl C*) *UNIV* = *UNIV*
apply(*induct C*) **apply** *auto* **done**

lemma *pre_transl*: *Nielson_VCGi.pre* (*transl C*) *Q* = *Nielson_VCG.pre C Q*
apply(*induct C arbitrary*: *Q*) **by** (*auto*)

lemma *preT_transl*: *Nielson_VCGi.preT* (*transl C*) *E* = *Nielson_VCG.preT C E*
apply(*induct C arbitrary*: *E*) **by** (*auto*)

lemma *postQ_transl*: *Nielson_VCGi.postQ* (*transl C*) = *Nielson_VCG.postQ C*

apply(*induct C*) **by** (*auto*)

lemma *time_transl*: *Nielson_VCGi.time* (*transl C*) = *Nielson_VCG.time C*
apply(*induct C*) **by**(*auto simp*: *preT_transl*)

lemma *vc_transl*: *Nielson_VCG.vc C Q* $\Longrightarrow$ *Nielson_VCGi.vc* (*transl C*) *Q UNIV UNIV*
proof (*induct C arbitrary*: *Q*)
next
case (*Aconseq x1 x2 x3 C*)
then show *?case* **apply** (*auto simp*: *pre_transl time_transl*) **apply** *presburger+* **done**
next
case (*Awhile A b C*)
obtain *I S E* **where** *A*=(*I*,*S*,*E*) **using** *prod_cases3* **by** *blast*
with *Awhile* **show** *?case* **apply** (*auto simp*: *pre_transl preT_transl time_transl postQ_transl*) **apply** *presburger+* **done**
qed (*auto simp*: *qdeps_UNIV fune_UNIV pre_transl*)

lemma *strip_transl*: *Nielson_VCGi.strip* (*transl C*) = *Nielson_VCG.strip C*
by (*induct C*, *auto*)

lemma *vc_restrict_complete*:
assumes $\vdash_1 \{P\}\ c\ \{\ e \Downarrow Q\}$
shows $\exists$ *C*. *Nielson_VCGi.strip C* = *c* $\wedge$ *Nielson_VCGi.vc C Q UNIV UNIV*
$\wedge$ ($\forall$ *l s*. *P l s* $\longrightarrow$ *Nielson_VCGi.pre C Q l s* $\wedge$ *Q l* (*Nielson_VCGi.postQ C s*))
$\wedge$ ($\exists$ *k*. $\forall$ *l s*. *P l s* $\longrightarrow$ *Nielson_VCGi.time C s* $\leq$ *k* $*$ *e s*)
(**is** $\exists$ *C*. *?G P c Q C e*)
proof −
obtain *C* **where** *C*: *Nielson_VCG.strip C* = *c* *Nielson_VCG.vc C Q* ($\forall$ *l s*. *P l s* $\longrightarrow$ *Nielson_VCG.pre C Q l s* $\wedge$ *Q l* (*Nielson_VCG.postQ C s*))
($\exists$ *k*. $\forall$ *l s*. *P l s* $\longrightarrow$ *Nielson_VCG.time C s* $\leq$ *k* $*$ *e s*) **using** *vc_complete*[*OF assms*] **by** *blast*
let *?C*=*transl C*
from *C* **have** *?G P c Q ?C e*
by(*auto simp*: *strip_transl vc_transl pre_transl postQ_transl time_transl*)
then show *?thesis* **..**

qed

end
theory *Nielson_Examples*
imports *Nielson_VCG*
begin

4.8.4 example

lemma $\vdash_1$ {%*l s. True*} *SKIP*;; *SKIP* { %*s. 1* $\Downarrow$ %*l s. True*}
proof –
let *?T* = %*l s. True*
have $\vdash_1$ {%*l s. True*} *strip* (*Aconseq ?T ?T* (%*s. 1*) (*Aseq Askip Askip*)) { %*s. 1* $\Downarrow$ %*l s. True*}
apply(*rule vc_sound''*) **by** *auto*
then show *?thesis* **by** *simp*
qed

lemma *finite* (*support P*) $\Longrightarrow$ $\vdash_1$ {*P*} *strip Askip* {*time Askip* $\Downarrow$ *P*}
apply(*rule vc_sound'*)
apply(*simp*)
apply(*simp*)
apply(*simp*)
apply(*simp*) **done**

lemma *support_single2*: *support* (λ*l s. P s*) = {}
unfolding *support_def* **by** *fastforce*

lemma $\vdash_1$ { %*l s. True* } *strip* (*Aassign a* (*N 1*)) {*time* (*Aassign a* (*N 1*)) $\Downarrow$ %*l s. s a* = *1*}
apply(*rule vc_sound'*)
apply(*simp_all add*: *support_single2*) **done**

lemma $\vdash_1$ { %*l s. True* } *strip* ((*a* ::= (*N 1*)) ;; *Askip*) {*time* ((*a* ::= (*N 1*)) ;; *Askip*) $\Downarrow$ %*l s. s a* = *1*}
apply(*rule vc_sound'*)
apply(*simp_all add*: *support_single2*) **done**

lemma $\vdash_1$ { %*l s. True* } *strip* ((*a* ::= (*N 1*)) ;; *b* ::= (*V a*)) {*time* ((*a*

::= (N 1)) ;; b ::= (V a)) ⇓ %l s. s b = 1}
  apply(*rule vc_sound′*)
  by(*simp_all add*: *support_single2*)

lemma assumes
  E: *E* = (%*s*. *1* + *2* ∗ (*4* − *nat* (*s a*))) **and**
  C: *C* = ({(*I*,(*S*,(*E*)))} *WHILE Less* (*V a*) (*N 3*) *DO a* ::= *Plus* (*V a*) (*N 1*))
shows ⋀*s*. *0* ≤ *s a* ⟹ *time C s* ≤ *9*
  unfolding *C E* **apply**(*simp*) **done**

Count up to 3 **lemma** *example_count_upto_3*: **assumes**
  I: *I* = (%*l s*. *s a* ≥ *0*) **and**
  E: *E* = (%*s*. *1* + *2* ∗ (*4* − *nat* (*s a*))) **and**
  S: *S* = (%*s*. (*if s a* ≥ *3 then s else s*(*a*:=*3*))) **and**
  C: *C* = ({(*I*,(*S*,(*E*)))} *WHILE Less* (*V a*) (*N 3*) *DO a* ::= *Plus* (*V a*) (*N 1*))
shows $\vdash_1$ { %*l s*. *0* ≤ *s a* } *strip C* {*time C* ⇓ %*l s*. *True* }
  unfolding *C*
  apply(*rule vc_sound′*)
  subgoal
    apply(*simp*)
    apply(*safe*)
    subgoal unfolding *I* **by** *simp*
    subgoal unfolding *I E* **by** *simp*
    subgoal unfolding *S* **by** *auto*
    subgoal unfolding *I E* **by** *auto*
    subgoal unfolding *I S* **by** *auto*
    done
  subgoal
    by *simp*
  subgoal
    unfolding *I* **by**(*simp add*: *support_inv*)
  subgoal unfolding *I* **by** *simp*
  done

Count up to b **lemma** *example_count_upto_b*: **assumes**
  I: *I* = (%*l s*. *s a* ≥ *0*) **and**
  E: *E* = (%*s*. *1* + *2* ∗ ((*nat b*+*1*) − *nat* (*s a*))) **and**
  S: *S* = (%*s*. (*if s a* ≥ *b then s else s*(*a*:=*b*))) **and**
  C: *C* = ({(*I*,(*S*,(*E*)))} *WHILE Less* (*V a*) (*N b*) *DO a* ::= *Plus* (*V a*) (*N 1*))
shows $\vdash_1$ { %*l s*. *0* ≤ *s a* } *strip C* {*time C* ⇓ %*l s*. *True* }

unfolding *C*
apply(*rule vc_sound'*) **by**(*auto simp*: *I E S support_inv*)

Example: multiplication by repeated addition **lemma** *helper*: $(A{::}int) * B + B = (A+1) * B$ **by**(*auto simp*: *distrib_right*)

lemma *mult*: **assumes**
I: $I = (\%l\ s.\ s\ ''a'' \geq 0 \wedge s\ ''a'' \geq s\ ''z'' \wedge s\ ''z'' \geq 0 \ \wedge s\ ''y'' = \ s\ ''z'' * (s\ ''b'')\ )$ **and**
E: $E = (\%s.\ 1 + 3 * ((nat\ (s\ ''a'') + 1) - nat\ (s\ ''z'')))$ **and**
S: $S = (\%s.\ (if\ s\ ''z'' \geq s\ ''a''\ then\ s\ else\ s(''y'':=(s\ ''a'') * (s\ ''b''),\ ''z'':=s\ ''a''\ )\ ))$ **and**
C: $C = (''y'' ::= (N\ 0);;\ ''z'' ::= (N\ 0)\ ;;\ \{(I,(S,(E)))\}\ \ WHILE\ Less\ (V\ ''z'')\ (V\ ''a'')\ DO\ (''y'' ::= Plus\ (V\ ''y'')\ (V\ ''b'')\ ;;\ ''z'' ::= Plus\ (V\ ''z'')\ (N\ 1))\ )$ **and**
f: $f = (\%s.\ 3 * (nat(s\ ''a'') + 2))$
shows $\vdash_1 \{\ \%l\ s.\ 0 \leq s\ ''a''\ \}\ strip\ C\ \{\ f \Downarrow \%l\ s.\ s\ ''y'' = s\ ''a'' * (s\ ''b'')\ \}$
unfolding *C*
apply(*rule vc_sound''*)
apply(*auto simp*: *I E S distrib_right support_inv f*)
subgoal for *s* **by** (*auto simp add*: *helper*)
done

lemma *mult_abstract*: **assumes**
I: $I = (\%l\ s.\ s\ ''a'' \geq 0 \wedge s\ ''a'' \geq s\ ''z'' \wedge s\ ''z'' \geq 0 \ \wedge s\ ''y'' = \ s\ ''z'' * (s\ ''b'')\ )$ **and**
E: $E = (\%s.\ 1 + 2* ((nat\ (s\ ''a'') + 1) - nat\ (s\ ''z'')))$ **and**
S: $S = (\%s.\ (if\ s\ ''z'' \geq s\ ''a''\ then\ s\ else\ s(''y'':=(s\ ''a'') * (s\ ''b''),\ ''z'':=s\ ''a''\ )\ ))$ **and**
e: $e = (\%s.\ 1)$ **and**
$lb[simp]$: $(lb{::}acom) = (\{\lambda l\ s.\ I\ l\ s \wedge s\ ''z'' < s\ ''a''\ \ /I/e\}\ CONSEQ\ (''y'' ::= Plus\ (V\ ''y'')\ (V\ ''b'')\ ;;\ ''z'' ::= Plus\ (V\ ''z'')\ (N\ 1))\ )$ **and**
$l[simp]$: $(l{::}acom) = \{(I,(S,(E)))\}\ \ WHILE\ \ (Less\ (V\ ''z'')\ (V\ ''a''))\ DO\ lb$ **and**
$e'[simp]$: $e' = (\%s.\ 1 + (nat\ (s\ ''a'')))$ **and**
$wl[simp]$: $(wl{::}acom) = \{I/\lambda l\ s.\ I\ l\ s \wedge s\ ''z'' \geq s\ ''a''/e'\}\ CONSEQ\ l$ **and**
C: $(C{::}acom) = (''y'' ::= (N\ 0);;\ ''z'' ::= (N\ 0)\ ;;\ wl\ )$ **and**
f: $f = (\%s.\ nat(s\ ''a'') + 1)$
shows $\vdash_1 \{\ \%l\ s.\ 0 \leq s\ ''a''\ \}\ strip\ (\ \ \{\%l\ s.\ 0 \leq s\ ''a''/\ \%l\ s.\ s\ ''y'' = s\ ''a'' * (s\ ''b'')/\ f\}\ CONSEQ\ C)\ \{\ f \Downarrow \%l\ s.\ s\ ''y'' = s\ ''a'' * (s\ ''b'')\ \}$

unfolding C
apply(*rule vc_sound''*)
apply(*auto simp*: *I E S distrib_right support_inv f e*)
subgoal for s **by** (*auto simp add*: *helper*)
apply(*rule exI*[**where** *x=100*]) **apply** *auto*
apply(*rule exI*[**where** *x=100*]) **apply** *auto*
done

Example: nested loops **lemma** *nested*: **assumes**
I2: $I2 = (\%l\ s.\ s\ ''a'' \geq 0 \wedge s\ ''b'' \geq 0 \wedge s\ ''a'' > s\ ''z'' \wedge s\ ''z'' \geq 0 \wedge s\ ''b'' \geq s\ ''g'' \wedge s\ ''g'' \geq 0 \wedge s\ ''y'' = (s\ ''z'') * (s\ ''b'') + s\ ''g'')$
and

I1: $I1 = (\%l\ s.\ s\ ''a'' \geq 0 \wedge s\ ''b'' \geq 0 \wedge s\ ''a'' \geq s\ ''z'' \wedge s\ ''z'' \geq 0 \wedge s\ ''y'' = s\ ''z'' * (s\ ''b''))$ **and**

E2: $E2 = (\%s.\ 1 + 3 * ((nat\ (s\ ''b'')) - nat\ (s\ ''g'')))$ **and**
S2: $S2 = (\%s.\ (if\ s\ ''g'' \geq s\ ''b''\ then\ s\ else\ s(''y'':=(s\ ''z'') * (s\ ''b'') + s\ ''b'',\ ''g'':=s\ ''b'')))$ **and**

E1: $E1 = (\%s.\ 1 + (4 + (3 * ((nat\ (s\ ''b''))))) * ((nat\ (s\ ''a'')) - nat\ (s\ ''z'')))$ **and**
S1: $S1 = (\%s.\ (if\ s\ ''z'' \geq s\ ''a''\ then\ s\ else\ s(''y'':=(s\ ''a'') * (s\ ''b''),\ ''z'':=s\ ''a'',\ ''g'':=s\ ''b'')))$ **and**
C: $C = (''y'' ::= (N\ 0);;$
$''z'' ::= (N\ 0)\ ;;$
$\{(I1,(S1,(E1)))\}$ *WHILE Less* $(V\ ''z'')\ (V\ ''a'')$ *DO*
(
$''g'' ::= (N\ 0)\ ;;$
(
$\{(I2,(S2,(E2)))\}$ *WHILE Less* $(V\ ''g'')\ (V\ ''b'')$ *DO*
$(''y'' ::= Plus\ (V\ ''y'')\ (N\ 1);;$
$''g'' ::= Plus\ (V\ ''g'')\ (N\ 1))$

) ;;
$''z'' ::= Plus\ (V\ ''z'')\ (N\ 1))$
) **and**
f: $f = (\%s.\ 3 + 4*nat(s\ ''a'') + 3 * (nat(s\ ''a'') * nat(s\ ''b'')))$
shows $\vdash_1 \{\ \%l\ s.\ 0 \leq s\ ''a'' \wedge s\ ''b'' \geq 0\ \}\ strip\ C\ \{\ f \Downarrow \%l\ s.\ s\ ''y'' = s\ ''a'' * (s\ ''b'')\ \}$
unfolding C
apply(*rule vc_sound''*)
proof(*goal_cases*)

case *1*
show *?case* **apply**(*simp*)
proof(*safe*, *goal_cases*)
case (*1 l s*)
from *1* **show** *?case* **unfolding** *I1* **unfolding** *I2* **by**(*auto*)
next
case (*2 l s*)
then show *?case* **unfolding** *I1 S2* **apply**(*auto*) **unfolding** *E2* **apply**(*simp*) **unfolding** *E2* **apply**(*auto*)
unfolding *E1* **apply**(*simp*)

apply(*simp*)
proof (*goal_cases*)
case *1*
then have *g*: $s\ ''a'' > s\ ''z''$ **by** *linarith*
then have *p*: $(nat\ (s\ ''a'') - nat\ (s\ ''z'' + 1)) = (nat\ (s\ ''a'') - nat\ (s\ ''z'')) - 1$ **and** *z*: $(nat\ (s\ ''a'') - nat\ (s\ ''z'')) \geq 1$
using *1* **apply** *linarith*
using *g 1* **by** *linarith*

have $Suc\ (Suc\ (Suc\ (Suc\ ((4 + 3 * nat\ (s\ ''b'')) * (nat\ (s\ ''a'') - nat\ (s\ ''z'' + 1)) + 3 * nat\ (s\ ''b''))))) =$
$4 + ((4 + 3 * nat\ (s\ ''b'')) * (nat\ (s\ ''a'') - nat\ (s\ ''z'' + 1)) + 3 * nat\ (s\ ''b''))$
by *auto*
also
have $\ldots = 4 + (\ (4 + 3 * nat\ (s\ ''b'')) * (nat\ (s\ ''a'') - nat\ (s\ ''z'')) - (4 + 3 * nat\ (s\ ''b''))\ \ + 3 * nat\ (s\ ''b''))$
apply(*simp only*: *p*)
proof −
have $\bigwedge n\ na.\ (n{::}nat) * na - n = n * (na - 1)$
by (*simp add*: *diff_mult_distrib2*)
then show $4 + ((4 + 3 * nat\ (s\ ''b'')) * (nat\ (s\ ''a'') - nat\ (s\ ''z'') - 1) + 3 * nat\ (s\ ''b'')) = 4 + ((4 + 3 * nat\ (s\ ''b'')) * (nat\ (s\ ''a'') - nat\ (s\ ''z'')) - (4 + 3 * nat\ (s\ ''b'')) + 3 * nat\ (s\ ''b''))$
by *presburger*
qed
also
have $\ldots = (4 + 3 * nat\ (s\ ''b'')) * (nat\ (s\ ''a'') - nat\ (s\ ''z''))$
using *z*
by (*smt* ‹$4 + ((4 + 3 * nat\ (s\ ''b'')) * (nat\ (s\ ''a'') - nat\ (s\ ''z'' + 1)) + 3 * nat\ (s\ ''b'')) = 4 + ((4 + 3 * nat\ (s\ ''b'')) * (nat\ (s\ ''a'') - nat\ (s\ ''z'')) - (4 + 3 * nat\ (s\ ''b'')) + 3 * nat\ (s\ ''b''))$› *add.left_commute diff_add distrib_left mult.right_neutral p*)

```
    finally show ?case by simp
  qed

next
  case (3 l s)
  { fix i :: int assume 0 ≤ i then have int (∑{0..<nat i}) + i = int
(∑{0..nat i})
      by (simp add: sum.last_plus)    } note bla=this
  from 3 show ?case unfolding I1 S1 S2  apply(auto simp add:  )
  proof (goal_cases)
    case 1
    then have a: s ''a'' =  s ''z'' + 1 by linarith
    show ?case apply(simp only: a) using 1
      by (simp add: distrib_left mult.commute fun_upd_twist)
  qed
next
  case (4 l s)
  then show ?case unfolding I1  by (auto)
next
  case (5 l s)
  then show ?case unfolding I1 E1 by auto
next
  case (6 l s)
  then show ?case unfolding I1 S1 by(simp)


next
  case (7 l s)
  then show ?case unfolding I2 apply(simp) done
next
  case (8 l s)
  then show ?case unfolding I2   apply(auto) unfolding  E2  by(auto)

next
  case (9 l s)
  { fix i :: int assume 0 ≤ i then have int (∑{0..<nat i}) + i = int
(∑{0..nat i})
      by (simp add: sum.last_plus)    } note bla=this
  from 9 show ?case unfolding I2   S2  apply(auto simp add:  ) done

next
  case (10 l s)
   then show ?case unfolding I2 I1 by (auto simp add: distrib_left
```

mult.commute)
  next
    case (*11 l s*)
    then show *?case* **unfolding** *I2 E2* **by** (*simp*)
  next
    case (*12 l s*)
    then show *?case* **unfolding** *I2 S2* **by**(*simp*)
  qed
next
  case *2*
  show *?case* **apply** (*rule exI*[**where** *x=1*]) **by** (*auto simp add*: *I1 C E1 f distrib_left mult.commute*)
qed (*auto simp*: *I1 I2 support_single2*)

with logical variables **lemma** *fin_sup_single*: *finite* (*support* ($\lambda l.$ *P* (*l a*)))
  apply(*rule finite_subset*[*OF support_single*]) **by** *simp*

lemmas *fin_support* = *fin_sup_single*

lemma *finite_support_and*: *finite* (*support A*) $\Longrightarrow$ *finite* (*support B*) $\Longrightarrow$ *finite* (*support* ($\lambda l\ s.\ A\ l\ s \wedge B\ l\ s$))
  apply(*rule finite_subset*[*OF support_and*]) **by** *blast*

end
theory *Nielson_Sqrt*
imports *Nielson_VCGi HOL−Library.Discrete_Functions*
begin

4.9 Example: discrete square root in Nielson's logic

As an example, consider the following program that computes the discrete square root:

definition *c* :: *com* **where** *c*=
```
    ''l''::= N 0 ;;
    ''m'' ::= N 0 ;;
    ''r''::= Plus (N 1) (V ''x'');;
    (WHILE (Less (Plus (N 1) (V ''l'')) (V ''r''))
        DO (''m'' ::= (Div (Plus (V ''l'') (V ''r'')) (N 2)) ;;
          (IF Not (Less (Times (V ''m'') (V ''m'')) (V ''x''))
            THEN ''l'' ::= V ''m''
            ELSE ''r'' ::= V ''m'');;
```

$''m'' ::= N\ 0))$

In this theory we will show that its running time is in the order of magnitude of the logarithm of the variable "x"

a little lemma we need later for bounding the running time:

lemma *absch*: $\bigwedge s\ k.\ 1 + s\ ''x'' = 2\ \hat{}\ k \Longrightarrow 5 * k \leq 96 + 100 * floor\_log$ $(nat\ (s\ ''x''))$
proof −
  fix *s* :: *state* **and** *k* :: *nat*
  assume *F*: $1 + s\ ''x'' = 2\ \hat{}\ k$
  then have *i*: $nat\ (1 + s\ ''x'') = 2\ \hat{}\ k$ **and** *nn*: $s\ ''x'' \geq 0$ **apply** (*auto simp*: *nat_power_eq*)
    by (*smt one_le_power*)
  have *F*: $1 + nat\ (s\ ''x'') = 2\ \hat{}k$ **unfolding** *i*[*symmetric*] **using** *nn* **by** *auto*
  show $5 * k \leq 96 + 100 * floor\_log\ (nat\ (s\ ''x''))$
  proof (*cases* $s\ ''x'' \geq 1$)
    case *True*
    have $5 * k = 5 * (floor\_log\ (2\hat{}k))$ **by** *auto*
    also have $\ldots = 5 * floor\_log\ (1 + nat\ (s\ ''x''))$ **by**(*simp only*: *F*[*symmetric*])
    also have $\ldots \leq 5 * floor\_log\ (nat\ (s\ ''x'' + s\ ''x''))$ **using** *True*
      apply *auto* **apply**(*rule monoD*[*OF floor_log_mono*]) **by** *auto*
    also have $\ldots = 5 * floor\_log\ (2 * nat\ (s\ ''x''))$ **by** (*auto simp*: *nat_mult_distrib*)
    also have $\ldots = 5 + 5 * (floor\_log\ (nat\ (s\ ''x'')))$ **using** *True* **by** *auto*
    also have $\ldots \leq 96 + 100 * floor\_log\ (nat\ (s\ ''x''))$ **by** *simp*
    finally show *?thesis* **.**
  next
    case *False*
    with *nn* **have** *gt1*: $s\ ''x'' = 0$ **by** *auto*
    from *F*[*unfolded gt1*] **have** $2\ \hat{}\ k = (1{::}int)$ **using** *floor_log_Suc_zero* **by** *auto*
    then have $k{=}0$
    by (*metis One_nat_def add.right_neutral gt1 i n_not_Suc_n nat_numeral nat_power_eq_Suc_0_iff numeral_2_eq_2 numeral_One*)
    then show *?thesis* **by**(*simp add*: *gt1*)
  qed
qed

For simplicity we assume, that during the process all segments between "l" and "r" have as length a power of two. This simplifies the analysis. To obtain this we choose the precondition P accordingly.

Now lets show the correctness of our time complexity: the binary search is in O(log "x")

lemma
assumes *P*: $P = (\lambda l\ s.\ (\exists k.\ 1 + s\ ''x'' = 2\ \hat{}\ k))$
and $e : e = (\lambda s.\ \mathit{floor\_log}\ (\mathit{nat}\ (s\ ''x'')) + 1)$ **and**
Q[*simp*]: $Q = (\lambda l\ s.\ \mathit{True})$
shows $\vdash_1 \{P\}\ c\ \{\ e \Downarrow Q\}$
proof –

— first we create an annotated command
let *?lb* = ''m'' ::=
(*Div* (*Plus* (*V* ''l'') (*V* ''r'')) (*N 2*)) ;;
(*IF Not* (*Less* (*Times* (*V* ''m'') (*V* ''m'')) (*V* ''x''))
THEN ''l'' ::= *V* ''m''
ELSE ''r'' ::= *V* ''m'');;
(''m'' ::= *N 0*)::*acom*
— with an Invariant
define *I* :: *assn2* **where** $I \equiv (\lambda l\ s.\ (\exists k.\ s\ ''r'' - s\ ''l'' = 2\ \hat{}\ k) \wedge s\ ''l'' \geq 0)$

— and an time bound annotation for the loop
define *E* :: *tbd* **where** *E* ≡ %*s*. *1* + *5* ∗ *floor_log* (*nat*(*s* ''r'' − *s* ''l''))
define *S* :: *state* ⇒ *state* **where** *S* ≡ %*s*. *s*
define *Es* :: *vname* ⇒ *vname set* **where** *Es* = (%*x*. {*x*})

define *R* :: (*assn2*∗(*vname set*))∗((*state*⇒*state*)∗(*tbd*∗((*vname set*∗(*vname* ⇒ *vname set*)))))
where *R*=((*I*,{''l'',''r''}),(*S*,(*E*,({''l'',''r''},*Es*))))

let *?C* = ''l''::= *N 0* ;; (''m'' ::= *N 0*) ;; ''r''::= *Plus* (*N 1*) (*V* ''x'');; ({*R*} *WHILE* (*Less* (*Plus* (*N 1*) (*V* ''l'')) (*V* ''r'')) *DO ?lb*)

— we show that the annotated command corresponds to the command we are interested in
have *s*: *strip ?C* = *c* **unfolding** *c_def* **by** *auto*

— now we show that the annotated command is correct; here we use the improved VCG and the Nielson
have *v*: $\vdash_1$ {*P*} *strip ?C* {*e* ⇓ *Q*}
proof (*rule vc_sound''*, *safe*)

— A) first lets show the verification conditions:
show *vc ?C Q* {} {} **unfolding** *R_def* **apply** (*simp only*: *vc.simps*) **apply** *auto*
subgoal unfolding *I_def* **by** *auto*
subgoal unfolding *I_def* **by** *auto*

subgoal unfolding *E_def* **by** *auto*
proof (*goal_cases*)
fix *s::state* **and** *l*
assume *I*: *I l s* **and** *2*: $1 + s\ ''l'' < s\ ''r''$
from *I* **obtain** *k :: nat* **where** *3*: $s\ ''r'' - s\ ''l'' = 2 \,\widehat{}\, k$ **and** *4*: $s\ ''l'' \geq 0$ **unfolding** *I_def* **by** *blast*
from *3 2* **have** $k>0$ **using** *gr0I* **by** *force*
then obtain k' **where** k': $k=k'+1$ **by** (*metis Suc_eq_plus1 Suc_pred*)

from *3* k' **have** *R1*: $s\ ''r'' - (s\ ''l'' + s\ ''r'')\ div\ 2 = 2 \,\widehat{}\, k'$ **and**
R2: $(s\ ''l'' + s\ ''r'')\ div\ 2 - s\ ''l'' = 2 \,\widehat{}\, k'$ **by** *auto*
then have *E1*: $\exists k.\ s\ ''r'' - (s\ ''l'' + s\ ''r'')\ div\ 2 = 2 \,\widehat{}\, k$ **and**
E2: $\exists k.\ (s\ ''l'' + s\ ''r'')\ div\ 2 - s\ ''l'' = 2 \,\widehat{}\, k$ **by** *auto*
then show $I\ l\ (s(''l'' := (s\ ''l'' + s\ ''r'')\ div\ 2,\ ''m'' := 0))$ **and**
$I\ l\ (s(''r'' := (s\ ''l'' + s\ ''r'')\ div\ 2,\ ''m'' := 0))$ **using** *2 4* **unfolding** *I_def* **by** *auto*

show $Suc\ (Suc\ (Suc\ (Suc\ (Suc\ (E\ (s(''l'' := (s\ ''l'' + s\ ''r'')\ div\ 2,\ ''m'' := 0))))))) \leq E\ s$
unfolding *E_def* **apply** *simp* **unfolding** *R1 3* k' **by** (*auto simp*: *nat_power_eq nat_mult_distrib*)
show $Suc\ (Suc\ (Suc\ (Suc\ (Suc\ (E\ (s(''r'' := (s\ ''l'' + s\ ''r'')\ div\ 2,\ ''m'' := 0))))))) \leq E\ s$
unfolding *E_def* **apply** *simp* **unfolding** *R2 3* k' **by** (*auto simp*: *nat_power_eq nat_mult_distrib*)
next
fix *l s*
show $Suc\ 0 \leq E\ s$ **unfolding** *E_def* **by** *auto*
show $Suc\ 0 \leq E\ s$ **unfolding** *E_def* **by** *auto*
qed
next
— B) lets show that the precondition implies the weakest precondition, and that the time bound of C can be bounded by log "x"
fix *s*
show $(\exists k>0.\ \forall l\ s.\ P\ l\ s \longrightarrow pre\ ?C\ Q\ l\ s \wedge time\ ?C\ s \leq k * e\ s)$
apply(*rule exI*[**where** $x=100$])
unfolding *P R_def I_def E_def e* **by** (*auto simp*: *nat_power_eq absch*)
qed
— last side conditions are proven automatically
(*auto simp*: *Q support_inv R_def I_def*)

— now we conclude with the correctness of the Hoare triple involving the

time bound
  from *s v* **show** *?thesis* **by** *simp*
qed

end

5 Quantitative Hoare Logic (due to Carbonneaux)

theory *Quant_Hoare*
imports *Big_StepT Complex_Main HOL−Library.Extended_Nat*
begin

abbreviation *eq a b* == (*And* (*Not* (*Less a b*)) (*Not* (*Less b a*)))

type_synonym *lvname* = *string*
type_synonym *assn* = *state* ⇒ *bool*
type_synonym *qassn* = *state* ⇒ *enat*

The support of an assn2

abbreviation *state_subst* :: *state* ⇒ *aexp* ⇒ *vname* ⇒ *state*
  (‹_[_'/_]› [*1000*,*0*,*0*] *999*)
where *s*[*a*/*x*] == *s*(*x* := *aval a s*)

fun *emb* :: *bool* ⇒ *enat* (‹↑›) **where**
  emb False = ∞
| *emb True* = *0*

5.1 Validity of quantitative Hoare Triple

definition *hoare2_valid* :: *qassn* ⇒ *com* ⇒ *qassn* ⇒ *bool*
  (‹$\models_2$ {(*1*_)}/ (_)/ {(*1*_)}› *50*) **where**
$\models_2$ {*P*} *c* {*Q*} ⟷ (∀ *s*. *P s* < ∞ ⟶ (∃ *t p*. ((*c*,*s*) ⇒ *p* ⇓ *t*) ∧ *P s* ≥ *p* + *Q t*))

5.2 Hoare logic for quantiative reasoning

inductive
  hoare2 :: *qassn* ⇒ *com* ⇒ *qassn* ⇒ *bool* (‹$\vdash_2$ ({(*1*_)}/ (_)/ {(*1*_)})› *50*)

where

Skip: $\vdash_2 \{\%s.\ eSuc\ (P\ s)\}\ SKIP\ \{P\}$ |

Assign: $\vdash_2 \{\lambda s.\ eSuc\ (P\ (s[a/x]))\}\ x{::=}a\ \{\ P\ \}$ |

If: $[\![\vdash_2 \{\lambda s.\ P\ s + \uparrow(\ bval\ b\ s)\}\ c_1\ \{\ Q\};$
$\quad\vdash_2 \{\lambda s.\ P\ s + \uparrow(\neg\ bval\ b\ s)\}\ c_2\ \{\ Q\}\ ]\!]$
$\Longrightarrow \vdash_2 \{\lambda s.\ eSuc\ (P\ s)\}\ IF\ b\ THEN\ c_1\ ELSE\ c_2\ \{\ Q\ \}$ |

Seq: $[\![\vdash_2 \{\ P_1\ \}\ c_1\ \{\ P_2\ \}; \vdash_2 \{P_2\}\ c_2\ \{\ P_3\ \}]\!] \Longrightarrow \vdash_2 \{P_1\}\ c_1;;c_2\ \{P_3\}$ |

While:
$[\![\ \ \vdash_2 \{\ \%s.\ I\ s + \uparrow(bval\ b\ s)\ \}\ c\ \{\ \%t.\ I\ t + 1\ \}\ \ ]\!]$
$\Longrightarrow \vdash_2 \{\lambda s.\ I\ s + 1\ \}\ WHILE\ b\ DO\ c\ \{\lambda s.\ \ I\ s + \uparrow(\neg\ bval\ b\ s)\ \}$ |

conseq: $[\![\vdash_2 \{P\}c\{Q\}\ ;\ \bigwedge s.\ P\ s \le P'\ s\ ;\ \bigwedge s.\ Q'\ s \le Q\ s\ ]\!] \Longrightarrow$
$\vdash_2 \{P'\}c\{\ Q'\}$

derived rules

lemma *strengthen_pre*: $[\![\forall s.\ P\ s \le P'\ s;\ \vdash_2 \{P\}\ c\ \{Q\}\ ]\!] \Longrightarrow \vdash_2 \{P'\}\ c\ \{Q\}$
using *conseq* **by** *blast*

lemma *weaken_post*: $[\![\vdash_2 \{P\}\ c\ \{Q\};\ \ \forall s.\ Q\ s \ge Q'\ s\ ]\!] \Longrightarrow\ \vdash_2 \{P\}\ c\ \{Q'\}$
using *conseq* **by** *blast*

lemma *Assign'*: $\forall s.\ P\ s \ge eSuc\ (\ Q(s[a/x])) \Longrightarrow \vdash_2 \{P\}\ x ::= a\ \{Q\}$
by (*simp add*: *strengthen_pre*[*OF _ Assign*])

lemma *progress*: $(c,\ s) \Rightarrow p \Downarrow t \Longrightarrow p > 0$
by (*induct rule*: *big_step_t.induct*, *auto*)

lemma *FalseImplies*: $\vdash_2 \{\%s.\ \infty\}\ c\ \{\ Q\}$
apply (*induction c arbitrary*: *Q*)
apply(*auto intro*: *hoare2.Skip hoare2.Assign hoare2.Seq hoare2.conseq*)
subgoal apply(*rule hoare2.conseq*) **apply**(*rule hoare2.If*[**where** *P*=%*s*. ∞]) **by**(*auto intro*: *hoare2.If hoare2.conseq*)
subgoal apply(*rule hoare2.conseq*) **apply**(*rule hoare2.While*[**where** *I*=%*s*. ∞]) **apply**(*rule hoare2.conseq*) **by** *auto*
done

5.3 Soundness

The soundness theorem:

lemma *help1*: **assumes** *enat a + X ≤ Y*
enat b + Z ≤ X
shows *enat (a + b) + Z ≤ Y*
using *assms* **by** (*metis ab_semigroup_add_class.add_ac(1) add_left_mono order_trans plus_enat_simps(1)*)

lemma *help2'*: **assumes** *enat p + INV t ≤ INV s*
0 < p INV s = enat n
shows *INV t < INV s*
using *assms iadd_le_enat_iff* **by** *auto*

lemma *help2*: **assumes** *enat p + INV t + 1 ≤ INV s*
INV s = enat n
shows *INV t < INV s*
using *assms le_less_trans not_less_iff_gr_or_eq* **by** *fastforce*

lemma *Seq_sound*: **assumes** $\models_2$ {*P1*} *C1* {*P2*}
$\models_2$ {*P2*} *C2* {*P3*}
shows $\models_2$ {*P1*} *C1* ;; *C2* {*P3*}
unfolding *hoare2_valid_def*
proof (*safe*)
fix *s*
assume *ninfP1*: *P1 s < ∞*
with *assms(1)[unfolded hoare2_valid_def]* **obtain** *t1 p1*
where *1*: (*C1*, *s*) ⇒ *p1* ⇓ *t1* **and** *q1*: *enat p1 + P2 t1 ≤ P1 s* **by** *blast*
with *ninfP1* **have** *ninfP2*: *P2 t1 < ∞*
using *not_le* **by** *fastforce*
with *assms(2)[unfolded hoare2_valid_def]* **obtain** *t2 p2*
where *2*: (*C2*, *t1*) ⇒ *p2* ⇓ *t2* **and** *q2*: *enat p2 + P3 t2 ≤ P2 t1* **by** *blast*
with *ninfP2* **have** *ninfP3*: *P3 t2 < ∞*
using *not_le* **by** *fastforce*

from *Big_StepT.Seq[OF 1 2]* **have** *bigstep*: (*C1*;; *C2*, *s*) ⇒ *p1 + p2* ⇓ *t2* **by** *simp*
from *help1[OF q1 q2]* **have** *potential*: *enat (p1 + p2) + P3 t2 ≤ P1 s* **.**

show ∃ *t p*. (*C1*;; *C2*, *s*) ⇒ *p* ⇓ *t* ∧ *enat p + P3 t ≤ P1 s*
apply(*rule exI*[**where** *x=t2*])
apply(*rule exI*[**where** *x=p1 + p2*])

using *bigstep potential* **by** *simp*
qed

theorem *hoare2_sound*: $\vdash_2 \{P\}c\{\ Q\} \implies \models_2 \{P\}c\{\ Q\}$
proof(*induction rule*: *hoare2.induct*)
case (*Skip P*)
show *?case* **unfolding** *hoare2_valid_def* **apply**(*safe*)
subgoal for *s* **apply**(*rule exI*[**where** *x*=*s*]) **apply**(*rule exI*[**where** *x*=*Suc 0*])
by (*auto simp*: *eSuc_enat_iff eSuc_enat*)
done
next
case (*Assign P a x*)
show *?case* **unfolding** *hoare2_valid_def* **apply**(*safe*)
subgoal for *s* **apply**(*rule exI*[**where** *x*=*s*[*a*/*x*]]) **apply**(*rule exI*[**where** *x*=*Suc 0*])
by (*auto simp*: *eSuc_enat_iff eSuc_enat*)
done
next
case (*Seq P1 C1 P2 C2 P3*)
thus *?case* **using** *Seq_sound* **by** *auto*
next
case (*If P b c1 Q c2*)
show *?case* **unfolding** *hoare2_valid_def*
proof (*safe*)
fix *s*
assume *eSuc* (*P s*) $< \infty$
then have *i*: *P s* $< \infty$
using *enat_ord_simps*(*4*) **by** *fastforce*
show $\exists$ *t p*. (*IF b THEN c1 ELSE c2*, *s*) $\Rightarrow$ *p* $\Downarrow$ *t* $\wedge$ *enat p* + *Q t* $\leq$ *eSuc* (*P s*)
proof(*cases bval b s*)
case *True*
with *i* **have** *P s* + *emb* (*bval b s*) $< \infty$ **by** *simp*
with *If*(*3*)[*unfolded hoare2_valid_def*] **obtain** *p t*
where *1*: (*c1*, *s*) $\Rightarrow$ *p* $\Downarrow$ *t* **and** *q*: *enat p* + *Q t* $\leq$ *P s* + *emb* (*bval b s*) **by** *blast*
from *Big_StepT.IfTrue*[*OF True 1*] **have** *2*: (*IF b THEN c1 ELSE c2*, *s*) $\Rightarrow$ *p* + *1* $\Downarrow$ *t* **by** *simp*
show *?thesis* **apply**(*rule exI*[**where** *x*=*t*]) **apply**(*rule exI*[**where** *x*=*p*+*1*])
apply(*safe*) **apply**(*fact*)
using *q True* **apply**(*simp*)

by (*metis eSuc_enat eSuc_ile_mono iadd_Suc*)
next
case *False*
with *i* **have** $P\ s + emb\ (\sim\ bval\ b\ s) < \infty$ **by** *simp*
with *If(4)[unfolded hoare2_valid_def]* **obtain** *p t*
where *1*: $(c2, s) \Rightarrow p \Downarrow t$ **and** *q*: $enat\ p + Q\ t \leq P\ s + emb\ (\sim\ bval\ b\ s)$ **by** *blast*
from *Big_StepT.IfFalse[OF False 1]* **have** *2*: $(IF\ b\ THEN\ c1\ ELSE\ c2, s) \Rightarrow p + 1 \Downarrow t$ **by** *simp*
show *?thesis* **apply**(*rule exI*[**where** *x=t*]) **apply**(*rule exI*[**where** *x=p+1*])
apply(*safe*) **apply**(*fact*)
using *q False* **apply**(*simp*)
by (*metis eSuc_enat eSuc_ile_mono iadd_Suc*)
qed
qed
next
case (*conseq P c Q P′ Q′*)
show *?case* **unfolding** *hoare2_valid_def*
proof (*safe*)
fix *s*
assume $P'\ s < \infty$
with *conseq(2)* **have** $P\ s < \infty$
using *le_less_trans* **by** *blast*
with *conseq(4)[unfolded hoare2_valid_def]* **obtain** *p t* **where** $(c, s) \Rightarrow p \Downarrow t$ $enat\ p + Q\ t \leq P\ s$ **by** *blast*
with *conseq(2,3)* **show** $\exists\, t\ p.\ (c, s) \Rightarrow p \Downarrow t \wedge enat\ p + Q'\ t \leq P'\ s$
by (*meson add_left_mono dual_order.trans*)
qed
next
case (*While INV b c*)

from *While(2)[unfolded hoare2_valid_def]*
have *WH2*: $\bigwedge s.\ INV\ s + \uparrow (bval\ b\ s) < \infty \Longrightarrow (\exists\, t\ p.\ (c, s) \Rightarrow p \Downarrow t \wedge enat\ p + INV\ t + 1 \leq INV\ s + \uparrow (bval\ b\ s))$
by (*simp add*: *add.commute add.left_commute*)

show *?case* **unfolding** *hoare2_valid_def*
proof (*safe*)
fix *s*
assume *ninfINV*: $INV\ s + 1 < \infty$
then have $INV\ s < \infty$
using *enat_ord_simps(4)* **by** *fastforce*
then obtain *n* **where** *i*: $INV\ s = enat\ n$ **using** *not_infinity_eq*

by *auto*

In order to prove validity, we induct on the value of the Invariant, which is a finite number and decreases in every loop iteration. For each step we show that validity holds.

have $INV\ s = enat\ n \Longrightarrow \exists\, t\ p.\ (WHILE\ b\ DO\ c,\ s) \Rightarrow p \Downarrow t \wedge enat\ p + (INV\ t + emb\ (\neg\ bval\ b\ t)) \le INV\ s + 1$
proof (*induct n arbitrary*: *s rule*: *less_induct*)
case (*less n*)
show *?case*
proof (*cases bval b s*)
case *False*
show *?thesis*
using *WhileFalse*[*OF False*] *one_enat_def* **by** *fastforce*
next
case *True*
— obtain the loop body from the outer IH
with *less*(*2*) *WH2* **obtain** *t p*
where *o*: $(c,\ s) \Rightarrow p \Downarrow t$
and *q*: $enat\ p + INV\ t + 1 \le INV\ s$ **by** *force*

— prepare premises to ...
from *q* **have** *g*: $INV\ t < INV\ s$
using *help2 less*(*2*) **by** *metis*
then have *ninfINVt*: $INV\ t < \infty$ **using** *less*(*2*)
using *enat_ord_simps*(*4*) **by** *fastforce*
then obtain n' **where** *i*: $INV\ t = enat\ n'$ **using** *not_infinity_eq*
by *auto*
with *less*(*2*) **have** *ii*: $n' < n$
using *g* **by** *auto*
— ... obtain the tail of the While loop from the inner IH
from *i ii less*(*1*) **obtain** *t2 p2*
where *o2*: $(WHILE\ b\ DO\ c,\ t) \Rightarrow p2 \Downarrow t2$
and *q2*: $enat\ p2 + (INV\ t2 + emb\ (\neg\ bval\ b\ t2)) \le INV\ t + 1$
by *blast*
have *ende*: $\sim$ *bval b t2*
apply(*rule ccontr*) **apply**(*simp*) **using** *q2 ninfINVt*
by (*simp add*: *i one_enat_def*)

— combine body and tail to one loop unrolling:
— - the Bigstep Semantic
from *WhileTrue*[*OF True o o2*] **have** *BigStep*: $(WHILE\ b\ DO\ c,\ s) \Rightarrow 1 + p + p2 \Downarrow t2$ **by** *simp*

— - the potentialPreservation
from *ende q2* **have** *q2'*: $enat\ p2 + INV\ t2 \leq INV\ t + 1$ **by** *simp*

have *potentialPreservation*: $enat\ (1 + p + p2) + (INV\ t2 + \uparrow (\neg\ bval\ b\ t2)) \leq INV\ s + 1$
proof −
have $enat\ (1 + p + p2) + (INV\ t2 + \uparrow (\neg\ bval\ b\ t2))$
$= enat\ (Suc\ (p + p2)) + INV\ t2$ **using** *ende* **by** *simp*
also have $\ldots = enat\ (Suc\ p) + enat\ p2 + INV\ t2$ **by** *fastforce*
also have $\ldots \leq enat\ (Suc\ p) + INV\ t + 1$ **using** *q2'*
by (*metis ab_semigroup_add_class.add_ac(1) add_left_mono*)
also have $\ldots \leq INV\ s + 1$ **using** *q*
by (*metis* (*no_types*, *opaque_lifting*) *add.commute add_left_mono eSuc_enat iadd_Suc plus_1_eSuc(1)*)
finally show $enat\ (1 + p + p2) + (INV\ t2 + \uparrow (\neg\ bval\ b\ t2)) \leq INV\ s + 1$ **.**
qed

— finally combine BigStep Semantic and TimeBound
show *?thesis*
apply(*rule exI*[**where** *x=t2*])
apply(*rule exI*[**where** $x = 1 + p + p2$])
apply(*safe*)
by(*fact BigStep potentialPreservation*)+
qed
qed
from *this*[*OF i*] **show** $\exists\, t\ p.\ (WHILE\ b\ DO\ c,\ s) \Rightarrow p \Downarrow t \wedge enat\ p + (INV\ t + emb\ (\neg\ bval\ b\ t)) \leq INV\ s + 1$ **.**
qed
qed

5.4 Completeness

definition *wp2* :: $com \Rightarrow qassn \Rightarrow qassn$ (‹wp_2›) **where**
$wp_2\ c\ Q\ =\ (\lambda s.\ (if\ (\exists\, t\ p.\ (c,s) \Rightarrow p \Downarrow t \wedge Q\ t < \infty)\ \ then\ enat\ (THE\ p.\ \exists\, t.\ (c,s) \Rightarrow p \Downarrow t) + Q\ (THE\ t.\ \exists\, p.\ (c,s) \Rightarrow p \Downarrow t)\ else\ \infty))$

lemma *wp2_alt*: $wp_2\ c\ Q = (\lambda s.\ (if\ {\downarrow}(c,s)\ then\ enat\ ({\downarrow_t}\ (c,\ s)) + Q\ ({\downarrow_s}\ (c,\ s))\ else\ \infty))$
apply(*rule ext*) **by**(*auto simp*: *bigstepT_the_state wp2_def split*: *if_split*)

theorem *wp2_is_weakestprePotential*: $\models_2 \{P\}c\{Q\} \longleftrightarrow (\forall\, s.\ wp_2\ c\ Q\ s$

$\leq$ *P s*)
  unfolding *wp2_def hoare2_valid_def*
  apply(*rule*)
  subgoal
    apply(*safe*) **subgoal for** *s*
      apply(*cases* $\exists$ *t p*. (*c*, *s*) $\Rightarrow$ *p* $\Downarrow$ *t* $\wedge$ *Q t* $< \infty$)
        apply(*simp*) **apply** *auto* **oops**

lemma *wp2_Skip*[*simp*]: wp_2 *SKIP Q* = (%*s*. *eSuc* (*Q s*))
  apply(*auto intro*!: *ext simp*: *wp2_def*)
   prefer *2*
   apply(*simp only*: *SKIPnot*)
   apply(*simp*)
  apply(*simp only*: *SKIPp SKIPt*)
  using *one_enat_def plus_1_eSuc*(*1*) **by** *auto*

lemma *wp2_Assign*[*simp*]: wp_2 (*x* ::= *e*) *Q* = (λ*s*. *eSuc* (*Q* (*s*(*x* := *aval e s*))))
by (*auto intro*!: *ext simp*: *wp2_def ASSp ASSt ASSnot eSuc_enat*)

lemma *wp2_Seq*[*simp*]: wp_2 (c_1;;c_2) *Q* = wp_2 c_1 (wp_2 c_2 *Q*)
unfolding *wp2_def*
proof (*rule*, *case_tac* $\exists$ *t p*. (c_1;; c_2, *s*) $\Rightarrow$ *p* $\Downarrow$ *t* $\wedge$ *Q t* $< \infty$, *goal_cases*)
  case (*1 s*)
  then obtain *u p* **where** *ter*: (c_1;; c_2, *s*) $\Rightarrow$ *p* $\Downarrow$ *u* **and** *Q*: *Q u* $< \infty$ **by** *blast*
  then obtain *t p1 p2* **where** *i*: (c_1 , *s*) $\Rightarrow$ *p1* $\Downarrow$ *t* **and** *ii*: (c_2 , *t*) $\Rightarrow$ *p2* $\Downarrow$ *u* **and** *p*: *p1* + *p2* = *p* **by** *blast*

  from *bigstepT_the_state*[*OF i*] **have** *t*: $\downarrow_s$ (c_1, *s*) = *t*
    by *blast*
  from *bigstepT_the_state*[*OF ii*] **have** *t2*: $\downarrow_s$ (c_2, *t*) = *u*
    by *blast*
  from *bigstepT_the_cost*[*OF i*] **have** *firstcost*: $\downarrow_t$ (c_1, *s*) = *p1*
    by *blast*
  from *bigstepT_the_cost*[*OF ii*] **have** *secondcost*: $\downarrow_t$ (c_2, *t*) = *p2*
    by *blast*

  have *totalcost*: $\downarrow_t$(c_1;; c_2, *s*) = *p1* + *p2*
    using *bigstepT_the_cost*[*OF ter*] *p* **by** *auto*
  have *totalstate*: $\downarrow_s$(c_1;; c_2, *s*) = *u*

using *bigstepT_the_state*[*OF ter*] **by** *auto*

have *c2*: $\exists$ *ta p*. (c_2, *t*) $\Rightarrow$ *p* $\Downarrow$ *ta* $\wedge$ *Q ta* $< \infty$
apply(*rule exI*[**where** *x*= *u*])
apply(*rule exI*[**where** *x*= *p2*]) **apply** *safe* **apply** *fact*+ **done**

have *C*: $\exists$ *t p*. (c_1, *s*) $\Rightarrow$ *p* $\Downarrow$ *t* $\wedge$ (*if* $\exists$ *ta p*. (c_2, *t*) $\Rightarrow$ *p* $\Downarrow$ *ta* $\wedge$ *Q ta* $< \infty$ *then enat* (*THE p*. *Ex* (*big_step_t* (c_2, *t*) *p*)) + *Q* (*THE ta*. $\exists$ *p*. (c_2, *t*) $\Rightarrow$ *p* $\Downarrow$ *ta*) *else* ∞) $< \infty$
apply(*rule exI*[**where** *x*=*t*])
apply(*rule exI*[**where** *x*=*p1*])
apply *safe*
apply *fact*
apply(*simp only*: *c2 if_True*)
using *Q bigstepT_the_state ii* **by** *auto*

show *?case*
apply(*simp only*: *1 if_True t t2 c2 C totalcost totalstate firstcost secondcost*) **by** *fastforce*
next
case (*2 s*)
show *?case* **apply**(*simp only*: *2 if_False*)
apply *auto* **using** *2*
by *force*
qed

lemma *wp2_If*[*simp*]:
wp_2 (*IF b THEN* c_1 *ELSE* c_2) *Q* = (λ*s*. *eSuc* (wp_2 (*if bval b s then* c_1 *else* c_2) *Q s*))
apply (*auto simp*: *wp2_def fun_eq_iff*)
subgoal for *x t p i ta ia xa* **apply**(*simp only*: *IfTrue*[*THEN bigstepT_the_state*])
apply(*simp only*: *IfTrue*[*THEN bigstepT_the_cost*])
apply(*simp only*: *bigstepT_the_cost bigstepT_the_state*)
by (*simp add*: *eSuc_enat*)
apply(*simp only*: *bigstepT_the_state bigstepT_the_cost*) **apply** *force*
apply(*simp only*: *bigstepT_the_state bigstepT_the_cost*)
proof(*goal_cases*)
case (*1 x t p i ta ia xa*)
note *f*= *IfFalse*[*THEN bigstepT_the_state*, *of b x* c_2 *xa ta Suc xa* c_1, *simplified*, *OF 1*(*4*) *1*(*5*)]
note *f2*= *IfFalse*[*THEN bigstepT_the_cost*, *of b x* c_2 *xa ta Suc xa* c_1, *simplified*, *OF 1*(*4*) *1*(*5*)]

note *g= bigstep_det*[*OF 1*(*1*) *1*(*5*)]
show *?case*
apply(*simp only*: *f f2*) **using** *1 g*
by (*simp add*: *eSuc_enat*)
next
case *2*
then
show *?case*
apply(*simp only*: *bigstepT_the_state bigstepT_the_cost*) **apply** *force*
done
qed

lemma assumes *b*: *bval b s*
shows *wp2WhileTrue*: wp_2 *c* (wp_2 (*WHILE b DO c*) *Q*) *s* + *1* ≤ wp_2 (*WHILE b DO c*) *Q s*
proof (*cases* ∃ *t p*. (*WHILE b DO c*, *s*) ⇒ *p* ⇓ *t* ∧ *Q t* < ∞)
case *True*
then obtain *t p* **where** *w*: (*WHILE b DO c*, *s*) ⇒ *p* ⇓ *t* **and** *q*: *Q t* < ∞ **by** *blast*
from *b w* **obtain** *p1 p2 t1* **where** *c*: (*c*, *s*) ⇒ *p1* ⇓ *t1* **and** *w'*: (*WHILE b DO c*, *t1*) ⇒ *p2* ⇓ *t* **and** *sum*: *1* + *p1* + *p2* = *p*
by *auto*
have *g*: ∃ *ta p*. (*WHILE b DO c*, *t1*) ⇒ *p* ⇓ *ta* ∧ *Q ta* < ∞
apply(*rule exI*[**where** *x=t*])
apply(*rule exI*[**where** *x=p2*])
apply *safe* **apply** *fact*+ **done**

have *h*: ∃ *t p*. (*c*, *s*) ⇒ *p* ⇓ *t* ∧ (*if* ∃ *ta p*. (*WHILE b DO c*, *t*) ⇒ *p* ⇓ *ta* ∧ *Q ta* < ∞ *then enat* (*THE p*. *Ex* (*big_step_t* (*WHILE b DO c*, *t*) *p*)) + *Q* (*THE ta*. ∃ *p*. (*WHILE b DO c*, *t*) ⇒ *p* ⇓ *ta*) *else* ∞) < ∞
apply(*rule exI*[**where** *x=t1*])
apply(*rule exI*[**where** *x=p1*])
apply *safe* **apply** *fact*
apply(*simp only*: *g if_True*) **using** *bigstepT_the_state bigstepT_the_cost w' q* **by**(*auto*)

have wp_2 *c* (wp_2 (*WHILE b DO c*) *Q*) *s* + *1* = *enat p* + *Q t*
unfolding *wp2_def* **apply**(*simp only*: *h if_True*)
apply(*simp only*: *bigstepT_the_state*[*OF c*] *bigstepT_the_cost*[*OF c*] *g if_True bigstepT_the_state*[*OF w'*] *bigstepT_the_cost*[*OF w'*]) **using** *sum*
by (*metis One_nat_def ab_semigroup_add_class.add_ac*(*1*) *add.commute add.right_neutral eSuc_enat plus_1_eSuc*(*2*) *plus_enat_simps*(*1*))
also have ... = wp_2 (*WHILE b DO c*) *Q s*

unfolding *wp2_def* **apply**(*simp only*: *True if_True*)
using *bigstepT_the_state bigstepT_the_cost w* **apply**(*simp*) **done**
finally show *?thesis* **by** *simp*
next
case *False*
have wp_2 (*WHILE b DO c*) *Q s* = ∞
unfolding *wp2_def*
apply(*simp only*: *False if_False*) **done**
then show *?thesis* **by** *auto*
qed

lemma assumes *b*: *bval b s*
shows *wp2WhileTrue′*: wp_2 *c* (wp_2 (*WHILE b DO c*) *Q*) *s* + *1* = wp_2 (*WHILE b DO c*) *Q s*
proof (*cases* ∃ *p t*. (*WHILE b DO c*, *s*) ⇒ *p* ⇓ *t*)
case *True*
then obtain *t p* **where** *w*: (*WHILE b DO c*, *s*) ⇒ *p* ⇓ *t* **by** *blast*
from *b w* **obtain** *p1 p2 t1* **where** *c*: (*c*, *s*) ⇒ *p1* ⇓ *t1* **and** *w′*: (*WHILE b DO c*, *t1*) ⇒ *p2* ⇓ *t* **and** *sum*: *1* + *p1* + *p2* = *p*
by *auto*
then have *z*: ↓ (*c*, *s*) **and** *z2*: ↓ (*WHILE b DO c*, *t1*) **by** *auto*

have wp_2 *c* (wp_2 (*WHILE b DO c*) *Q*) *s* + *1* = *enat p* + *Q t*
unfolding *wp2_alt* **apply**(*simp only*: *z if_True*)
apply(*simp only*: *bigstepT_the_state*[*OF c*] *bigstepT_the_cost*[*OF c*] *z2 if_True bigstepT_the_state*[*OF w′*] *bigstepT_the_cost*[*OF w′*])
using *sum*
by (*metis One_nat_def ab_semigroup_add_class.add_ac*(*1*) *add.commute add.right_neutral eSuc_enat plus_1_eSuc*(*2*) *plus_enat_simps*(*1*))
also have ... = wp_2 (*WHILE b DO c*) *Q s*
unfolding *wp2_alt* **apply**(*simp only*: *True if_True*)
using *bigstepT_the_state bigstepT_the_cost w* **apply**(*simp*) **done**
finally show *?thesis* **by** *simp*
next
case *False*
have ¬ (↓ (*WHILE b DO c*, $\downarrow_s$(*c*,*s*)) ∧ ↓ (*c*, *s*))
proof (*rule*)
assume *P*: ↓ (*WHILE b DO c*, $\downarrow_s$ (*c*, *s*)) ∧ ↓ (*c*, *s*)
then obtain *t s′* **where** *A*: (*c*,*s*) ⇒ *t* ⇓ *s′* **by** *blast*
with *A P* **have** ↓ (*WHILE b DO c*, *s′*) **using** *bigstepT_the_state* **by** *auto*
then obtain *t′ s″* **where** *B*: (*WHILE b DO c*,*s′*) ⇒ *t′* ⇓ *s″* **by** *auto*
have (*WHILE b DO c*, *s*) ⇒ *1*+*t*+*t′* ⇓ *s″* **apply**(*rule WhileTrue*) **using** *b A B* **by** *auto*

then have ↓ (*WHILE b DO c*, *s*) **by** *auto*
thus *False* **using** *False* **by** *auto*
qed
then have ¬↓ (*WHILE b DO c*, $\downarrow_s$(*c*,*s*)) ∨ ¬↓ (*c*, *s*) **by** *simp*

then show *?thesis* **apply** *rule*
subgoal unfolding *wp2_alt* **apply**(*simp only*: *if_False False*) **by** *auto*
subgoal unfolding *wp2_alt* **apply**(*simp only*: *if_False False*) **by** *auto*
done
qed

lemma assumes *b*: ~ *bval b s*
shows *wp2WhileFalse*: $Q\ s\ +\ 1 \leq wp_2$ (*WHILE b DO c*) *Q s*
proof (*cases* ∃ *t p*. (*WHILE b DO c*, *s*) ⇒ *p* ⇓ *t* ∧ *Q t* < ∞)
case *True*
with *b* **obtain** *t p* **where** *w*: (*WHILE b DO c*, *s*) ⇒ *p* ⇓ *t* **and** *Q t* < ∞ **by** *blast*
with *b* **have** *c*: *s*=*t p*=*Suc 0* **by** *auto*
have wp_2 (*WHILE b DO c*) *Q s* = *Q s* + *1*
unfolding *wp2_def* **apply**(*simp only*: *True if_True*)
using *w c bigstepT_the_cost bigstepT_the_state* **by**(*auto simp add*: *one_enat_def*)
then show *?thesis* **by** *auto*
next
case *False*
have wp_2 (*WHILE b DO c*) *Q s* = ∞
unfolding *wp2_def*
apply(*simp only*: *False if_False*) **done**
then show *?thesis* **by** *auto*
qed

lemma *thet_WhileFalse*: ~ *bval b s* ⟹ $\downarrow_t$ (*WHILE b DO c*, *s*) = *1* **by** *auto*
lemma *thes_WhileFalse*: ~ *bval b s* ⟹ $\downarrow_s$ (*WHILE b DO c*, *s*) = *s* **by** *auto*

lemma assumes *b*: ~ *bval b s*
shows *wp2WhileFalse'*: *Q s* + *1* = wp_2 (*WHILE b DO c*) *Q s*
proof −
from *b* **have** *T*: ↓ (*WHILE b DO c*, *s*) **by** *auto*
show *?thesis* **unfolding** *wp2_alt* **using** *b* **apply**(*simp only*: *T if_True*)
by(*simp add*: *thet_WhileFalse thes_WhileFalse one_enat_def*)
qed

```
lemma wp2While: (if bval b s then wp₂ c (wp₂ (WHILE b DO c) Q) s else
Q s) + 1 = wp₂ (WHILE b DO c) Q s
  apply(cases bval b s)
  using wp2WhileTrue' apply simp
  using wp2WhileFalse' apply simp   done



lemma assumes ⋀Q. ⊢₂ {wp₂ c Q} c {Q}
  shows ⊢₂ {wp₂ (WHILE b DO c) Q} WHILE b DO c {Q}
proof −
  let ?I = %s. (if bval b s then wp₂ c (wp₂ (WHILE b DO c) Q) s else Q s)
  from assms[of wp₂ (WHILE b DO c) Q]
  have A: ⊢₂ {wp₂ c (wp₂ (WHILE b DO c) Q)} c {wp₂ (WHILE b DO
c) Q} .
  have B: ⊢₂ {λs. (?I s) + ↑ (bval b s)} c {λt. (?I t) + 1}
    apply(rule conseq)
      apply(rule A)
     apply simp
      using wp2While apply simp done
  from hoare2.While[where I=?I]
  have C: ⊢₂ {λs. (?I s) + ↑ (bval b s)} c {λt. (?I t) + 1} ⟹
         ⊢₂ {λs. (?I s) + 1} WHILE b DO c {λs. (?I s) + ↑ (¬ bval b s)}
by simp
  from C[OF B] have D: ⊢₂ {λs. (?I s) + 1} WHILE b DO c {λs. (?I s)
+ ↑ (¬ bval b s)} .
  show ⊢₂ {wp₂ (WHILE b DO c) Q} WHILE b DO c {Q}
    apply(rule conseq)
      apply(rule D)
    using wp2While apply simp
      apply simp  done
qed


lemma wp2_is_pre: ⊢₂ {wp₂ c Q} c { Q}
proof (induction c arbitrary: Q)
  case SKIP show ?case by (auto intro: hoare2.Skip)
next
  case Assign show ?case by (auto intro:hoare2.Assign)
next
  case Seq thus ?case by (auto intro:hoare2.Seq)
```

next
  case (*If x1 c1 c2 Q*) **thus** *?case*
    apply (*auto intro*!: *hoare2.If*)
     apply(*rule hoare2.conseq*)
       apply(*auto*)
     apply(*rule hoare2.conseq*)
      apply(*auto*)
    done
next
  case (*While b c*)
  show *?case*
    apply(*rule conseq*)
       apply(*rule hoare2.While*[**where** *I=%s.* (*if bval b s then* wp_2 *c* (wp_2 (*WHILE b DO c*) *Q*) *s else Q s*)])
      apply(*rule conseq*)
        apply(*rule While*[*of* wp_2 (*WHILE b DO c*) *Q*])
    using *wp2While* **by** *auto*
qed

lemma *wp2_is_weakestprePotential1*: $\models_2 \{P\}c\{Q\} \Longrightarrow (\forall s.\ wp_2\ c\ Q\ s \leq P\ s)$
apply(*auto simp*: *hoare2_valid_def wp2_def*)
proof (*goal_cases*)
  case (*1 s t p i*)
  show *?case*
  proof(*cases* $P\ s < \infty$)
    case *True*
    with *1*(*1*) **obtain** *t p'* **where** *i*: $(c, s) \Rightarrow p' \Downarrow t$ **and** *ii*: *enat* $p' + Q\ t \leq P\ s$
      by *auto*
   show *?thesis* **apply**(*simp add*: *bigstepT_the_state*[*OF i*] *bigstepT_the_cost*[*OF i*] *ii*) **done**
  qed *simp*
qed *force*

lemma *wp2_is_weakestprePotential2*: $(\forall s.\ wp_2\ c\ Q\ s \leq P\ s) \Longrightarrow\ \models_2 \{P\}c\{Q\}$
apply(*auto simp*: *hoare2_valid_def wp2_def*)
proof (*goal_cases*)
  case (*1 s i*)
  then have *A*: (*if* $\exists t.\ (\exists p.\ (c, s) \Rightarrow p \Downarrow t) \wedge (\exists i.\ Q\ t = enat\ i)$ *then enat* (*THE p. Ex* (*big_step_t* (*c, s*) *p*)) + *Q* (*THE t.* $\exists p.\ (c, s) \Rightarrow p \Downarrow t$) *else*

$\infty) \leq P\ s$
  by *fast*
 show *?case*
 proof $(cases\ \exists t.\ (\exists p.\ (c,\ s) \Rightarrow p \Downarrow t) \wedge (\exists i.\ Q\ t = enat\ i))$
  case *True*
  then obtain $t\ p$ **where** i: $(c,\ s) \Rightarrow p \Downarrow t$ **by** *blast*
  from *True A* **have** $enat\ p + Q\ t \leq P\ s$ **by** (*simp add*: *bigstepT_the_cost*[*OF i*] *bigstepT_the_state*[*OF i*])
  then have $(c,\ s) \Rightarrow p \Downarrow t \wedge enat\ p + Q\ t \leq enat\ i$ **using** *1*(*2*) *i* **by** *simp*
  then show *?thesis* **by** *auto*
 next
  case *False*
  with *A* **have** $P\ s \geq \infty$ **by** *auto*
  then show *?thesis* **using** *1* **by** *auto*
 qed
qed

theorem *wp2_is_weakestprePotential*: $(\forall s.\ wp_2\ c\ Q\ s \leq P\ s) \longleftrightarrow \models_2 \{P\}c\{Q\}$
using *wp2_is_weakestprePotential2 wp2_is_weakestprePotential1* **by** *metis*

theorem *hoare2_complete*: $\models_2 \{P\}c\{Q\} \Longrightarrow \vdash_2 \{P\}c\{\ Q\}$
apply(*rule conseq*[*OF wp2_is_pre*, **where** Q'=Q **and** Q=Q, *simplified*])
using *wp2_is_weakestprePotential1* **by** *blast*

corollary *hoare2_sound_complete*: $\vdash_2 \{P\}c\{Q\} \longleftrightarrow \models_2 \{P\}c\{\ Q\}$
by (*metis hoare2_sound hoare2_complete*)

end

5.5 Verification Condition Generator

theory *Quant_VCG*
imports *Quant_Hoare*
begin

datatype *acom* =
  Askip (‹*SKIP*›) |
  Aassign vname aexp (‹(_ ::= _)› [*1000*, *61*] *61*) |
  Aseq acom acom (‹_;;/ _› [*60*, *61*] *60*) |
  Aif bexp acom acom (‹(*IF* _/ *THEN* _/ *ELSE* _)› [*0*, *0*, *61*] *61*) |
  Awhile qassn bexp acom (‹({_}/ *WHILE* _/ *DO* _)› [*0*, *0*, *61*] *61*)

notation *com.SKIP* (‹*SKIP*›)

fun *strip* :: *acom* ⇒ *com* **where**
strip SKIP = *SKIP* |
strip (*x* ::= *a*) = (*x* ::= *a*) |
strip (C_1;; C_2) = (*strip* C_1;; *strip* C_2) |
strip (*IF b THEN* C_1 *ELSE* C_2) = (*IF b THEN strip* C_1 *ELSE strip* C_2) |
strip ({_} *WHILE b DO C*) = (*WHILE b DO strip C*)

fun *pre* :: *acom* ⇒ *qassn* ⇒ *qassn* **where**
pre SKIP Q = (λ*s*. *eSuc* (*Q s*)) |
pre (*x* ::= *a*) *Q* = (λ*s*. *eSuc* (*Q* (*s*[*a*/*x*]))) |
pre (C_1;; C_2) *Q* = *pre* C_1 (*pre* C_2 *Q*) |
pre (*IF b THEN* C_1 *ELSE* C_2) *Q* =
  (λ*s*. *eSuc* (*if bval b s then pre* C_1 *Q s else pre* C_2 *Q s*)) |
pre ({*I*} *WHILE b DO C*) *Q* = (λ*s*. *I s* + *1*)

fun *vc* :: *acom* ⇒ *qassn* ⇒ *bool* **where**
vc SKIP Q = *True* |
vc (*x* ::= *a*) *Q* = *True* |
vc (C_1 ;; C_2) *Q* = ((*vc* C_1 (*pre* C_2 *Q*)) ∧ (*vc* C_2 *Q*)) |
vc (*IF b THEN* C_1 *ELSE* C_2) *Q* = (*vc* C_1 *Q* ∧ *vc* C_2 *Q*) |
vc ({*I*} *WHILE b DO C*) *Q* = ((∀ *s*. (*pre C* (λ*s*. *I s* + *1*) *s* ≤ *I s* + ↑(*bval b s*)) ∧ (*Q s* ≤ *I s* + ↑ (¬ *bval b s*))) ∧ *vc C* (%*s*. *I s* + *1*))

5.5.1 Soundness of VCG

lemma *vc_sound*: *vc C Q* ⟹ ⊢₂ {*pre C Q*} *strip C* { *Q* }
proof (*induct C arbitrary*: *Q*)
  case (*Aif b C1 C2*)
  then have *Aif1*: $\vdash_2$ {*pre C1 Q*} *strip C1* {*Q*} **and** *Aif2*: $\vdash_2$ {*pre C2 Q*} *strip C2* {*Q*} **by** *auto*
    show *?case* **apply** *auto* **apply**(*rule hoare2.conseq*)
      apply(*rule hoare2.If*[**where** *P*=%*s*. *if bval b s then pre C1 Q s else pre C2 Q s* **and** *Q*=*Q*])
    subgoal

apply(*rule hoare2.conseq*)
apply (*fact Aif1*)
subgoal for *s* **apply**(*cases bval b s*) **by** *auto*
apply *simp* **done**
subgoal
apply(*rule hoare2.conseq*)
apply (*fact Aif2*)
subgoal for *s* **apply**(*cases bval b s*) **by** *auto*
apply *simp* **done**
apply *auto*
done
next
case (*Awhile I b C*)
then have *i*: ($\bigwedge Q.\ vc\ C\ Q \Longrightarrow \vdash_2 \{pre\ C\ Q\}\ strip\ C\ \{Q\}$)
and *ii*: $\forall s.\ pre\ C\ (\lambda s.\ I\ s + 1)\ s \leq I\ s + \uparrow (bval\ b\ s) \wedge Q\ s \leq\ I\ s + \uparrow (\neg\ bval\ b\ s)$
and *iii*: $vc\ C\ (\lambda s.\ I\ s + 1)$ **by** *auto*

from *i iii* **have** *A*: $\vdash_2 \{pre\ C\ (\lambda s.\ I\ s + 1)\}\ strip\ C\ \{(\lambda s.\ I\ s + 1)\}$ **by** *auto*

have $\vdash_2 \{\lambda s.\ I\ s + 1\}\ WHILE\ b\ DO\ strip\ C\ \{Q\}$
apply(*rule hoare2.conseq*)
apply(*rule hoare2.While*[**where** *I=I*])
apply(*rule hoare2.conseq*)
apply(*rule A*) **using** *ii* **by** *auto*
then show *?case* **by** *auto*
qed (*auto intro*: *hoare2.Skip hoare2.Assign hoare2.Seq*)

lemma *vc_sound'*: $[\![vc\ C\ Q\ ;\ (\forall s.\ pre\ C\ Q\ s \leq P\ s)\ ]\!] \Longrightarrow\ \vdash_2 \{P\}\ strip\ C\ \{\ Q\ \}$
apply(*rule hoare2.conseq*)
apply(*rule vc_sound*) **by** *auto*

5.5.2 Completeness

lemma *pre_mono*: **assumes** $\bigwedge s.\ P'\ s \leq P\ s$
shows $\bigwedge s.\ pre\ C\ P'\ s \leq pre\ C\ P\ s$
using *assms* **by** (*induct C arbitrary*: $P\ P'$, *auto*)

lemma *vc_mono*: **assumes** $\bigwedge s.\ P'\ s \leq P\ s$
shows $vc\ C\ P \Longrightarrow vc\ C\ P'$

using *assms* **proof** (*induct C arbitrary*: *P P′*)
  case (*Awhile I b C*)
  thus *?case*
    apply (*auto simp*: *pre_mono*)
    using *order.trans* **by** *blast*
qed (*auto simp*: *pre_mono*)

lemma $\vdash_2 \{ P \}\ c\ \{ Q \} \Longrightarrow \exists C.\ strip\ C = c \wedge vc\ C\ Q \wedge (\forall s.\ pre\ C\ Q\ s \leq P\ s)$
  (**is** _ $\Longrightarrow$ $\exists C.$ *?G P c Q C*)
proof (*induction rule*: *hoare2.induct*)
  case (*Skip P*)
  show *?case* (**is** $\exists C.$ *?C C*)
  proof show *?C Askip* **by** *auto*
  qed
next
  case (*Assign P a x*)
  show *?case* (**is** $\exists C.$ *?C C*)
  proof show *?C*(*Aassign x a*) **by** *simp* **qed**
next
  case (*If P b* c_1 *Q* c_2)
  from *If*(*3*) **obtain** *C1* **where** *strip1*: *strip C1* = c_1 **and** *vc1*: *vc C1 Q*
    and *pre1*: $(\bigwedge s.\ pre\ C1\ Q\ s \leq P\ s + \uparrow (bval\ b\ s))$ **by** *blast*
  from *If*(*4*) **obtain** *C2* **where** *strip2*: *strip C2* = c_2 **and** *vc2*: *vc C2 Q*
    and *pre2*: $(\bigwedge s.\ pre\ C2\ Q\ s \leq P\ s + \uparrow (\neg\ bval\ b\ s))$ **by** *blast*

  show *?case*
    apply(*rule exI*[**where** *x*=*IF b THEN C1 ELSE C2*], *safe*)
    subgoal using *strip1 strip2* **by** *auto*
    subgoal using *vc1 vc2* **by** *auto*
    subgoal for *s* **using** *pre1*[*of s*] *pre2*[*of s*] **by** *auto*
    done
next
  case (*Seq* P_1 c_1 P_2 c_2 P_3)
  from *Seq*(*3*) **obtain** *C1* **where** *strip1*: *strip C1* = c_1 **and** *vc1*: *vc C1* P_2
    and *pre1*: $(\forall s.\ pre\ C1\ P_2\ s \leq P_1\ s)$ **by** *blast*
  from *Seq*(*4*) **obtain** *C2* **where** *strip2*: *strip C2* = c_2 **and** *vc2*: *vc C2* P_3
    and *pre2*: $(\forall s.\ pre\ C2\ P_3\ s \leq P_2\ s)$ **by** *blast*
  {
    fix *s*
    have $pre\ C1\ (pre\ C2\ P_3)\ s \leq P_1\ s$
      apply(*rule order.trans*[**where** *b*=*pre C1* P_2 *s*])
       apply(*rule pre_mono*) **using** *pre2* **apply** *simp* **using** *pre1* **by** *simp*

} **note** *pre* = *this*
show *?case*
apply(*rule exI*[**where** *x*=*C1* ;; *C2*], *safe*)
subgoal using *strip1 strip2* **by** *simp*
subgoal using *vc1 vc2 vc_mono pre2* **by** *auto*
subgoal using *pre* **by** *auto*
done
next
case (*While I b c*)
from *While*(*2*) **obtain** *C* **where** *strip*: *strip C* = *c* **and** *vc*: *vc C* ($\lambda a.$ *I a* + *1*)
and *pre*: $\bigwedge s.$ *pre C* ($\lambda a.$ *I a* + *1*) *s* $\leq$ *I s* + $\uparrow$ (*bval b s*) **by** *blast*
show *?case*
apply(*rule exI*[**where** *x*={*I*} *WHILE b DO C*], *safe*)
subgoal using *strip* **by** *simp*
subgoal using *pre vc* **by** *auto*
subgoal by *simp*
done
next
case (*conseq P c Q P' Q'*)
then obtain *C* **where** *strip C* = *c* **and** *vc*: *vc C Q* **and** *pre*: $\bigwedge s.$ *pre C Q s* $\leq$ *P s* **by** *blast*

from *pre_mono*[*OF conseq*(*3*)] **have** *1*: $\bigwedge s.$ *pre C Q' s* $\leq$ *pre C Q s* **by** *auto*

show *?case*
apply(*rule exI*[**where** *x*=*C*])
apply *safe*
apply *fact*
subgoal using *vc conseq*(*3*) *vc_mono* **by** *auto*
subgoal using *pre conseq*(*2*) *1* **using** *order.trans* **by** *metis*
done
qed

end

5.6 Examples

theory *Quant_Examples*

imports *Quant_VCG*
begin

fun *sum* :: *int* ⇒ *int* **where**
sum i = (*if i* ≤ *0 then 0 else sum* (*i* − *1*) + *i*)

abbreviation *wsum* ==
  WHILE Less (*N 0*) (*V ''x''*)
  DO (*''y''* ::= *Plus* (*V ''y''*) (*V ''x''*);;
      ''x'' ::= *Plus* (*V ''x''*) (*N* (− *1*)))

lemma *example*: $\vdash_2$ {*λs. enat* (*2* + *3∗n*) + *emb* (*s ''x''* = *int n*)} *''y''* ::=
N 0;; *wsum* {*λs. 0* }
apply(*rule Seq*)
 prefer *2*
 apply(*rule conseq*)
 apply(*rule While*[**where** *I*=*λs. enat* (*3* ∗ *nat* (*s ''x''*))])
  apply(*rule Seq*)
   prefer *2*
      apply(*rule Assign*)
      apply(*rule Assign'*)
     apply(*simp*)
     apply(*safe*) **subgoal for** *s* **apply**(*cases 0* < *s ''x''*) **apply**(*simp*)
   apply (*smt Suc_eq_plus1 Suc_nat_eq_nat_zadd1 distrib_left_numeral eSuc_numeral enat_numeral eq_iff iadd_Suc_right nat_mult_1_right one_add_one plus_1_eSuc*(*1*) *plus_enat_simps*(*1*) *semiring_norm*(*5*))
    apply(*simp*) **done**
    apply *blast*
  apply *simp*
apply(*rule Assign'*)
apply *simp*
  apply(*safe*) **subgoal for** *s* **apply**(*cases s ''x''* = *int n*) **apply**(*simp*)
     apply (*simp add*: *eSuc_enat plus_1_eSuc*(*2*))
    apply *simp*
    done
  done

lemma *example_sound*: $\models_2$ {*λs. enat* (*2* + *3∗n*) + *emb* (*s ''x''* = *int n*)}
''y'' ::= *N 0*;; *wsum* {*λs. 0* }
apply(*rule hoare2_sound*) **apply** (*rule example*) **done**

5.6.1 Examples for the use of the VCG

abbreviation *Wsum* ==

```
  {λs. enat (3 * nat (s ''x''))} WHILE Less (N 0) (V ''x'')
  DO (''y'' ::= Plus (V ''y'') (V ''x'');;
      ''x'' ::= Plus (V ''x'') (N (- 1)))

lemma ⊢₂ {λs. enat (2 + 3*n) + emb (s ''x'' = int n)} ''y'' ::= N 0;;
wsum {λs. 0 }
proof -
  have ⊢₂ {λs. enat (2 + 3*n) + emb (s ''x'' = int n)} strip (''y'' ::= N
0;; Wsum) {λs. 0 }
    apply(rule vc_sound')
    subgoal
      apply simp
      apply(safe) subgoal for s apply(cases 0 < s ''x'')
         apply(simp)
       apply (smt Suc_eq_plus1 Suc_nat_eq_nat_zadd1 distrib_left_numeral
eSuc_numeral enat_numeral eq_iff iadd_Suc_right nat_mult_1_right one_add_one
plus_1_eSuc(1) plus_enat_simps(1) semiring_norm(5))
        apply(simp) done
      done
    subgoal
      apply simp
      apply(safe) subgoal for s apply(cases s ''x'' = int n) apply(simp)

         apply (simp add: eSuc_enat plus_1_eSuc(2))
        apply simp
        done
      done
    done
  then show ?thesis by simp
qed

end
```

6 Quantitative Hoare Logic (big-O style)

```
theory QuantK_Hoare
imports Big_StepT Complex_Main HOL-Library.Extended_Nat
begin


abbreviation eq a b == (And (Not (Less a b)) (Not (Less b a)))

type_synonym lvname = string
```

type_synonym *assn* = *state* ⇒ *bool*
type_synonym *qassn* = *state* ⇒ *enat*

The support of an assn2

abbreviation *state_subst* :: *state* ⇒ *aexp* ⇒ *vname* ⇒ *state*
(‹_[_'/_]› [*1000*,*0*,*0*] *999*)
where $s[a/x] == s(x := aval\ a\ s)$

fun *emb* :: *bool* ⇒ *enat* (‹↑›) **where**
emb False = ∞
| *emb True* = *0*

6.1 Definition of Validity

definition *hoare2o_valid* :: *qassn* ⇒ *com* ⇒ *qassn* ⇒ *bool*
(‹$\models_{2'}$'{(*1*_)}/ (_)/ {(*1*_)}› *50*) **where**
$\models_{2'} \{P\}\ c\ \{Q\} \longleftrightarrow (\exists k{>}0.\ (\forall s.\ \ P\ s < \infty \longrightarrow (\exists t\ p.\ ((c,s) \Rightarrow p \Downarrow t) \wedge enat\ k * P\ s \geq p + enat\ k * Q\ t)))$

6.2 Hoare Rules

inductive
hoareQ :: *qassn* ⇒ *com* ⇒ *qassn* ⇒ *bool* (‹$\vdash_{2'}$' ({(*1*_)}/ (_)/ {(*1*_)})› *50*)
where

Skip: $\vdash_{2'} \{\%s.\ eSuc\ (P\ s)\}\ SKIP\ \{P\}$ |

Assign: $\vdash_{2'} \{\lambda s.\ eSuc\ (P\ (s[a/x]))\}\ x{::=}a\ \{\ P\ \}$ |

If: $[\![\ \vdash_{2'} \{\lambda s.\ P\ s + \uparrow(\ bval\ b\ s)\}\ c_1\ \{\ Q\};$
$\vdash_{2'} \{\lambda s.\ P\ s + \uparrow(\neg\ bval\ b\ s)\}\ c_2\ \{\ Q\}\ ]\!]$
$\Longrightarrow \vdash_{2'} \{\lambda s.\ eSuc\ (P\ s)\}\ IF\ b\ THEN\ c_1\ ELSE\ c_2\ \{\ Q\ \}$ |

Seq: $[\![\ \vdash_{2'} \{\ P_1\ \}\ c_1\ \{\ P_2\ \};\ \vdash_{2'} \{P_2\}\ c_2\ \{\ P_3\ \}]\!] \Longrightarrow \vdash_{2'} \{P_1\}\ c_1;;c_2\ \{P_3\}$
|

While:
$[\![\ \ \vdash_{2'} \{\ \%s.\ I\ s + \uparrow(bval\ b\ s)\ \}\ c\ \{\ \%t.\ I\ t + 1\ \}\ \ ]\!]$
$\Longrightarrow \vdash_{2'} \{\lambda s.\ I\ s + 1\ \}\ WHILE\ b\ DO\ c\ \{\lambda s.\ \ I\ s + \uparrow(\neg\ bval\ b\ s)\ \ \}$ |

conseq: $[\![\ \vdash_{2'} \{P\}c\{Q\}\ ;\ \bigwedge s.\ P\ s \leq enat\ k *\ \ P'\ s\ ;\ \bigwedge s.\ enat\ k * Q'\ s \leq Q\ s;\ k{>}0\ ]\!] \Longrightarrow$
$\vdash_{2'} \{P'\}c\{\ Q'\}$

Derived Rules

lemma *const*: $\llbracket \vdash_{2'} \{\lambda s.\ enat\ k * P\ s\}c\{\lambda s.\ enat\ k * Q\ s\};\ \ k{>}0\ \rrbracket \Longrightarrow$
$\vdash_{2'} \{P\}c\{\ Q\}$
apply(*rule conseq*) **by** *auto*

inductive
hoareQ' :: *qassn* ⇒ *com* ⇒ *qassn* ⇒ *bool* (‹$\vdash_Z$ ({(1_)}/ (_)/ {(1_)})›
50)
where

ZSkip: $\vdash_Z \{\%s.\ eSuc\ (P\ s)\}\ SKIP\ \{P\}$ |

ZAssign: $\vdash_Z \{\lambda s.\ eSuc\ (P\ (s[a/x]))\}\ x{::=}a\ \{\ P\ \}$ |

ZIf: $\llbracket \vdash_Z \{\lambda s.\ P\ s + \uparrow(\ bval\ b\ s)\}\ c_1\ \{\ Q\};$
$\vdash_Z \{\lambda s.\ P\ s + \uparrow(\neg\ bval\ b\ s)\}\ c_2\ \{\ Q\}\ \rrbracket$
$\Longrightarrow \vdash_Z \{\lambda s.\ eSuc\ (P\ s)\}\ IF\ b\ THEN\ c_1\ ELSE\ c_2\ \{\ Q\ \}$ |

ZSeq: $\llbracket \vdash_Z \{\ P_1\ \}\ c_1\ \{\ P_2\ \}; \vdash_Z \{P_2\}\ c_2\ \{\ P_3\ \}\rrbracket \Longrightarrow \vdash_Z \{P_1\}\ c_1;;c_2\ \{P_3\}$
|

ZWhile:
$\llbracket\ \ \vdash_Z \{\ \%s.\ I\ s + \uparrow(bval\ b\ s)\ \}\ c\ \{\ \%t.\ I\ t + 1\ \}\ \ \rrbracket$
$\Longrightarrow \vdash_Z \{\lambda s.\ I\ s + 1\ \}\ WHILE\ b\ DO\ c\ \{\lambda s.\ \ I\ s + \uparrow(\neg\ bval\ b\ s)\ \ \}$ |

Zconseq': $\llbracket \vdash_Z \{P\}c\{Q\}\ ;\ \bigwedge s.\ P\ s \le\ \ P'\ s\ ;\ \bigwedge s.\ Q'\ s \le Q\ s\ \rrbracket \Longrightarrow$
$\vdash_Z \{P'\}c\{\ Q'\}$ |

Zconst: $\llbracket \vdash_Z \{\lambda s.\ enat\ k * P\ s\}c\{\lambda s.\ enat\ k * Q\ s\};\ \ k{>}0\ \rrbracket \Longrightarrow$
$\vdash_Z \{P\}c\{\ Q\}$

lemma *Zconseq*: $\llbracket \vdash_Z \{P\}c\{Q\}\ ;\ \bigwedge s.\ P\ s \le enat\ k *\ \ P'\ s\ ;\ \bigwedge s.\ enat\ k *$
$Q'\ s \le\ Q\ s;\ k{>}0\ \rrbracket \Longrightarrow$
$\vdash_Z \{P'\}c\{\ Q'\}$
apply(*rule Zconst*[*of k P' c Q'*])
apply(*rule Zconseq'*[**where** *P*=*P* **and** *Q*=*Q*]) **by** *auto*

lemma *ZQ*: $\vdash_Z \{\ P\ \}\ c\ \{\ Q\ \} \Longrightarrow \vdash_{2'} \{\ P\ \}\ c\ \{\ Q\ \}$
apply(*induct rule*: *hoareQ'.induct*)
apply (*auto simp*: *hoareQ.Skip hoareQ.Assign hoareQ.If hoareQ.Seq*
hoareQ.While)

subgoal using *conseq*[**where** *k=1*] **using** *one_enat_def* **by** *auto*
subgoal for *k P c Q* **using** *const* **by** *auto*
done

lemma *QZ*: $\vdash_{2'} \{ P \}\ c\ \{ Q \} \Longrightarrow \vdash_Z \{ P \}\ c\ \{ Q \}$
apply(*induct rule*: *hoareQ.induct*)
apply (*auto simp*: *ZSkip ZAssign ZIf ZSeq ZWhile*)
using *Zconseq* **by** *blast*

lemma *QZ_iff*: $\vdash_{2'} \{ P \}\ c\ \{ Q \} \longleftrightarrow \vdash_Z \{ P \}\ c\ \{ Q \}$
using *ZQ QZ* **by** *metis*

6.3 Soundness

lemma *enatSuc0*[*simp*]: *enat* (*Suc 0*) ∗ *x* = *x*
using *one_enat_def* **by** *auto*

theorem *hoareQ_sound*: $\vdash_{2'} \{P\}c\{ Q\} \Longrightarrow\ \models_{2'} \{P\}c\{ Q\}$
apply(*unfold hoare2o_valid_def*)
proof(*induction rule*: *hoareQ.induct*)
case (*Skip P*)
show *?case* **apply**(*rule exI*[**where** *x=1*]) **apply**(*auto*)
subgoal for *s* **apply**(*rule exI*[**where** *x=s*]) **apply**(*rule exI*[**where** *x=Suc 0*])
apply *safe*
apply *fast*
by (*metis add.left_neutral add.right_neutral eSuc_enat iadd_Suc le_iff_add zero_enat_def*)
done
next
case (*Assign P a x*)
show *?case* **apply**(*rule exI*[**where** *x=1*]) **apply**(*auto*)
subgoal for *s* **apply**(*rule exI*[**where** *x=s*[*a/x*]]) **apply**(*rule exI*[**where** *x=Suc 0*])
apply *safe*
apply *fast*
by (*metis add.left_neutral add.right_neutral eSuc_enat iadd_Suc le_iff_add zero_enat_def*)
done
next
case (*Seq P1 C1 P2 C2 P3*)
from *Seq*(*3*) **obtain** *k1* **where** *Seq3*: $\forall s.\ P1\ s < \infty \longrightarrow (\exists t\ p.\ (C1,\ s) \Rightarrow p \Downarrow t \wedge enat\ p + enat\ k1 * P2\ t \le enat\ k1 * P1\ s)$ **and** *10*: *k1>0* **by** *blast*

from *Seq(4)* **obtain** *k2* **where** *Seq4*: $\forall s.\ P2\ s < \infty \longrightarrow (\exists t\ p.\ (C2,\ s) \Rightarrow p \Downarrow t \wedge enat\ p + enat\ k2 * P3\ t \leq enat\ k2 * P2\ s)$ **and** *20*: $k2>0$ **by** *blast*
let $?k = lcm\ k1\ k2$
show *?case* **apply**(*rule exI*[**where** $x=?k$])
proof (*safe*)
from *10 20* **show** $lcm\ k1\ k2>0$ **by** (*auto simp*: *lcm_pos_nat*)
fix *s*
assume *ninfP1*: $P1\ s < \infty$
with *Seq3* **obtain** *t1 p1* **where** *1*: $(C1,\ s) \Rightarrow p1 \Downarrow t1$ **and** *q1*: $enat\ p1 + k1 * P2\ t1 \leq k1 * P1\ s$ **by** *blast*
with *ninfP1* **have** *ninfP2*: $P2\ t1 < \infty$
using *not_le 10* **by** *fastforce*
with *Seq4* **obtain** *t2 p2* **where** *2*: $(C2,\ t1) \Rightarrow p2 \Downarrow t2$ **and** *q2*: $enat\ p2 + k2 * P3\ t2 \leq k2 * P2\ t1$ **by** *blast*
with *ninfP2* **have** *ninfP3*: $P3\ t2 < \infty$
using *not_le 20* **by** *fastforce*
then obtain *u2* **where** *u2*: $P3\ t2 = enat\ u2$ **by** *auto*
from *ninfP2* **obtain** *u1* **where** *u1*: $P2\ t1 = enat\ u1$ **by** *auto*
from *ninfP1* **obtain** *u0* **where** *u0*: $P1\ s = enat\ u0$ **by** *auto*

from *Big_StepT.Seq*[*OF 1 2*] **have** *12*: $(C1;;\ C2,\ s) \Rightarrow p1 + p2 \Downarrow t2$ **by** *simp*

have *i*: $(C1;;\ C2,\ s) \Rightarrow p1+p2 \Downarrow t2$ **using** *1* **and** *2* **by** *auto*

from *10 20* **have** *p*: $k1\ div\ gcd\ k1\ k2 > 0\ k2\ div\ gcd\ k1\ k2 > 0$
by (*simp_all add*: *div_greater_zero_iff*)

have *za*: $?k = (k1\ div\ gcd\ k1\ k2) * k2$
apply(*simp only*: *lcm_nat_def*)
by (*simp add*: *dvd_div_mult*)

have *za2*: $?k = (k2\ div\ gcd\ k1\ k2) * k1$
apply(*simp only*: *lcm_nat_def*)
by (*metis dvd_div_mult gcd_dvd2 mult.commute*)

from *q1*[*unfolded u1 u2 u0*] **have** *z*: $p1 + k1 * u1 \leq k1 * u0$ **by** *auto*
from *q2*[*unfolded u1 u2 u0*] **have** *y*: $p2 + k2 * u2 \leq k2 * u1$ **by** *auto*
have $p1+p2 + ?k * u2 \leq p1 + (k1\ div\ gcd\ k1\ k2)*p2 + ?k * u2$ **using** *p* **by** *simp*
also have $\ldots \leq (k2\ div\ gcd\ k1\ k2)*p1 + (k1\ div\ gcd\ k1\ k2)*p2 + ?k * u2$ **using** *p* **by** *simp*

also have ... = (*k2 div gcd k1 k2*)∗*p1* + (*k1 div gcd k1 k2*)∗(*p2* + *k2*∗ *u2*)
apply(*simp only*: *za*) **by** *algebra*
also have ... ≤ (*k2 div gcd k1 k2*)∗*p1* + (*k1 div gcd k1 k2*)∗(*k2* ∗ *u1*) **using** *y*
by (*metis add_left_mono distrib_left le_iff_add*)
also have ... = (*k2 div gcd k1 k2*)∗*p1* + *?k* ∗ *u1* **by**(*simp only*: *za*)
also have ... = (*k2 div gcd k1 k2*)∗*p1* + (*k2 div gcd k1 k2*) ∗(*k1*∗ *u1*) **by**(*simp only*: *za2*)
also have ... ≤ (*k2 div gcd k1 k2*)∗(*p1* + *k1*∗*u1*)
by (*simp add*: *distrib_left*)
also have ... ≤ (*k2 div gcd k1 k2*)∗(*k1* ∗ *u0*) **using** *z*
by *fastforce*
also have ... ≤ *?k* ∗ *u0* **by**(*simp only*: *za2*)
finally
have *p1*+*p2* + *?k* ∗ *u2* ≤ *?k* ∗ *u0* **.**
then have *ii*: *enat* (*p1*+*p2*) + *?k* ∗ *P3 t2* ≤ *?k* ∗ *P1 s*
unfolding *u0 u2* **by** *auto*

from *i ii* **show** ∃ *t p*. (*C1*;; *C2*, *s*) ⇒ *p* ⇓ *t* ∧ *enat p* + *?k* ∗ *P3 t* ≤ *?k* ∗ *P1 s* **by** *blast*
qed
next
case (*If P b c1 Q c2*)
from *If*(*3*) **obtain** *kT* **where** *If3*: ∀ *s*. *P s* + ↑ (*bval b s*) < ∞ ⟶ (∃ *t p*. (*c1*, *s*) ⇒ *p* ⇓ *t* ∧ *enat p* + *enat kT* ∗ *Q t* ≤ *enat kT* ∗ (*P s* + ↑ (*bval b s*))) **and** *T*: *kT* > *0* **by** *blast*
from *If*(*4*) **obtain** *kF* **where** *If4*: ∀ *s*. *P s* + ↑ (¬ *bval b s*) < ∞ ⟶ (∃ *t p*. (*c2*, *s*) ⇒ *p* ⇓ *t* ∧ *enat p* + *enat kF* ∗ *Q t* ≤ *enat kF* ∗ (*P s* + ↑ (¬ *bval b s*))) **and** *F*: *kF* > *0* **by** *blast*
show *?case* **apply**(*rule exI*[**where** *x*=*kT*∗*kF*])
proof (*safe*)
from *T F* **show** *0* < *kT* ∗ *kF* **by** *auto*
fix *s*
assume *eSuc* (*P s*) < ∞
then have *i*: *P s* < ∞
using *enat_ord_simps*(*4*) **by** *fastforce*
then obtain *u0* **where** *u0*: *P s* = *enat u0* **by** *auto*
show ∃ *t p*. (*IF b THEN c1 ELSE c2*, *s*) ⇒ *p* ⇓ *t* ∧ *enat p* + *enat* (*kT* ∗ *kF*) ∗ *Q t* ≤ *enat* (*kT* ∗ *kF*) ∗ *eSuc* (*P s*)
proof(*cases bval b s*)
case *True*
with *i* **have** *P s* + *emb* (*bval b s*) < ∞ **by** *simp*
with *If3* **obtain** *p t* **where** *1*: (*c1*, *s*) ⇒ *p* ⇓ *t* **and** *q*: *enat p* + *enat*

$kT * Q\ t \leq enat\ kT * (P\ s + emb\ (bval\ b\ s))$ **by** *blast*
from *Big_StepT.IfTrue[OF True 1]* **have** *2*: $(IF\ b\ THEN\ c1\ ELSE\ c2,\ s) \Rightarrow p + 1 \Downarrow t$ **by** *simp*

from *q* **have** $Q\ t < \infty$ **using** *i T True*
using *less_irrefl* **by** *fastforce*
then obtain *u1* **where** *u1*: $Q\ t = enat\ u1$ **by** *auto*
from *q True* **have** q': $p + kT * u1 \leq kT * u0$ **unfolding** *u0 u1* **by** *auto*
have $(p+1) + (kT * kF) * u1 \leq kF*(p+1) + (kT * kF) * u1$ **using** *F*
by *(simp add: mult_eq_if)*
also have $\ldots \leq kF*(p+1 + kT * u1)$
by *(simp add: add_mult_distrib2)*
also have $\ldots \leq kF*(1 + kT * u0)$
using q' **by** *auto*
also have $\ldots \leq (kT * kF) * Suc\ u0$ **using** *T* **by** *simp*
finally
have $(p+1) + (kT * kF) * u1 \leq (kT * kF) * Suc\ u0$ **.**
then have *1*: $enat\ (p+1) + enat\ (kT * kF) * Q\ t \leq enat\ (kT * kF) * eSuc\ (P\ s)$
unfolding *u1 u0* **by** *(simp add: eSuc_enat)*
from *1 2* **show** *?thesis* **by** *metis*
next
case *False*
with *i* **have** $P\ s + emb\ (\sim bval\ b\ s) < \infty$ **by** *simp*
with *If4* **obtain** *p t* **where** *1*: $(c2,\ s) \Rightarrow p \Downarrow t$ **and** *q*: $enat\ p + enat\ kF * Q\ t \leq enat\ kF * (P\ s + emb\ (\sim bval\ b\ s))$ **by** *blast*
from *Big_StepT.IfFalse[OF False 1]* **have** *2*: $(IF\ b\ THEN\ c1\ ELSE\ c2,\ s) \Rightarrow p + 1 \Downarrow t$ **by** *simp*

from *q* **have** $Q\ t < \infty$ **using** *i F False*
using *less_irrefl* **by** *fastforce*
then obtain *u1* **where** *u1*: $Q\ t = enat\ u1$ **by** *auto*
from *q False* **have** q': $p + kF * u1 \leq kF * u0$ **unfolding** *u0 u1* **by** *auto*
have $(p+1) + (kF * kT) * u1 \leq kT*(p+1) + (kF * kT) * u1$ **using** *T*
by *(simp add: mult_eq_if)*
also have $\ldots \leq kT*(p+1 + kF * u1)$
by *(simp add: add_mult_distrib2)*
also have $\ldots \leq kT*(1 + kF * u0)$
using q' **by** *auto*
also have $\ldots \leq (kF * kT) * Suc\ u0$ **using** *F* **by** *simp*

finally
have $(p+1) + (kT * kF) * u1 \le (kT * kF) * Suc\ u0$
by (*simp add*: *mult.commute*)
then have *1*: $enat\ (p+1) + enat\ (kT * kF) * Q\ t \le enat\ (kT * kF) * eSuc\ (P\ s)$
unfolding *u1 u0* **by** (*simp add*: *eSuc_enat*)
from *1 2* **show** *?thesis* **by** *metis*
qed
qed
next
case (*conseq P c Q k1 P′ Q′*)
from *conseq*(*5*) **obtain** *k* **where** *c4*: $\forall s.\ P\ s < \infty \longrightarrow (\exists t\ p.\ (c, s) \Rightarrow p \Downarrow t \wedge enat\ p + enat\ k * Q\ t \le enat\ k * P\ s)$ **and** *0*: $k>0$ **by** *blast*
show *?case* **apply**(*rule exI*[**where** $x=k*k1$])
proof (*safe*)
show $k*k1>0$ **using** *0 conseq*(*4*) **by** *auto*
fix *s*
assume $P'\ s < \infty$
with *conseq*(*2*,*4*) **have** $P\ s < \infty$
using *le_less_trans*
by (*metis enat.distinct*(*2*) *enat_ord_simps*(*4*) *imult_is_infinity*)
with *c4* **obtain** *p t* **where** *1*: $(c, s) \Rightarrow p \Downarrow t$ **and** *2*: $enat\ p + enat\ k * Q\ t \le enat\ k * P\ s$ **by** *blast*

have $enat\ p + enat\ (k*k1) * Q'\ t = enat\ p + enat\ (k) * ((enat\ k1) * Q'\ t)$
by (*metis mult.assoc times_enat_simps*(*1*))
also have $\ldots \le enat\ p + enat\ (k) * Q\ t$ **using** *conseq*(*3*)
by (*metis add_left_mono distrib_left le_iff_add*)
also have $\ldots \le enat\ k * P\ s$ **using** *2* **by** *auto*
also have $\ldots \le enat\ (k*k1) * P'\ s$ **using** *conseq*(*2*)
by (*metis mult.assoc mult_left_mono not_less not_less_zero times_enat_simps*(*1*))
finally have *2*: $enat\ p + enat\ (k*k1) * Q'\ t \le enat\ (k*k1) * P'\ s$
by *auto*
from *1 2* **show** $\exists t\ p.\ (c, s) \Rightarrow p \Downarrow t \wedge enat\ p + (k*k1) * Q'\ t \le (k*k1) * P'\ s$ **by** *auto*
qed
next
case (*While INV b c*)
then obtain *k* **where** *W2*: $\forall s.\ INV\ s + \uparrow (bval\ b\ s) < \infty \longrightarrow (\exists t\ p.\ (c, s) \Rightarrow p \Downarrow t \wedge enat\ p + enat\ k * (INV\ t + 1) \le enat\ k * (INV\ s + \uparrow (bval\ b\ s)))$ **and** *g0*: $k>0$
by *blast*
show *?case* **apply**(*rule exI*[**where** $x=k$])

proof (*safe*)
show *0<k* **by** *fact*
fix *s*
assume *ninfINV*: *INV s + 1 < ∞*
then have *f*: *INV s < ∞*
using *enat_ord_simps(4)* **by** *fastforce*
then obtain *n* **where** *i*: *INV s = enat n* **using** *not_infinity_eq*
by *auto*

have *INV s = enat n ⟹ ∃ t p. (WHILE b DO c, s) ⇒ p ⇓ t ∧ enat p + enat k ∗ (INV t + emb (¬ bval b t)) ≤ enat k ∗ (INV s + 1)*
proof (*induct n arbitrary*: *s rule*: *less_induct*)
case (*less n*)

then show *?case*
proof (*cases bval b s*)
case *False*
show *?thesis*
apply(*rule exI*[**where** *x=s*])
apply(*rule exI*[**where** *x=Suc 0*])
apply *safe*
apply (*fact WhileFalse*[*OF False*])
using *False*
apply (*simp add*: *one_enat_def*) **using** *g0*
by (*metis One_nat_def Suc_ile_eq add.commute add_left_mono distrib_left enat_0_iff(2) mult.right_neutral not_gr_zero one_enat_def*)

next
case *True*
with *less(2) W2* **have** (*∃ t p. (c, s) ⇒ p ⇓ t ∧ enat p + enat k ∗ (INV t + 1) ≤ enat k ∗ INV s*)
by *force*
then obtain *t p* **where** *o*: (*c, s*) *⇒ p ⇓ t* **and** *q*: *enat p + enat k ∗ (INV t + 1) ≤ enat k ∗ INV s* **by** *auto*
from *o bigstep_progress* **have** *p*: *p > 0* **by** *blast*

from *q* **have** *pf*: *enat k ∗ (INV t + 1) ≤ enat k ∗ INV s*
using *dual_order.trans* **by** *fastforce*
then have *INV t < ∞* **using** *less(2)*
using *g0 not_le* **by** *fastforce*
then obtain *invt* **where** *invt*: *INV t = enat invt* **by** *auto*
from *pf g0* **have** *g*: *INV t < INV s*
unfolding *less(2) invt*

by (*metis* (*full_types*) *Suc_ile_eq add.commute eSuc_enat enat_ord_simps*(*1*) *nat_mult_le_cancel_disj plus_1_eSuc*(*1*) *times_enat_simps*(*1*))

then have *ninfINVt*: $INV\ t < \infty$ **using** *less*(*2*)
using *enat_ord_simps*(*4*) **by** *fastforce*
then obtain n' **where** *i*: $INV\ t = enat\ n'$ **using** *not_infinity_eq*
by *auto*
with *less*(*2*) **have** *ii*: $n' < n$
using *g* **by** *auto*
from *i ii less*(*1*) **obtain** *t2 p2* **where** *o2*: $(WHILE\ b\ DO\ c,\ t) \Rightarrow p2 \Downarrow t2$ **and** *q2*: $enat\ p2 + enat\ k * (INV\ t2 + emb\ (\neg\ bval\ b\ t2)) \leq enat\ k * (\ INV\ t + 1)$ **by** *blast*
have *ende*: ~ *bval b t2*
apply(*rule ccontr*) **apply**(*simp*) **using** *q2 g0 ninfINVt*
by (*simp add*: *i one_enat_def*)
from *WhileTrue*[*OF True o o2*] **have** $(WHILE\ b\ DO\ c,\ s) \Rightarrow 1 + p + p2 \Downarrow t2$ **by** *simp*

from *ende q2* **have** *q2'*: $enat\ p2 + enat\ k * INV\ t2 \leq enat\ k * (INV\ t + 1)$ **by** *simp*

show *?thesis*
apply(*rule exI*[**where** *x=t2*])
apply(*rule exI*[**where** $x= 1 + p + p2$])
apply(*safe*)
apply(*fact*)
using *ende* **apply**(*simp*)
proof −
have $enat\ (Suc\ (p + p2)) + enat\ k * INV\ t2 = enat\ (Suc\ p) + enat\ p2 + enat\ k * INV\ t2$ **by** *fastforce*
also have $\ldots \leq enat\ (Suc\ p) + enat\ k * (INV\ t + 1)$ **using** *q2'*
by (*metis ab_semigroup_add_class.add_ac*(*1*) *add_left_mono*)
also have $\ldots \leq 1 + enat\ k * (INV\ s)$ **using** *q*
by (*metis* (*no_types*, *opaque_lifting*) *add.commute add_left_mono eSuc_enat iadd_Suc plus_1_eSuc*(*1*))
also have $\ldots \leq enat\ k + enat\ k * (INV\ s)$ **using** *g0*
by (*simp add*: *Suc_leI one_enat_def*)
also have $\ldots \leq enat\ k * (INV\ s + 1)$
by (*simp add*: *add.commute distrib_left*)
finally show $enat\ (Suc\ (p + p2)) + enat\ k * INV\ t2 \leq enat\ k * (INV\ s + 1)$ **.**
qed
qed

qed

from *this*[*OF i*] **show** $\exists\, t\ p.\ (WHILE\ b\ DO\ c,\ s) \Rightarrow p \Downarrow t \wedge enat\ p + enat\ k * (INV\ t + emb\ (\neg\ bval\ b\ t)) \le enat\ k * (INV\ s + 1)$ **.**

qed
qed

lemma *conseq'*:
$[\![\vdash_{2'} \{P\}\ c\ \{Q\}\ ;\ \forall s.\ P\ s \le P'\ s; \forall s.\ Q'\ s \le Q\ s\ ]\!] \Longrightarrow \vdash_{2'} \{P'\}\ c\ \{Q'\}$
apply(*rule conseq*[**where** *k=1*]) **by** *auto*

lemma *strengthen_pre*:
$[\![\forall s.\ P\ s \le P'\ s;\ \vdash_{2'} \{P\}\ c\ \{Q\}\ ]\!] \Longrightarrow \vdash_{2'} \{P'\}\ c\ \{Q\}$
apply(*rule conseq*[**where** *k=1* **and** *Q'=Q* **and** *Q=Q*]) **by** *auto*

lemma *weaken_post*:
$[\![\vdash_{2'} \{P\}\ c\ \{Q\};\ \forall s.\ Q\ s \ge Q'\ s\ ]\!] \Longrightarrow\ \vdash_{2'} \{P\}\ c\ \{Q'\}$
apply(*rule conseq*[**where** *k=1*]) **by** *auto*

lemma *Assign'*: $\forall s.\ P\ s \ge eSuc\ (\ Q(s[a/x])) \Longrightarrow \vdash_{2'} \{P\}\ x ::= a\ \{Q\}$
by (*simp add*: *strengthen_pre*[*OF _ Assign*])

6.4 Completeness

lemma *bigstep_det*: $(c1,\ s) \Rightarrow p1 \Downarrow t1 \Longrightarrow (c1,\ s) \Rightarrow p \Downarrow t \Longrightarrow p1=p \wedge t1=t$
using *big_step_t_determ2* **by** *simp*

lemma *bigstepT_the_cost*: $(c,\ s) \Rightarrow P \Downarrow T \Longrightarrow (THE\ n.\ \exists a.\ (c,\ s) \Rightarrow n \Downarrow a) = P$
using *bigstep_det* **by** *blast*

lemma *bigstepT_the_state*: $(c,\ s) \Rightarrow P \Downarrow T \Longrightarrow (THE\ a.\ \exists n.\ (c,\ s) \Rightarrow n \Downarrow a) = T$
using *bigstep_det* **by** *blast*

lemma *SKIPnot*: $(\neg\ (SKIP,\ s) \Rightarrow p \Downarrow t) = (s \neq t \vee p \neq Suc\ 0)$ **by** *blast*

lemma *SKIPp*: $(THE\ p.\ \exists t.\ (SKIP,\ s) \Rightarrow p \Downarrow t) = Suc\ 0$
apply(*rule the_equality*)

apply *fast*
apply *auto* **done**

lemma *SKIPt*: $(THE\ t.\ \exists\, p.\ (SKIP,\ s) \Rightarrow p \Downarrow t) = s$
apply(*rule the_equality*)
apply *fast*
apply *auto* **done**

lemma *ASSp*: $(THE\ p.\ Ex\ (big\_step\_t\ (x ::= e,\ s)\ p)) = Suc\ 0$
apply(*rule the_equality*)
apply *fast*
apply *auto* **done**

lemma *ASSt*: $(THE\ t.\ \exists\, p.\ (x ::= e,\ s) \Rightarrow p \Downarrow t) = s(x := aval\ e\ s)$
apply(*rule the_equality*)
apply *fast*
apply *auto* **done**

lemma *ASSnot*: $(\ \neg\ (x ::= e,\ s) \Rightarrow p \Downarrow t\ ) = (p \neq Suc\ 0 \lor t \neq s(x := aval\ e\ s))$
apply *auto* **done**

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

definition *wpQ* :: *com* ⇒ *qassn* ⇒ *qassn* (‹wp_Q›) **where**
$wp_Q\ c\ Q\ =\ (\lambda s.\ (if\ (\exists\, t\ p.\ (c,s) \Rightarrow p \Downarrow t \land Q\ t < \infty)\ \ then\ enat\ (THE\ p.\ \exists\, t.\ (c,s) \Rightarrow p \Downarrow t) + Q\ (THE\ t.\ \exists\, p.\ (c,s) \Rightarrow p \Downarrow t)\ else\ \infty))$

lemma *wpQ_skip*[*simp*]: $wp_Q\ SKIP\ Q = (\%s.\ eSuc\ (Q\ s))$
apply(*auto intro*!: *ext simp*: *wpQ_def*)
prefer *2*
apply(*simp only*: *SKIPnot*)
apply(*simp*)
apply(*simp only*: *SKIPp SKIPt*)
using *one_enat_def plus_1_eSuc*(*1*) **by** *auto*

lemma *wpQ_ass*[*simp*]: $wp_Q\ (x ::= e)\ Q = (\lambda s.\ eSuc\ (Q\ (s(x := aval\ e\ s))))$
by (*auto intro*!: *ext simp*: *wpQ_def ASSp ASSt ASSnot eSuc_enat*)

lemma *wpt_Seq*[*simp*]: $wp_Q\ (c_1;;c_2)\ Q = wp_Q\ c_1\ (wp_Q\ c_2\ Q)$
unfolding *wpQ_def*

proof (*rule*, *case_tac* $\exists\, t\ p.\ (c_1;;\ c_2,\ s) \Rightarrow p \Downarrow t \wedge Q\ t < \infty$, *goal_cases*)
case (*1 s*)
then obtain *u p* **where** *ter*: $(c_1;;\ c_2,\ s) \Rightarrow p \Downarrow u$ **and** *Q*: $Q\ u < \infty$ **by** *blast*
then obtain *t p1 p2* **where** *i*: $(c_1\ ,\ s) \Rightarrow p1 \Downarrow t$ **and** *ii*: $(c_2\ ,\ t) \Rightarrow p2 \Downarrow u$ **and** *p*: $p1 + p2 = p$ **by** *blast*

from *bigstepT_the_state*[*OF i*] **have** *t*: $(THE\ t.\ \exists\, p.\ (c_1,\ s) \Rightarrow p \Downarrow t) = t$
by *blast*
from *bigstepT_the_state*[*OF ii*] **have** *t2*: $(THE\ u.\ \exists\, p.\ (c_2,\ t) \Rightarrow p \Downarrow u) = u$
by *blast*
from *bigstepT_the_cost*[*OF i*] **have** *firstcost*: $(THE\ p.\ \exists\, t.\ (c_1,\ s) \Rightarrow p \Downarrow t) = p1$
by *blast*
from *bigstepT_the_cost*[*OF ii*] **have** *secondcost*: $(THE\ p.\ \exists\, u.\ (c_2,\ t) \Rightarrow p \Downarrow u) = p2$
by *blast*

have *totalcost*: $(THE\ p.\ Ex\ (big\_step\_t\ (c_1;;\ c_2,\ s)\ p)) = p1 + p2$
using *bigstepT_the_cost*[*OF ter*] *p* **by** *auto*
have *totalstate*: $(THE\ t.\ \exists\, p.\ (c_1;;\ c_2,\ s) \Rightarrow p \Downarrow t) = u$
using *bigstepT_the_state*[*OF ter*] **by** *auto*

have *c2*: $\exists\, ta\ p.\ (c_2,\ t) \Rightarrow p \Downarrow ta \wedge Q\ ta < \infty$
apply(*rule exI*[**where** *x= u*])
apply(*rule exI*[**where** *x= p2*]) **apply** *safe* **apply** *fact+* **done**

have *C*: $\exists\, t\ p.\ (c_1,\ s) \Rightarrow p \Downarrow t \wedge (if\ \exists\, ta\ p.\ (c_2,\ t) \Rightarrow p \Downarrow ta \wedge Q\ ta < \infty\ then\ enat\ (THE\ p.\ Ex\ (big\_step\_t\ (c_2,\ t)\ p)) + Q\ (THE\ ta.\ \exists\, p.\ (c_2,\ t) \Rightarrow p \Downarrow ta)\ else\ \infty) < \infty$
apply(*rule exI*[**where** *x=t*])
apply(*rule exI*[**where** *x=p1*])
apply *safe*
apply *fact*
apply(*simp only*: *c2 if_True*)
using *Q bigstepT_the_state ii* **by** *auto*

show *?case*
apply(*simp only*: *1 if_True t t2 c2 C totalcost totalstate firstcost secondcost*) **by** *fastforce*
next

case (*2 s*)
show *?case* **apply**(*simp only*: *2 if_False*)
apply *auto* **using** *2*
by *force*
qed

lemma *wpQ_If*[*simp*]:
wp_Q (*IF b THEN* c_1 *ELSE* c_2) *Q* = ($\lambda s.$ *eSuc* (wp_Q (*if bval b s then* c_1 *else* c_2) *Q s*))
apply (*auto simp*: *wpQ_def fun_eq_iff*)
subgoal for *x t p i ta ia xa* **apply**(*simp only*: *IfTrue*[*THEN bigstepT_the_state*])
apply(*simp only*: *IfTrue*[*THEN bigstepT_the_cost*])
apply(*simp only*: *bigstepT_the_cost bigstepT_the_state*)
by (*simp add*: *eSuc_enat*)
apply(*simp only*: *bigstepT_the_state bigstepT_the_cost*) **apply** *force*
apply(*simp only*: *bigstepT_the_state bigstepT_the_cost*)
proof(*goal_cases*)
case (*1 x t p i ta ia xa*)
note *f*= *IfFalse*[*THEN bigstepT_the_state*, *of b x* c_2 *xa ta Suc xa* c_1, *simplified*, *OF 1*(*4*) *1*(*5*)]
note *f2*= *IfFalse*[*THEN bigstepT_the_cost*, *of b x* c_2 *xa ta Suc xa* c_1, *simplified*, *OF 1*(*4*) *1*(*5*)]
note *g*= *bigstep_det*[*OF 1*(*1*) *1*(*5*)]
show *?case*
apply(*simp only*: *f f2*) **using** *1 g*
by (*simp add*: *eSuc_enat*)
next
case *2*
then
show *?case*
apply(*simp only*: *bigstepT_the_state bigstepT_the_cost*) **apply** *force*
done
qed

lemma *hoareQ_inf*: $\vdash_{2'}$ {%*s*. ∞} *c* { *Q*}
apply (*induction c arbitrary*: *Q*)
apply(*auto intro*: *hoareQ.Skip hoareQ.Assign hoareQ.Seq hoareQ.conseq*)
subgoal apply(*rule hoareQ.conseq*) **apply**(*rule hoareQ.If*[**where** *P*=%*s*. ∞]) **by**(*auto intro*: *hoareQ.If hoareQ.conseq*)
subgoal apply(*rule hoareQ.conseq*) **apply**(*rule hoareQ.While*[**where** *I*=%*s*. ∞]) **apply**(*rule hoareQ.conseq*) **by** *auto*
done

lemma assumes *b*: *bval b s*
shows *wpQ_WhileTrue*: wp_Q *c* (wp_Q (*WHILE b DO c*) *Q*) *s* + *1* ≤ wp_Q (*WHILE b DO c*) *Q s*
proof (*cases* ∃ *t p*. (*WHILE b DO c*, *s*) ⇒ *p* ⇓ *t* ∧ *Q t* < ∞)
case *True*
then obtain *t p* **where** *w*: (*WHILE b DO c*, *s*) ⇒ *p* ⇓ *t* **and** *q*: *Q t* < ∞ **by** *blast*
from *b w* **obtain** *p1 p2 t1* **where** *c*: (*c*, *s*) ⇒ *p1* ⇓ *t1* **and** *w'*: (*WHILE b DO c*, *t1*) ⇒ *p2* ⇓ *t* **and** *sum*: *1* + *p1* + *p2* = *p*
by *auto*
have *g*: ∃ *ta p*. (*WHILE b DO c*, *t1*) ⇒ *p* ⇓ *ta* ∧ *Q ta* < ∞
apply(*rule exI*[**where** *x*=*t*])
apply(*rule exI*[**where** *x*=*p2*])
apply *safe* **apply** *fact*+ **done**

have *h*: ∃ *t p*. (*c*, *s*) ⇒ *p* ⇓ *t* ∧ (*if* ∃ *ta p*. (*WHILE b DO c*, *t*) ⇒ *p* ⇓ *ta* ∧ *Q ta* < ∞ *then enat* (*THE p*. *Ex* (*big_step_t* (*WHILE b DO c*, *t*) *p*)) + *Q* (*THE ta*. ∃ *p*. (*WHILE b DO c*, *t*) ⇒ *p* ⇓ *ta*) *else* ∞) < ∞
apply(*rule exI*[**where** *x*=*t1*])
apply(*rule exI*[**where** *x*=*p1*])
apply *safe* **apply** *fact*
apply(*simp only*: *g if_True*) **using** *bigstepT_the_state bigstepT_the_cost w' q* **by**(*auto*)

have wp_Q *c* (wp_Q (*WHILE b DO c*) *Q*) *s* + *1* = *enat p* + *Q t*
unfolding *wpQ_def* **apply**(*simp only*: *h if_True*)
apply(*simp only*: *bigstepT_the_state*[*OF c*] *bigstepT_the_cost*[*OF c*] *g if_True bigstepT_the_state*[*OF w'*] *bigstepT_the_cost*[*OF w'*]) **using** *sum*
by (*metis One_nat_def ab_semigroup_add_class.add_ac*(*1*) *add.commute add.right_neutral eSuc_enat plus_1_eSuc*(*2*) *plus_enat_simps*(*1*))
also have ... = wp_Q (*WHILE b DO c*) *Q s*
unfolding *wpQ_def* **apply**(*simp only*: *True if_True*)
using *bigstepT_the_state bigstepT_the_cost w* **apply**(*simp*) **done**
finally show *?thesis* **by** *simp*
next
case *False*
have wp_Q (*WHILE b DO c*) *Q s* = ∞
unfolding *wpQ_def*
apply(*simp only*: *False if_False*) **done**
then show *?thesis* **by** *auto*
qed

lemma assumes *b*: ~ *bval b s*
shows *wpQ_WhileFalse*: *Q s* + *1* ≤ wp_Q (*WHILE b DO c*) *Q s*

proof (*cases* $\exists t\ p.$ (*WHILE b DO c*, *s*) $\Rightarrow p \Downarrow t \wedge Q\ t < \infty$)
  case *True*
  with *b* **obtain** *t p* **where** *w*: (*WHILE b DO c*, *s*) $\Rightarrow p \Downarrow t$ **and** $Q\ t < \infty$ **by** *blast*
  with *b* **have** *c*: *s=t p=Suc 0* **by** *auto*
  have wp_Q (*WHILE b DO c*) *Q s* = *Q s* + *1*
    unfolding *wpQ_def* **apply**(*simp only*: *True if_True*)
     using *w c bigstepT_the_cost bigstepT_the_state* **by**(*auto simp add*: *one_enat_def*)
  then show *?thesis* **by** *auto*
next
  case *False*
  have wp_Q (*WHILE b DO c*) *Q s* = ∞
    unfolding *wpQ_def*
    apply(*simp only*: *False if_False*) **done**
  then show *?thesis* **by** *auto*
qed

lemma *wpQ_is_pre*: $\vdash_{2'}$ {wp_Q *c Q*} *c* { *Q*}
proof (*induction c arbitrary*: *Q*)
  case *SKIP* **show** *?case* **apply** (*auto intro*: *hoareQ.Skip*) **done**
next
  case *Assign* **show** *?case* **apply** (*auto intro*:*hoareQ.Assign*) **done**
next
  case *Seq* **thus** *?case* **by** (*auto intro*:*hoareQ.Seq*)
next
  case (*If x1 c1 c2 Q*) **thus** *?case*
    apply (*auto intro*!: *hoareQ.If*)
     apply(*rule hoareQ.conseq*)
       apply(*auto*)
     apply(*rule hoareQ.conseq*)
      apply(*auto*)
    done
next
  case (*While b c*)
  show *?case*
      apply(*rule conseq*[**where** *k=1*])
      apply(*rule hoareQ.While*[**where** *I=%s.* (*if bval b s then* wp_Q *c* (wp_Q (*WHILE b DO c*) *Q*) *s else Q s*)])
      apply(*rule conseq*[**where** *k=1*])
        apply(*rule While*[*of* wp_Q (*WHILE b DO c*) *Q*])
       apply(*case_tac bval b s*)
        apply(*simp*) **apply**(*simp*)

```
    subgoal for s
      apply(cases bval b s)
      using wpQ_WhileTrue apply simp
      using wpQ_WhileFalse apply simp done
      apply simp
    subgoal for s
      apply(cases bval b s)
      using wpQ_WhileTrue apply simp
      using wpQ_WhileFalse apply simp done
     apply(case_tac bval b s)
      apply(simp) apply(simp)
      apply simp done
qed


lemma wpQ_is_pre': ⊢2' {wp_Q c (%s. enat k * Q s )} c {(%s. enat k * Q
s )}
  using wpQ_is_pre by blast

lemma wpQ_is_weakestprePotential1: ⊨2' {P}c{Q} ⟹ (∃k>0. ∀s. wp_Q
c (%s. enat k* Q s) s ≤ enat k * P s)
apply(auto simp: hoare2o_valid_def wpQ_def)
proof (goal_cases)
  case (1 k)
  show ?case
  proof (rule exI[where x=k], safe)
    show 0<k by fact
  next
    fix s t p i
    assume (c, s) ⇒ p ⇓ t enat k * Q t = enat i

    show enat (↓t (c, s)) + enat k * Q (↓s (c, s)) ≤ enat k * P s
    proof (cases P s < ∞)
      case True
      with 1 obtain t p' where i: (c, s) ⇒ p' ⇓ t and ii: enat p' + enat k
*  Q t ≤ enat k * P s
        by auto
     show ?thesis by(simp add: bigstepT_the_state[OF i] bigstepT_the_cost[OF
i] ii)
    next
      case False
      then show ?thesis
        using 1 by auto
    qed
```

next
  fix s
  assume $\forall t.\ (\forall p.\ \neg\ (c,\ s) \Rightarrow p \Downarrow t) \vee enat\ k * Q\ t = \infty$
  then show $enat\ k * P\ s = \infty$ **using** *1* **by** *force*
qed
qed

theorem *hoareQ_complete*: $\models_{2'} \{P\}c\{Q\} \Longrightarrow \vdash_{2'} \{P\}c\{\ Q\}$
proof –
  assume $\models_{2'} \{P\}c\{Q\}$
  with *wpQ_is_weakestprePotential1* **obtain** k **where** $k{>}0$
    and *1*: $\bigwedge s.\ wp_Q\ c\ (\lambda s.\ enat\ k * Q\ s)\ s \le enat\ k * P\ s$ **by** *blast*
  show $\vdash_{2'} \{P\}c\{Q\}$
    apply(*rule conseq*[*OF wpQ_is_pre'*])
    apply(*fact 1*)
    apply *simp* **by** *fact*
qed

theorem *hoareQ_complete'*: $\models_{2'} \{P\}c\{Q\} \Longrightarrow \vdash_{2'} \{P\}c\{\ Q\}$
  unfolding *hoare2o_valid_def*
proof –
  assume $\exists k{>}0.\ \forall s.\ P\ s < \infty \longrightarrow (\exists t\ p.\ (c,\ s) \Rightarrow p \Downarrow t \wedge enat\ p + enat\ k * Q\ t \le enat\ k * P\ s)$
  then obtain k **where** f: $\forall s.\ P\ s < \infty \longrightarrow (\exists t\ p.\ (c,\ s) \Rightarrow p \Downarrow t \wedge enat\ p + enat\ k * Q\ t \le enat\ k * P\ s)$ **and** k: $k{>}0$ **by** *auto*

  show $\vdash_{2'} \{P\}c\{\ Q\}$
    apply(*rule conseq*[*OF wpQ_is_pre'*, **where** $Q'{=}Q$, *simplified*, **where** $k1{=}k$ **and** $k{=}k$ **and** $Q1{=}Q$])
    unfolding *wpQ_def*
    subgoal for s
      proof(*cases* $P\ s < \infty$)
        case *True*
        with f **obtain** $t\ p'$ **where** i: $(c,\ s) \Rightarrow p' \Downarrow t$ **and** ii: $enat\ p' + enat\ k * Q\ t \le enat\ k * P\ s$
          by *auto*
        from *ii k True* **have** *iii*: $enat\ k * Q\ t < \infty$
          using *imult_is_infinity* **by** *fastforce*
        have *kla*: $\exists t\ p.\ (c,\ s) \Rightarrow p \Downarrow t \wedge enat\ k * Q\ t < \infty$
          using *iii i* **by** *auto*
        show *?thesis* **unfolding** *bigstepT_the_state*[*OF i*]
            unfolding *bigstepT_the_cost*[*OF i*]
            apply(*simp only*: *kla*) **using** *ii* **by** *simp*
      next

case *False*
then show *?thesis* **using** *k* **by** *auto*
qed
subgoal by *auto*
using *k* **by** *auto*
qed

corollary *hoareQ_sound_complete*: $\vdash_{2'} \{P\}c\{Q\} \longleftrightarrow \models_{2'} \{P\}c\{\ Q\}$
by (*metis hoareQ_sound hoareQ_complete*)

6.5 Example

lemma fixes *X::int* **assumes** $0 < X$ **shows**
Z: *eSuc* (*enat* (*nat* $(2 * X) *$ *nat* $(2 * X))) \leq$ *enat* $(5 *$ (*nat* $(X * X)))$
proof −
from *assms* **have** *nn*: $0 \leq X$ **by** *auto*
from *assms* **have** $0 <$ *nat* X **by** *auto*
then have $0 <$ *enat* (*nat* X) **by** (*simp add*: *zero_enat_def*)
then have A: *eSuc 0* $\leq$ *enat* (*nat* X) **using** *ileI1*
by *blast*

have (*nat* X) $\leq$ (*nat* $(X{*}X)$) **using** *nn nat_mult_distrib* **by** *auto*
then have D: *enat* (*nat* X) $\leq$ *enat* (*nat* $(X{*}X)$) **by** *auto*

have C: (*enat* (*nat* $(2 * X) *$ *nat* $(2 * X))) = 4*$ *enat* (*nat* $(X * X)$)
using *nn nat_mult_distrib*
by (*simp add*: *numeral_eq_enat*)
have *eSuc* (*enat* (*nat* $(2 * X) *$ *nat* $(2 * X)))$
$=$ *eSuc 0* $+$ (*enat* (*nat* $(2 * X) *$ *nat* $(2 * X)))$
using *one_eSuc plus_1_eSuc*(*1*) **by** *auto*
also have $\ldots \leq$ *enat* (*nat* X) $+$ (*enat* (*nat* $(2 * X) *$ *nat* $(2 * X)))$
using A *add_right_mono* **by** *blast*
also have $\ldots \leq$ *enat* (*nat* X) $+ 4*$ *enat* (*nat* $(X * X)$) **using** C **by** *auto*
also have $\ldots \leq$ *enat* (*nat* $(X * X)$) $+ 4*$ *enat* (*nat* $(X * X)$) **using** D
by *auto*
also have $\ldots = 5*$ *enat* (*nat* $(X * X)$)
by (*metis eSuc_numeral mult_eSuc semiring_norm*(*5*))
also have $\ldots =$ *enat* ($5*$ *nat* $(X * X)$)
by (*simp add*: *numeral_eq_enat*)
finally
show *?thesis* **.**
qed

lemma *weakenpre*: $[\![\vdash_{2'} \{P\}c\{Q\} ~;~~ (\forall s.~P~s \leq~~ P'~s) ~]\!] \Longrightarrow$
$\vdash_{2'} \{P'\}c\{~Q\}$ **using** *conseq*[**where** Q'=*Q* **and** *k*=*1*]
by *auto*

lemma *whileDecr*: $\vdash_{2'}$ { %*s*. *enat* (*nat* (*s* ''*x*'')) + *1*} *WHILE* (*Less* (*N 0*) (*V* ''*x*'')) *DO* (*SKIP*;; *SKIP*;; ''*x*'' ::= *Plus* (*V* ''*x*'') (*N* (−*1*))) { %*s*. *enat 0*}
apply(*rule conseq*[**where** *k*=*4*])
apply(*rule While*[**where** *I*=%*s*. *enat 4* ∗ (*enat* (*nat* (*s* ''*x*'')))])
prefer *2*
subgoal for *s* **apply**(*simp only*: *one_enat_def plus_enat_simps times_enat_simps enat_ord_code*(*1*)) **by** *presburger*
apply(*rule Seq*[**where** P_2=wp_Q (''*x*'' ::= *Plus* (*V* ''*x*'') (*N* (−*1*))) (λ*t*. *enat 4* ∗ *enat* (*nat* (*t* ''*x*'')) + *1*)])
apply(*simp*)
apply(*rule Seq*[**where** P_2=wp_Q (*SKIP*) (λ*s*. *eSuc* (*enat* (*4* ∗ *nat* (*s* ''*x*'' − *1*)) + *1*))])
apply *simp*
subgoal apply(*rule weakenpre*) **apply**(*rule Skip*) **apply** *auto*
subgoal for *s* **apply**(*cases s* ''*x*'' > *0*) **apply** *auto*
apply(*simp only*: *one_enat_def plus_enat_simps times_enat_simps enat_ord_code*(*1*) *eSuc_enat*) **done**
done
subgoal apply *simp* **apply**(*rule Skip*) **done**
subgoal apply *simp* **apply**(*rule weakenpre*) **apply**(*rule Assign*) **by** *simp*
apply *simp*
subgoal for *s* **apply**(*cases s* ''*x*'' > *0*) **by** *auto*
by *simp*

lemma *whileDecrIf*: $\vdash_{2'}$ { %*s*. *enat* (*nat* (*s* ''*x*'')) + *1*} *WHILE* (*Less* (*N 0*) (*V* ''*x*'')) *DO* ((*IF Less* (*N 0*) (*V* ''*z*'') *THEN SKIP*;; *SKIP ELSE SKIP*);; ''*x*'' ::= *Plus* (*V* ''*x*'') (*N* (−*1*))) { %*s*. *enat 0*}
apply(*rule conseq*[*OF While*, **where** *k*=*6* **and** *I1*=%*s*. *enat 6* ∗ (*enat* (*nat* (*s* ''*x*'')))])
prefer *2*
subgoal for *s* **apply**(*simp only*: *one_enat_def plus_enat_simps times_enat_simps enat_ord_code*(*1*)) **by** *presburger*
apply(*rule Seq*[**where** P_2=wp_Q (''*x*'' ::= *Plus* (*V* ''*x*'') (*N* (−*1*))) (λ*t*. *enat 6* ∗ *enat* (*nat* (*t* ''*x*'')) + *1*)])
apply(*simp*)
apply(*rule weakenpre*)
apply(*rule If*[**where** *P*=wp_Q (*IF Less* (*N 0*) (*V* ''*z*'') *THEN SKIP*;;

SKIP ELSE SKIP) (λs. eSuc (enat (6 ∗ nat (s ''x'' − 1)) + 1))])
subgoal
apply simp
apply(rule Seq[**where** P_2=wp_Q (SKIP) (λs. eSuc (enat (6 ∗ nat (s ''x'' − 1)) + 1))])
subgoal **apply**(rule weakenpre) **apply**(rule Skip) **by** auto
subgoal **apply**(rule weakenpre) **apply**(rule Skip) **by** auto
done
subgoal
apply simp
subgoal **apply**(rule weakenpre) **apply**(rule Skip) **by** auto
done
subgoal
apply auto
subgoal **for** s **apply**(cases s ''x'' > 0) **apply** auto
apply(simp only: one_enat_def plus_enat_simps times_enat_simps enat_ord_code(1) eSuc_enat) **done**
subgoal **for** s **apply**(cases s ''x'' > 0) **apply** auto
apply(simp only: one_enat_def plus_enat_simps times_enat_simps enat_ord_code(1) eSuc_enat) **done**
done
subgoal **apply** simp **apply**(rule weakenpre) **apply**(rule Assign) **by** simp
apply simp
subgoal **for** s **apply**(cases s ''x'' > 0) **by** auto
by simp

lemma whileDecrIf2: $\vdash_{2'}$ { %s. enat (nat (s ''x'')) + 1} WHILE (Less (N 0) (V ''x'')) DO ((IF Less (N 0) (V ''z'') THEN SKIP;; SKIP ELSE SKIP);; ''x'' ::= Plus (V ''x'') (N (−1))) { %s. enat 0}
apply(rule conseq[OF While, **where** k=6 **and** I1=%s. enat 6 ∗ (enat (nat (s ''x'')))])
apply(rule Seq[**where** P_2=wp_Q (''x'' ::= Plus (V ''x'') (N (−1))) (λt. enat 6 ∗ enat (nat (t ''x'')) + 1)])
apply(simp)
apply(rule weakenpre)
apply(rule If[**where** P=wp_Q (IF Less (N 0) (V ''z'') THEN SKIP;; SKIP ELSE SKIP) (λs. eSuc (enat (6 ∗ nat (s ''x'' − 1)) + 1))])
subgoal
apply simp
apply(rule Seq[**where** P_2=wp_Q (SKIP) (λs. eSuc (enat (6 ∗ nat (s ''x'' − 1)) + 1))])
subgoal **apply**(rule weakenpre) **apply**(rule Skip) **by** auto
subgoal **apply**(rule weakenpre) **apply**(rule Skip) **by** auto

done
subgoal
apply *simp*
subgoal **apply**(*rule weakenpre*) **apply**(*rule Skip*) **by** *auto*
done
prefer *2*
subgoal **apply** *simp* **apply**(*rule weakenpre*) **apply**(*rule Assign*) **by** *simp*
subgoal
apply *auto*
subgoal **for** *s* **apply**(*cases s ''x'' > 0*) **apply** *auto*
apply(*simp only*: *one_enat_def plus_enat_simps times_enat_simps enat_ord_code*(*1*) *eSuc_enat*) **done**
subgoal **for** *s* **apply**(*cases s ''x'' > 0*) **apply** *auto*
apply(*simp only*: *one_enat_def plus_enat_simps times_enat_simps enat_ord_code*(*1*) *eSuc_enat*) **done**
done
subgoal **for** *s* **apply**(*simp only*: *one_enat_def plus_enat_simps times_enat_simps enat_ord_code*(*1*)) **by** *presburger*
subgoal **for** *s* **apply**(*cases s ''x'' > 0*) **by** *auto*
by *simp*

end

6.6 Verification Condition Generator

theory *QuantK_VCG*
imports *QuantK_Hoare*
begin

6.6.1 Ceiling integer division on extended natural numbers

definition *mydiv* (*a*::*nat*) (*k*::*nat*) = (*if k dvd a then a div k else* (*a div k*) + *1*)

lemma *mydivcode*: $k>0 \Longrightarrow D\geq k \Longrightarrow$ *mydiv D k* = *Suc* (*mydiv* (*D*−*k*) *k*)

unfolding *mydiv_def* **apply** (*auto simp add*: *le_div_geq*)
using *dvd_minus_self* **by** *auto*

lemma *mydivcode1*: *mydiv 0 k* = *0*
unfolding *mydiv_def* **by** *auto*

lemma *mydivcode2*: $k>0 \Longrightarrow 0<D \Longrightarrow D<k \Longrightarrow$ *mydiv D k = Suc 0*
unfolding *mydiv_def* **by** *auto*

lemma *mydiv_mono*: $a \le b \Longrightarrow$ *mydiv a k* $\le$ *mydiv b k* **unfolding** *mydiv_def*
apply(*cases k dvd a*)
subgoal apply(*cases k dvd b*) **apply** *auto* **apply** (*auto simp add*: *div_le_mono*)
using *div_le_mono le_Suc_eq* **by** *blast*
subgoal apply(*cases k dvd b*) **apply** *auto* **apply** (*auto simp add*: *div_le_mono*)
by (*metis Suc_leI add.right_neutral div_le_mono div_mult_mod_eq dvd_imp_mod_0 le_add1 le_antisym less_le*)
done

lemma *mydiv_cancel*: $0 < k \Longrightarrow$ *mydiv* $(k * i)$ $k = i$
unfolding *mydiv_def* **by** *auto*

lemma assumes *k*: $k>0$ **and** *B*: $B \le k*A$
shows *mydiv_le_E*: *mydiv B k* $\le A$
proof −
from *mydiv_mono*[*OF B*] **and** *k mydiv_cancel* **show** *?thesis*
by *metis*
qed
lemma *mydiv_mult_leq*: $0 < k \Longrightarrow l \le k \Longrightarrow$ *mydiv* $(l*A)$ $k \le A$
by(*simp add*: *mydiv_le_E*)

lemma *mydiv_cancel3*: $0 < k \Longrightarrow i \le k *$ *mydiv i k*
by (*auto simp add*: *mydiv_def dividend_less_times_div le_eq_less_or_eq*)

definition *ediv a k* = (*if* $a=\infty$ *then* ∞ *else enat* (*mydiv* (*THE i. a=enat i*) *k*))

lemma *ediv_enat*[*simp*]: *ediv* (*enat a*) *k* = *enat* (*mydiv a k*)
unfolding *ediv_def* **by** *auto*
lemma *ediv_mydiv*[*simp*]: *ediv* (*enat a*) $k \le$ *enat f* $\longleftrightarrow$ *mydiv a k* $\le f$
unfolding *ediv_def* **by** *auto*

lemma *ediv_mono*: $a \le b \Longrightarrow$ *ediv a k* $\le$ *ediv b k*
unfolding *ediv_def* **by** (*auto simp add*: *mydiv_mono*)

lemma *ediv_cancel2*: $k>0 \Longrightarrow$ *ediv* (*enat* $k * x$) $k = x$
unfolding *ediv_def* **apply**(*cases* $x=\infty$) **using** *mydiv_cancel* **by** *auto*

lemma *ediv_cancel3*: $k>0 \Longrightarrow A \le$ *enat* $k *$ *ediv A k*

unfolding *ediv_def* **apply**(*cases A*=∞) **using** *mydiv_cancel3* **by** *auto*

6.6.2 Definition of VCG

datatype *acom* =
  Askip (‹*SKIP*›) |
  Aassign vname aexp (‹(_ ::= _)› [*1000*, *61*] *61*) |
  Aseq acom acom (‹_;;/ _› [*60*, *61*] *60*) |
  Aif bexp acom acom (‹(*IF* _/ *THEN* _/ *ELSE* _)› [*0*, *0*, *61*] *61*) |
  Awhile qassn bexp acom (‹({_}/ *WHILE* _/ *DO* _)› [*0*, *0*, *61*] *61*)
| *Abst nat acom* (‹({_}/ *Ab* _)› [*0*, *61*] *61*)

notation *com.SKIP* (‹*SKIP*›)

fun *strip* :: *acom* ⇒ *com* **where**
strip SKIP = *SKIP* |
strip (*x* ::= *a*) = (*x* ::= *a*) |
strip (C_1;; C_2) = (*strip* C_1;; *strip* C_2) |
strip (*IF b THEN* C_1 *ELSE* C_2) = (*IF b THEN strip* C_1 *ELSE strip* C_2) |
strip ({_} *WHILE b DO C*) = (*WHILE b DO strip C*) |
strip ({_} *Ab C*) = *strip C*

fun *pre* :: *acom* ⇒ *qassn* ⇒ *qassn* **where**
pre SKIP Q = (λ*s*. *eSuc* (*Q s*)) |
pre (*x* ::= *a*) *Q* = (λ*s*. *eSuc* (*Q* (*s*[*a*/*x*]))) |
pre (C_1;; C_2) *Q* = *pre* C_1 (*pre* C_2 *Q*) |
pre (*IF b THEN* C_1 *ELSE* C_2) *Q* =
  (λ*s*. *eSuc* (*if bval b s then pre* C_1 *Q s else pre* C_2 *Q s*)) |
pre ({*P*} *WHILE b DO C*) *Q* = (%*s*. *P s* + *1*) |
pre ({*k*} *Ab C*) *Q* = (λ*s*. *ediv* (*pre C* (λ*s*. *k*∗*Q s*) *s*) *k*)

In contrast to *pre*, *vc* produces a formula that is independent of the state:

fun *vc* :: *acom* ⇒ *qassn* ⇒ *bool* **where**
vc SKIP Q = *True* |
vc (*x* ::= *a*) *Q* = *True* |
vc (C_1 ;; C_2) *Q* = ((*vc* C_1 (*pre* C_2 *Q*)) ∧ (*vc* C_2 *Q*)) |
vc (*IF b THEN* C_1 *ELSE* C_2) *Q* = (*vc* C_1 *Q* ∧ *vc* C_2 *Q*) |
vc ({*I*} *WHILE b DO C*) *Q* = ((∀*s*. (*pre C* (λ*s*. *I s* + *1*) *s* ≤ *I s* + ↑(*bval b s*)) ∧ (*Q s* ≤ *I s* + ↑ (¬ *bval b s*))) ∧ *vc C* (%*s*. *I s* + *1*)) |
vc ({*k*} *Ab C*) *Q* = (*vc C* (λ*s*. *enat k*∗ *Q s*) ∧ *k*>*0* ~~∧ (∀ s. pre C (λs. enat k ∗ Q s) s ≤ enat k ∗ ediv (pre C (λs. enat k ∗ Q s) s) k)~~)

6.6.3 Soundness of VCG

lemma *vc_sound*: *vc C Q* ⟹ ⊢$_{2'}$ {*pre C Q*} *strip C* { *Q* }

```
proof (induct C arbitrary: Q)
  case (Aif b C1 C2)
  then have Aif1: ⊢2' {pre C1 Q} strip C1 {Q} and Aif2: ⊢2' {pre C2 Q}
strip C2 {Q} by auto
    show ?case apply auto apply(rule hoareQ.conseq[where k=1])
       apply(rule hoareQ.If[where P=%s. if bval b s then pre C1 Q s else
pre C2 Q s and Q=Q])
    subgoal
      apply(rule hoareQ.conseq[where k=1])
         apply (fact Aif1)
      subgoal for s apply(cases bval b s) by auto
      apply auto done
    subgoal
      apply(rule hoareQ.conseq[where k=1])
        apply (fact Aif2)
      subgoal for s apply(cases bval b s) by auto
      apply auto done
     apply auto
    done
next
  case (Awhile I b C)
  then have i: (⋀Q. vc C Q ⟹ ⊢2' {pre C Q} strip C {Q})
   and ii': ∀s. pre C (λs. I s + 1) s ≤ I s + ↑ (bval b s)
   and ii'': ⋀s. Q s ≤  I s + ↑ (¬ bval b s)
     and iii: vc C (λs. I s + 1)
    by auto

  from i iii have  A: ⊢2' {pre C (λs. I s + 1)} strip C {(λs. I s + 1)} by
auto

  show ?case
    apply simp
    apply(rule conseq[where k=1])
       apply(rule While[where I=I])
       apply(rule weakenpre)
        apply(rule A)
      apply(rule ii') apply simp
    using ii'' apply auto done
next
  case (Abst k C)
  then have vc: vc C (λs. k* Q s) and k: k>0 by auto
   from Abst(1) vc have C: ⊢2' {pre C (%s. k*Q s)} strip C {(%s. k*Q
s)} by auto
  show ?case apply(simp)
```

apply(*rule conseq*)
apply(*rule C*) **using** *k* **apply** *auto*
using *ediv_cancel3* **by** *auto*
qed (*auto intro*: *hoareQ.Skip hoareQ.Assign hoareQ.Seq*)

lemma *vc_sound'*: $\llbracket vc\ C\ Q\ ;\ (\forall s.\ pre\ C\ Q\ s \leq P\ s)\ \rrbracket \Longrightarrow\ \vdash_{2'} \{P\}\ strip\ C\ \{\ Q\ \}$
apply(*rule hoareQ.conseq*[**where** *k=1*])
apply(*rule vc_sound*) **by** *auto*

lemma *vc_sound''*: $\llbracket vc\ C\ Q'\ ;\ (\forall s.\ \ pre\ C\ Q'\ s \leq k * P\ s)\ \ ;\ (\bigwedge s.\ enat\ k * Q\ s \leq Q'\ s);\ k{>}0\ \rrbracket \Longrightarrow\ \vdash_{2'} \{P\}\ strip\ C\ \{\ Q\ \}$
apply(*rule hoareQ.conseq*)
apply(*rule vc_sound*) **by** *auto*

6.6.4 Completeness

lemma *pre_mono*: **assumes** $\bigwedge s.\ P'\ s \leq P\ s$
shows $\bigwedge s.\ pre\ C\ P'\ s \leq pre\ C\ P\ s$
using *assms* **by** (*induct C arbitrary*: *P P'*, *auto simp*: *ediv_mono mult_left_mono*)

lemma *vc_mono*: **assumes** $\bigwedge s.\ P'\ s \leq P\ s$
shows $vc\ C\ P \Longrightarrow vc\ C\ P'$
using *assms*
proof (*induct C arbitrary*: *P P'*)
case (*Awhile I b C Q*)
thus *?case*
apply (*auto simp*: *pre_mono*)
using *order.trans* **by** *blast*
next
case (*Abst x1 C*)
then show *?case* **by** (*auto simp*: *mult_left_mono*)
qed (*auto simp*: *pre_mono*)

lemma $\vdash_{2'}\ \{\ P\ \}\ c\ \{\ Q\ \} \Longrightarrow \exists\, C.\ strip\ C = c \wedge vc\ C\ Q \wedge (\forall s.\ pre\ C\ Q\ s \leq P\ s)$
(**is** _ $\Longrightarrow\ \ \exists\, C.\ ?G\ P\ c\ Q\ C$)
proof (*induction rule*: *hoareQ.induct*)
case (*conseq P c Q k P' Q'*)
then obtain *C* **where** *strip*: *strip C = c* **and** *vc*: *vc C Q* **and** *pre*: $\bigwedge s.$

pre C Q s $\leq$ *P s*
  by *blast*

 { fix *s*
  have *pre C* (λ*s. enat k* $*$ *Q′ s*) *s* $\leq$ *pre C Q s* **using** *pre_mono conseq*(*3*) **by** *simp*
  also
  from *pre conseq*(*2*) **have** ... $\leq$ *enat k* $*$ *P′ s* **using** *order.trans* **by** *blast*
  finally have *pre C* (λ*s. enat k* $*$ *Q′ s*) *s* $\leq$ *enat k* $*$ *P′ s* **by** *auto*
  then have *ediv* (*pre C* (λ*s. enat k* $*$ *Q′ s*) *s*) *k* $\leq$ *ediv* (*enat k* $*$ *P′ s*) *k* **using** *ediv_mono* **by** *auto*
  moreover note *ediv_cancel2*[*OF conseq*(*4*)]
  ultimately have *ediv* (*pre C* (λ*s. enat k* $*$ *Q′ s*) *s*) *k* $\leq$ *P′ s*
   by *simp*
 } note *compensate=this*

 show *?case*
  apply(*rule exI*[**where** *x*={*k*} *Ab C*])
  apply(*safe*)
  subgoal using *strip* **by** *simp*
  subgoal apply *simp* **apply** *safe*
   subgoal using *vc vc_mono conseq*(*3*) **by** *force*
   subgoal by *fact*
   done
  subgoal apply *simp* **using** *compensate* **by** *auto*
   done
next
 case (*Skip P*)
 show *?case* (**is** $\exists$ *C. ?C C*)
 proof show *?C Askip* **by** *auto* **qed**
next
 case (*Assign P a x*)
 show *?case* (**is** $\exists$ *C. ?C C*)
 proof show *?C*(*Aassign x a*) **by** *auto* **qed**
next
 case (*If P b* c_1 *Q* c_2)
 from *If*(*3*) **obtain** *C1* **where** *strip1*: *strip C1* = c_1 **and** *vc1*: *vc C1 Q*
   and *pre1*: ($\bigwedge$*s. pre C1 Q s* $\leq$ (*P s* + $\uparrow$(*bval b s*)))
   by *blast*
 from *If*(*4*) **obtain** *C2* **where** *strip2*: *strip C2* = c_2 **and** *vc2*: *vc C2 Q*
   and *pre2*: ($\bigwedge$*s. pre C2* (λ*s. Q s*) *s* $\leq$ (*P s* + $\uparrow$($\neg$ *bval b s*)))
  by *blast*

```
  show ?case
    apply(rule exI[where x=IF b THEN C1 ELSE C2], safe)
    subgoal using strip1 strip2 by auto
    subgoal  using vc1 vc2 by auto
    subgoal for s using pre1[of s] pre2[of s] by auto
    done
next
  case (Seq P₁ c₁ P₂ c₂ P₃)
  from Seq(3) obtain C1 where strip1: strip C1 = c₁ and vc1: vc C1 P₂
      and pre1: (∀ s. pre C1 P₂ s ≤   P₁ s)    by blast
  from Seq(4) obtain C2 where strip2: strip C2 = c₂ and vc2: vc C2 P₃
      and pre2: ⋀s. pre C2 P₃ s ≤   P₂ s   by blast

  {
    fix s
    have pre C1 (pre C2 P₃) s ≤ P₁ s
      apply(rule order.trans[where b=pre C1 P₂ s])
       apply(rule pre_mono) using pre2 apply simp using pre1 by simp
  } note pre = this
  show ?case
    apply(rule exI[where x=C1 ;; C2], safe)
    subgoal using strip1 strip2 by simp
    subgoal apply simp apply safe  using vc1 vc2 vc_mono pre2 by auto

    subgoal    apply simp using  pre by auto
      done
next
  case (While I b c)
  from While(2) obtain C where strip: strip C = c and vc: vc C (λa. I
a + 1)
    and pre: ⋀s. pre C (λa. I a + 1) s ≤ I s + ↑ (bval b s) by blast
  show ?case
    apply(rule exI[where x={I} WHILE b DO C], safe)
    subgoal using strip by simp
    subgoal apply simp using pre vc by auto
    subgoal by simp
  done
qed

lemma ⊢Z { P } c { Q } ⟹ ∃ C. strip C = c ∧ vc C Q ∧ (∀ s. pre C Q
s ≤ P s)
  (is _ ⟹   ∃ C. ?G P c Q C)
proof (induction rule: hoareQ'.induct)
  case (ZSkip P)
```

show *?case* (**is** $\exists\, C.\ ?C\ C$)
proof **show** *?C Askip* **by** *auto*
qed
next
case (*ZAssign P a x*)
show *?case* (**is** $\exists\, C.\ ?C\ C$)
proof **show** *?C*(*Aassign x a*) **by** *simp* **qed**
next
case (*ZIf P b* c_1 *Q* c_2)
from *ZIf*(*3*) **obtain** *C1* **where** *strip1*: *strip C1* = c_1 **and** *vc1*: *vc C1 Q* **and** *pre1*: ($\bigwedge s.$ *pre C1 Q s* $\leq$ *P s* + $\uparrow$ (*bval b s*)) **by** *blast*
from *ZIf*(*4*) **obtain** *C2* **where** *strip2*: *strip C2* = c_2 **and** *vc2*: *vc C2 Q* **and** *pre2*: ($\bigwedge s.$ *pre C2 Q s* $\leq$ *P s* + $\uparrow$ ($\neg$ *bval b s*)) **by** *blast*

show *?case* **apply**(*rule exI*[**where** *x*=*IF b THEN C1 ELSE C2*])
apply(*safe*)
subgoal **using** *strip1 strip2* **by** *auto*
subgoal **using** *vc1 vc2* **by** *auto*
subgoal **for** *s* **apply** *auto*
subgoal **using** *pre1*[*of s*] **by** *auto*
subgoal **using** *pre2*[*of s*] **by** *auto*
done
done
next
case (*ZSeq* P_1 c_1 P_2 c_2 P_3)
from *ZSeq*(*3*) **obtain** *C1* **where** *strip1*: *strip C1* = c_1 **and** *vc1*: *vc C1* P_2 **and** *pre1*: ($\forall\, s.$ *pre C1* P_2 *s* $\leq$ P_1 *s*) **by** *blast*
from *ZSeq*(*4*) **obtain** *C2* **where** *strip2*: *strip C2* = c_2 **and** *vc2*: *vc C2* P_3 **and** *pre2*: ($\forall\, s.$ *pre C2* P_3 *s* $\leq$ P_2 *s*) **by** *blast*
{
fix *s*
have *pre C1* (*pre C2* P_3) *s* $\leq$ P_1 *s*
apply(*rule order.trans*[**where** *b*=*pre C1* P_2 *s*])
apply(*rule pre_mono*) **using** *pre2* **apply** *simp* **using** *pre1* **by** *simp*
} **note** *pre* = *this*
show *?case* **apply**(*rule exI*[**where** *x*=*C1* ;; *C2*])
apply *safe*
subgoal **using** *strip1 strip2* **by** *simp*
subgoal **using** *vc1 vc2 vc_mono pre2* **by** *auto*
subgoal **using** *pre* **by** *auto*
done
next
case (*ZWhile I b c*)
from *ZWhile*(*2*) **obtain** *C* **where** *strip*: *strip C* = *c* **and** *vc*: *vc C* ($\lambda a.$

I a + 1)
    and *pre*: $\bigwedge$*s. pre C* (λ*a. I a + 1*) *s* $\leq$ *I s* + $\uparrow$ (*bval b s*) **by** *blast*
  show *?case* **apply**(*rule exI*[**where** *x*={*I*} *WHILE b DO C*])
    apply *safe*
    subgoal using *strip* **by** *simp*
    subgoal using *pre vc* **by** *auto*
    subgoal by *simp*
  done
next
  case (*Zconseq' P c Q P' Q'*)
  then obtain *C* **where** *strip C* = *c* **and** *vc*: *vc C Q* **and** *pre*: $\bigwedge$*s. pre C Q s* $\leq$ *P s* **by** *blast*

  from *pre_mono*[*OF Zconseq'*(*3*)] **have** *1*: $\bigwedge$*s. pre C Q' s* $\leq$ *pre C Q s* **by** *auto*

  show *?case*
    apply(*rule exI*[**where** *x*=*C*])
    apply *safe*
      apply *fact*
    subgoal using *vc Zconseq'*(*3*) *vc_mono* **by** *auto*
    subgoal using *pre Zconseq'*(*2*) *1* **using** *order.trans* **by** *metis*
    done
next
  case (*Zconst k P c Q*)
  then obtain *C* **where** *strip*: *strip C* = *c* **and** *vc*: *vc C* (λ*a. enat k* $*$ *Q a*)
    and *k*: *k*>*0* **and** *pre*: $\bigwedge$*s. pre C* (λ*a. enat k* $*$ *Q a*) *s* $\leq$ *enat k* $*$ *P s* **by** *blast*
  show *?case*
    apply(*rule exI*[**where** *x*={*k*} *Ab C*]) **apply** *safe*
    subgoal using *strip* **by** *auto*
    subgoal using *vc k* **by** *auto*
    subgoal apply *auto* **using** *ediv_mono*[*OF pre*] *ediv_cancel2*[*OF k*] **by** *metis*
    done
qed

end

6.7 Examples for quantitative Hoare logic

theory *QuantK_Examples*

imports *QuantK_VCG*
begin

fun *sum* :: *int* ⇒ *int* **where**
sum i = (*if i* ≤ *0 then 0 else sum* (*i* − *1*) + *i*)

abbreviation *wsum* ==
  WHILE Less (*N 0*) (*V ''x''*)
  DO (*''y''* ::= *Plus* (*V ''y''*) (*V ''x''*);;
     ''x'' ::= *Plus* (*V ''x''*) (*N* (− *1*)))

lemma *example*: $\vdash_{2'}$ {λ*s. enat* (*2* + *3*∗*n*) + *emb* (*s ''x''* = *int n*)} *''y''* ::=
N 0;; *wsum* {λ*s. 0* }
apply(*rule Seq*)
 prefer *2*
 apply(*rule conseq'*)
 apply(*rule While*[**where** *I*=λ*s. enat* (*3* ∗ *nat* (*s ''x''*))])
  apply(*rule Seq*)
   prefer *2*
     apply(*rule Assign*)
     apply(*rule Assign'*)
    apply(*simp*)
    apply(*safe*) **subgoal for** *s* **apply**(*cases 0* < *s ''x''*) **apply**(*simp*)
   apply (*smt Suc_eq_plus1 Suc_nat_eq_nat_zadd1 distrib_left_numeral eSuc_numeral enat_numeral eq_iff iadd_Suc_right nat_mult_1_right one_add_one plus_1_eSuc*(*1*) *plus_enat_simps*(*1*) *semiring_norm*(*5*))
   apply(*simp*) **done**
   apply *blast*
  apply *simp*
apply(*rule Assign'*)
apply *simp*
 apply(*safe*) **subgoal for** *s* **apply**(*cases s ''x''* = *int n*) **apply**(*simp*)
    apply (*simp add*: *eSuc_enat plus_1_eSuc*(*2*))
   apply *simp*
   done
  done

lemma *example_sound*: $\models_{2'}$ {λ*s. enat* (*2* + *3*∗*n*) + *emb* (*s ''x''* = *int n*)}
''y'' ::= *N 0*;; *wsum* {λ*s. 0* }
apply(*rule hoareQ_sound*) **apply** (*rule example*) **done**

schematic_goal $\vdash_{2'}$ $\{\lambda s.\ ?A\ s + emb\ (s\ ''x'' = int\ n)\}$ $''y'' ::= N\ 0;;$ *wsum* $\{\lambda s.\ 0\ \}$
apply(*rule Seq*)
 prefer *2*
 apply(*rule conseq'*)
      apply(*rule While*)
  apply(*rule Seq*)
   prefer *2*
     apply(*rule Assign*)
     apply(*rule Assign'*)
    apply(*simp*)
    apply(*safe*) **apply**(*case_tac* $0 < s\ ''x''$) **apply**(*simp*) **defer**
   apply(*simp*)
   apply *blast*
  apply *simp*
apply(*rule Assign'*)
apply *simp*
  apply(*safe*) **apply**(*case_tac* $s\ ''x'' = int\ n$) **apply**(*simp*)
    apply (*simp add*: *eSuc_enat plus_1_eSuc*(*2*)) **defer**
   apply *simp*
  prefer *2* **apply** *auto* **oops**

6.7.1 Example for VCG

lemma $\vdash_{2'}$ $\{\lambda s.\ 1\}$ *SKIP* ;; *SKIP* $\{\lambda s.\ 0\ \}$
proof −
  have $\vdash_{2'}$ $\{\lambda s.\ enat\ 1\}$ *strip* ($\{2\}$ *Ab* (*SKIP* ;; *SKIP*)) $\{\lambda s.\ 0\ \}$
   apply(*rule vc_sound'*)
    apply(*auto simp*: *eSuc_enat zero_enat_def*)
   by (*simp add*: *mydivcode mydivcode1 mydivcode2*)
  then show *?thesis* **by** (*simp add*: *one_enat_def*)
qed

lemma *hoareQ_Seq_assoc*: $\vdash_{2'}$ $\{P\}$ *A*;; *B*;; *C* $\{Q\}$ = ($\vdash_{2'}$ $\{P\}$ *A*;; (*B*;; *C*) $\{Q\}$)
 by(*auto simp*: *hoare2o_valid_def hoareQ_sound_complete Seq_t_assoc*)

lemma $\vdash_{2'}$ $\{\lambda s.\ 1\}$ *SKIP* ;; *SKIP* ;; *SKIP* $\{\lambda s.\ 0\ \}$
proof −
  have $\vdash_{2'}$ $\{\lambda s.\ enat\ 1\}$ *strip* ($\{2\}$ *Ab* (*SKIP* ;; $\{2\}$ *Ab* (*SKIP* ;; *SKIP*))) $\{\lambda s.\ 0\ \}$

```
    apply(rule vc_sound')
     apply(auto simp: eSuc_enat zero_enat_def)
    by (simp add: mydivcode mydivcode1 mydivcode2)
  then show ?thesis by (simp add: one_enat_def hoareQ_Seq_assoc)
qed
```

```
abbreviation Wsum ==
  {λs. enat (3 * nat (s ''x''))} WHILE Less (N 0) (V ''x'')
  DO (''y'' ::= Plus (V ''y'') (V ''x'');;
      ''x'' ::= Plus (V ''x'') (N (− 1)))
```

```
lemma ⊢2' {λs. enat (2 + 3*n) + emb (s ''x'' = int n)} ''y'' ::= N 0;;
wsum {λs. 0 }
proof −
  have ⊢2' {λs. enat (2 + 3*n) + emb (s ''x'' = int n)} strip (''y'' ::= N
0;; Wsum) {λs. 0 }
    apply(rule vc_sound')
    subgoal
      apply simp
      apply(safe) subgoal for s apply(cases 0 < s ''x'')
        apply(simp)
   apply ( smt Suc_eq_plus1 Suc_nat_eq_nat_zadd1 distrib_left_numeral
eSuc_numeral enat_numeral eq_iff iadd_Suc_right nat_mult_1_right one_add_one
plus_1_eSuc(1) plus_enat_simps(1) semiring_norm(5))
        apply(simp) done
      done
    subgoal
      apply simp
      apply(safe) subgoal for s apply(cases s ''x'' = int n) apply(simp)

        apply (simp add: eSuc_enat plus_1_eSuc(2))
        apply simp
        done
      done
    done
  then show ?thesis by simp
 qed
```

```
lemma assumes n0: n>0 shows ⊢2' {λs. enat (n ) + emb (s ''x'' = int
```

n)} ''*y*'' ::= *N 0*;; *wsum* {λ*s*. *0* }
proof −
 from *n0* **obtain** *n'* **where** *n'*: *n*=*Suc n'*
  using *not0_implies_Suc* **by** *blast*
 have $\vdash_{2'}$ {λ*s*. *enat* (*n*) + *emb* (*s* ''*x*'' = *int n*)} *strip* ({*5*} *Ab* (''*y*'' ::= *N 0*;; *Wsum*)) {λ*s*. *0* }
  apply(*rule vc_sound'*)
  subgoal
   apply *simp*
   apply(*safe*) **subgoal for** *s* **apply**(*cases 0* < *s* ''*x*'')
    apply(*simp*)
 apply (*smt Suc_eq_plus1 Suc_nat_eq_nat_zadd1 distrib_left_numeral eSuc_numeral enat_numeral eq_iff iadd_Suc_right nat_mult_1_right one_add_one plus_1_eSuc*(*1*) *plus_enat_simps*(*1*) *semiring_norm*(*5*))
    apply(*simp*) **done**
   done
  subgoal
   apply *simp*
   apply(*safe*) **subgoal for** *s* **apply**(*cases s* ''*x*'' = *int n*) **apply**(*simp*)

    subgoal apply (*simp add*: *eSuc_enat plus_1_eSuc*(*2*))
     apply(*simp add*: *n'*) **apply** (*simp add*: *mydiv_le_E*) **done**
    apply *simp*
    done
   done
  done
 then show *?thesis* **by** *simp*
qed

lemma $\vdash_{2'}$ {λ*s*. *enat* (*n*+*1*) + *emb* (*s* ''*x*'' = *int n*)} ''*y*'' ::= *N 0*;; *wsum* {λ*s*. *0* }
proof −
 have $\vdash_{2'}$ {λ*s*. *enat* (*n*+*1*) + *emb* (*s* ''*x*'' = *int n*)} *strip* ({*3*} *Ab* (''*y*'' ::= *N 0*;; *Wsum*)) {λ*s*. *0* }
  apply(*rule vc_sound'*)
  subgoal
   apply *simp*
   apply(*safe*) **subgoal for** *s* **apply**(*cases 0* < *s* ''*x*'')
    apply(*simp*)
 apply (*smt Suc_eq_plus1 Suc_nat_eq_nat_zadd1 distrib_left_numeral eSuc_numeral enat_numeral eq_iff iadd_Suc_right nat_mult_1_right one_add_one plus_1_eSuc*(*1*) *plus_enat_simps*(*1*) *semiring_norm*(*5*))
    apply(*simp*) **done**
   done

```
    subgoal
      apply simp
      apply(safe) subgoal for s apply(cases s ''x'' = int n) apply(simp)

        subgoal apply (simp add: eSuc_enat plus_1_eSuc(2))
            apply (simp add: mydiv_le_E) done
        apply simp
        done
      done
    done
  then show ?thesis by simp
qed
```

```
abbreviation Wsum1 z ==
  {λs. enat (z ∗ nat (s ''x''))} WHILE Less (N 0) (V ''x'')
  DO (''y'' ::= Plus (V ''y'') (V ''x'');;
      ''x'' ::= Plus (V ''x'') (N (− 1)))

abbreviation Wsum2 n vier ==
  {λs. enat (vier ∗ (nat (s ''x'') +  n + 1))  } WHILE Less (N 0) (V ''x'')
  DO (''y'' ::= Plus (V ''y'') (V ''x'');;
      ''x'' ::= Plus (V ''x'') (N (− 1)))
```

```
end
theory QuantK_Sqrt
imports QuantK_VCG HOL−Library.Discrete_Functions
begin
```

6.8 Example: discrete square root in the quantitative Hoare logic

As an example, consider the following program that computes the discrete square root:

```
definition c :: com where c=
        ''l''::= N 0 ;;
        ''m'' ::= N 0 ;;
        ''r''::= Plus (N 1) (V ''x'');;
        (WHILE (Less (Plus (N 1) (V ''l'')) (V ''r''))
```

```
DO (''m'' ::= (Div (Plus (V ''l'') (V ''r'')) (N 2)) ;;
  (IF Not (Less (Times (V ''m'') (V ''m'')) (V ''x''))
    THEN ''l'' ::= V ''m''
    ELSE ''r'' ::= V ''m'');;
  ''m'' ::= N 0))
```

In this theory we will show that its running time is in the order of magnitude of the logarithm of the variable "x"

a little lemma we need later for bounding the running time:

```
lemma absch: ⋀s k. 1 + s ''x'' = 2 ^ k ⟹ 5 * k ≤ 96 + 100 * floor_log
(nat (s ''x''))
proof -
  fix s :: state and k :: nat
  assume F: 1 + s ''x'' = 2 ^ k
  then have i: nat (1 + s ''x'') = 2 ^ k and nn: s ''x''≥ 0 apply (auto
simp: nat_power_eq)
    by (smt one_le_power)
  have F: 1 + nat (s ''x'') = 2 ^k unfolding i[symmetric] using nn by
auto
  show 5 * k ≤ 96 + 100 * floor_log (nat (s ''x''))
  proof (cases s ''x'' ≥ 1)
    case True
    have 5 * k = 5 * (floor_log (2^k))     by auto
    also have ... = 5 * floor_log (1 + nat (s ''x'')) by(simp only: F[symmetric])
    also have ... ≤ 5 * floor_log (nat (s ''x'' + s ''x'')) using True
      apply auto apply(rule monoD[OF floor_log_mono]) by auto
     also have ... = 5 * floor_log (2 * nat (s ''x'')) by (auto simp:
nat_mult_distrib)
    also have ... = 5 + 5 * (floor_log (nat (s ''x''))) using True by auto
    also have ... ≤ 96 + 100 * floor_log (nat (s ''x'')) by simp
    finally show ?thesis .
  next
    case False
    with nn have gt1: s ''x'' = 0 by auto
    from F[unfolded gt1] have 2 ^ k = (1::int) using floor_log_Suc_zero
by auto
    then have k=0
    by (metis One_nat_def add.right_neutral gt1 i n_not_Suc_n nat_numeral
nat_power_eq_Suc_0_iff numeral_2_eq_2 numeral_One)
    then show ?thesis by(simp add: gt1)
  qed
qed
```

For simplicity we assume, that during the process all segments between

”l” and ”r” have as length a power of two. This simplifies the analysis. To obtain this we choose the prepotential P accordingly.

Now lets show the correctness of our time complexity: the binary search is in O(log ”x”)

lemma
assumes
P: $P\ = (\lambda s. \uparrow (\ \ (\exists k.\ 1 + s\ ''x''\ = 2\ \hat{}\ k)) + (floor\_log\ (nat\ (\ s\ ''x'')) + 1))$ **and**
Q[*simp*]: $Q = (\lambda\_.\ 0)$
shows $\vdash_{2'} \{P\}\ c\ \{Q\}$
proof −
— first we create an annotated command
let $?lb = ''m'' ::=$
$(Div\ (Plus\ (V\ ''l'')\ (V\ ''r''))\ (N\ 2))\ ;;$
$(IF\ Not\ (Less\ (Times\ (V\ ''m'')\ (V\ ''m''))\ (V\ ''x''))$
$THEN\ ''l'' ::= V\ ''m''$
$ELSE\ ''r'' ::= V\ ''m'');;$
$(''m'' ::= N\ 0)::acom$
— with an invariant potential
define *I* **where** $I \equiv (\lambda s::state.\ ((\ emb\ (\ \ s\ ''l''{\geq}0\ \ \ \wedge (\ \exists k.\ s\ ''r'' - s\ ''l'' = 2\ \hat{}\ k)\ ) + 5 * floor\_log\ (nat\ (s\ ''r'') - nat\ (s\ ''l'')))::enat)\ )$
let $?C =\ \ ((''l''::= N\ 0)\ ::\ acom)\ ;;\ (''m'' ::= N\ 0)\ ;;\ ''r''::= Plus\ (N\ 1)\ (V\ ''x'');;\ (\{I\}\ WHILE\ (Less\ (Plus\ (N\ 1)\ (V\ ''l''))\ (V\ ''r''))\ DO\ ?lb)$

— we show that the annotated command corresponds to the command we are interested in
have *s*: *strip ?C = c* **unfolding** *c_def* **by** *auto*

— now we show that the annotated command is correct; here we use the VCG for the QuantK logic
have *v*: $\vdash_{2'} \{P\}\ strip\ ?C\ \{Q\}$
proof (*rule vc_sound''*, *safe*)

— A) first lets show the verification conditions:
show *vc ?C Q* **apply** *auto*
unfolding *I_def*
subgoal for *s*
apply(*cases* $(\exists k.\ s\ ''r''\ - s\ ''l'' = 2\ \hat{}\ k)$) **apply** *auto*
apply(*cases* $(1 + s\ ''l'' < s\ ''r'')$) **apply** *auto*
apply(*cases* $0 \leq s\ ''l''$) **apply** *auto*
proof (*goal_cases*)
case (*1 k*)
then have $k{>}0$ **using** *gr0I* **by** *force*

then obtain k' **where** k': $k=k'+1$ **by** (*metis Suc_eq_plus1 Suc_pred*)

from *1* k' **have** *R*: $s\ ''r'' - (s\ ''l'' + s\ ''r'')\ div\ 2 = 2\ \hat{}\ k'$ **by** *auto*
have *gN*: $s\ ''l''\leq s\ ''r''\ \ s\ ''l''\geq 0\ s\ ''r'' \geq 0$ **using** *1* **by** *auto*
have *n*: $nat\ (\ s\ ''r'' - (s\ ''l'' + s\ ''r'')\ div\ 2\ ) =\ nat\ (s\ ''r'') - nat\ ((s\ ''l'' + s\ ''r'')\ div\ 2)$
using *gN* **apply**(*simp add*: *nat_diff_distrib nat_div_distrib*) **done**

have R': $nat\ (s\ ''r'') -\ nat\ ((s\ ''l'' + s\ ''r'')\ div\ 2) = 2\ \hat{}\ k'$
apply(*simp only*: *n*[*symmetric*] *R nat_power_eq*) **by** *auto*
have S': $nat\ (s\ ''r'') - nat\ (s\ ''l'') = 2\ \hat{}\ k$
using *gN* **apply**(*simp only*: *nat_diff_distrib*[*symmetric*] *1*(*2*) *nat_power_eq*) **by** *auto*
have *N*: $0 \leq (s\ ''l'' + s\ ''r'')\ div\ 2$ **using** *gN* **by** *auto*

from *N* **show** *?case* **apply** (*simp*) **apply** (*simp only* : *R* R' S' k') **by** (*auto simp*: *eSuc_enat plus_1_eSuc*(*2*))
qed
subgoal for *s*
apply(*cases* $\exists k.\ s\ ''r''\ - s\ ''l'' = 2\ \hat{}\ k$) **apply** *auto*
apply (*cases* $(1 + s\ ''l'' < s\ ''r'')$) **apply** *auto*
apply(*cases* $0 \leq s\ ''l''$) **apply** *auto*
proof (*goal_cases*)
case (*1 k*)
from *1*(*2*,*3*) **have** $k>0$ **using** *gr0I* **by** *force*
then obtain k' **where** k': $k=k'+1$ **by** (*metis Suc_eq_plus1 Suc_pred*)

from *1* k' **have** *R*: $(s\ ''l'' + s\ ''r'')\ div\ 2\ - s\ ''l'' = 2\ \hat{}\ k'$ **by** *auto*
have *gN*: $s\ ''l''\leq s\ ''r''\ \ s\ ''l''\geq 0\ s\ ''r'' \geq 0$ **using** *1* **by** *auto*
have *n*: $nat\ ((s\ ''l'' + s\ ''r'')\ div\ 2\ - s\ ''l'') =\ nat\ (\ (s\ ''l'' + s\ ''r'')\ div\ 2) - nat\ (s\ ''l'')$
using *gN* **apply**(*simp add*: *nat_diff_distrib nat_div_distrib*) **done**

have R': $nat\ (\ (s\ ''l'' + s\ ''r'')\ div\ 2) - nat\ (s\ ''l'') = 2\ \hat{}\ k'$
apply(*simp only*: *n*[*symmetric*] *R nat_power_eq*) **by** *auto*
have S': $nat\ (s\ ''r'') - nat\ (s\ ''l'') = 2\ \hat{}\ k$
using *gN* **apply**(*simp only*: *nat_diff_distrib*[*symmetric*] *1*(*2*) *nat_power_eq*) **by** *auto*

show *?case* **apply** (*simp only* : *R* R' S' k') **by** (*auto simp*: *eSuc_enat plus_1_eSuc*(*2*))
qed done
next
— B) lets show that the precondition implies the weakest precondition,

and that the time bound of C can be bounded by log "x"

fix *s*

show $pre\ ?C\ Q\ s \le enat\ 100 * P\ s$ **unfolding** *I_def* **apply**(*simp only*: *P*) **apply** *auto* **apply**(*cases* ($\exists k.\ 1 + s\ ''x'' = 2\ \hat{}\ k$))

apply (*auto simp*: *eSuc_enat plus_1_eSuc*(*2*) *nat_power_eq*)

using *absch* **by** *force*

qed *auto*

from *s v* **show** *?thesis* **by** *simp*

qed

end

7 Partial States

7.1 Partial evaluation of expressions

theory *Partial_Evaluation*
imports *AExp Vars*
begin

type_synonym *partstate* = (*vname* ⇒ *val option*)

definition *emb* :: *partstate* ⇒ *state* ⇒ *state* **where**
emb ps s = (%*v*. (*case* (*ps v*) *of* (*Some r*) ⇒ *r* | *None* ⇒ *s v*))

definition *part* :: *state* ⇒ *partstate* **where**
part s = (%*v*. *Some* (*s v*))

lemma *emb_part*[*simp*]: *emb* (*part s*) *q* = *s* **unfolding** *emb_def part_def* **by** *auto*

lemma *part_emb*[*simp*]: *dom ps* = *UNIV* ⟹ *part* (*emb ps q*) = *ps* **unfolding** *emb_def part_def* **apply**(*rule ext*)
by (*simp add*: *domD option.case_eq_if*)

lemma *dom_part*[*simp*]: *dom* (*part s*) = *UNIV* **unfolding** *part_def* **by** *auto*

abbreviation *optplus* :: *int option* ⇒ *int option* ⇒ *int option* **where** *optplus a b* ≡ (*case a of None* ⇒ *None* | *Some a'* ⇒ (*case b of None* ⇒ *None* | *Some b'* ⇒ *Some* (*a'* + *b'*)))

abbreviation *opttimes* :: *int option* ⇒ *int option* ⇒ *int option* **where** *opttimes a b* ≡ (*case a of None* ⇒ *None* | *Some a′* ⇒ (*case b of None* ⇒ *None* | *Some b′* ⇒ *Some* (*a′* ∗ *b′*)))
abbreviation *optdiv* :: *int option* ⇒ *int option* ⇒ *int option* **where** *optdiv a b* ≡ (*case a of None* ⇒ *None* | *Some a′* ⇒ (*case b of None* ⇒ *None* | *Some b′* ⇒ *Some* (*a′ div b′*)))

fun *paval′* :: *aexp* ⇒ *partstate* ⇒ *val option* **where**
paval′ (*N n*) *s* = *Some n* |
paval′ (*V x*) *s* = *s x* |
paval′ (*Plus* a_1 a_2) *s* = *optplus* (*paval′* a_1 *s*) (*paval′* a_2 *s*) |
paval′ (*Times* a_1 a_2) *s* = *opttimes* (*paval′* a_1 *s*) (*paval′* a_2 *s*) |
paval′ (*Div* a_1 a_2) *s* = *optdiv* (*paval′* a_1 *s*) (*paval′* a_2 *s*)

lemma *paval′ a ps* = *Some v* ⟹ *vars a* ⊆ *dom ps*
proof(*induct a arbitrary*: *v*)
  case (*Plus a1 a2*)
  from *Plus*(*3*) **obtain** *v1* **where** *1*: *paval′ a1 ps* = *Some v1*
    by *fastforce*
  with *Plus*(*3*) **obtain** *v2* **where** *2*: *paval′ a2 ps* = *Some v2*
    by *fastforce*
  from *Plus*(*1*)[*OF 1*] *Plus*(*2*)[*OF 2*] **show** *?case* **by** *auto*
next
  case (*Times a1 a2*)
  from *Times*(*3*) **obtain** *v1* **where** *1*: *paval′ a1 ps* = *Some v1*
    by *fastforce*
  with *Times*(*3*) **obtain** *v2* **where** *2*: *paval′ a2 ps* = *Some v2*
    by *fastforce*
  from *Times*(*1*)[*OF 1*] *Times*(*2*)[*OF 2*] **show** *?case* **by** *auto*
next
  case (*Div a1 a2*)
  from *Div*(*3*) **obtain** *v1* **where** *1*: *paval′ a1 ps* = *Some v1*
    by *fastforce*
  with *Div*(*3*) **obtain** *v2* **where** *2*: *paval′ a2 ps* = *Some v2*
    by *fastforce*
  from *Div*(*1*)[*OF 1*] *Div*(*2*)[*OF 2*] **show** *?case* **by** *auto*
qed (*simp_all*, *blast*)

lemma *paval′_aval*: *paval′ a ps* = *Some v* ⟹ *aval a* (*emb ps s*) = *v*
proof(*induct a arbitrary*: *v*)
  case (*Plus a1 a2*)
  from *Plus*(*3*) **obtain** *v1* **where** *1*: *paval′ a1 ps* = *Some v1*
    by *fastforce*

with *Plus*(*3*) **obtain** *v2* **where** *2*: *paval′ a2 ps* = *Some v2*
  by *fastforce*
from *Plus*(*1*)[*OF 1*] *Plus*(*2*)[*OF 2*] *Plus*(*3*) *1 2* **show** *?case* **by** *auto*
next
  case (*Times a1 a2*)
  from *Times*(*3*) **obtain** *v1* **where** *1*: *paval′ a1 ps* = *Some v1*
    by *fastforce*
  with *Times*(*3*) **obtain** *v2* **where** *2*: *paval′ a2 ps* = *Some v2*
    by *fastforce*
  from *Times*(*1*)[*OF 1*] *Times*(*2*)[*OF 2*] *Times*(*3*) *1 2* **show** *?case* **by** *auto*
next
  case (*Div a1 a2*)
  from *Div*(*3*) **obtain** *v1* **where** *1*: *paval′ a1 ps* = *Some v1*
    by *fastforce*
  with *Div*(*3*) **obtain** *v2* **where** *2*: *paval′ a2 ps* = *Some v2*
    by *fastforce*
  from *Div*(*1*)[*OF 1*] *Div*(*2*)[*OF 2*] *Div*(*3*) *1 2* **show** *?case* **by** *auto*
qed (*simp_all add*: *emb_def*)

fun *paval* :: *aexp* ⇒ *partstate* ⇒ *val* **where**
paval (*N n*) *s* = *n* |
paval (*V x*) *s* = *the* (*s x*) |
paval (*Plus* a_1 a_2) *s* = *paval* a_1 *s* + *paval* a_2 *s* |
paval (*Times* a_1 a_2) *s* = *paval* a_1 *s* ∗ *paval* a_2 *s* |
paval (*Div* a_1 a_2) *s* = *paval* a_1 *s div paval* a_2 *s*

lemma *paval_aval*: *vars a* ⊆ *dom ps* ⟹ *paval a ps* = *aval a* (λ*v. case ps v of None* ⇒ *s v* | *Some r* ⇒ *r*)
  by (*induct a*, *auto*)

lemma *paval′_paval*: *vars a* ⊆ *dom ps* ⟹ *paval′ a ps* = *Some* (*paval a ps*)
  by (*induct a*, *auto*)

lemma *paval_paval′*: *paval′ a ps* = *Some v* ⟹ *paval a ps* = *v*
proof(*induct a arbitrary*: *v*)
  case (*Plus a1 a2*)
  from *Plus*(*3*) **obtain** *v1* **where** *1*: *paval′ a1 ps* = *Some v1*
    by *fastforce*
  with *Plus*(*3*) **obtain** *v2* **where** *2*: *paval′ a2 ps* = *Some v2*
    by *fastforce*
  from *Plus*(*1*)[*OF 1*] *Plus*(*2*)[*OF 2*] *Plus*(*3*) *1 2* **show** *?case* **by** *auto*

next
  case (*Times a1 a2*)
  from *Times*(*3*) **obtain** *v1* **where** *1*: *paval′ a1 ps* = *Some v1*
    by *fastforce*
  with *Times*(*3*) **obtain** *v2* **where** *2*: *paval′ a2 ps* = *Some v2*
    by *fastforce*
  from *Times*(*1*)[*OF 1*] *Times*(*2*)[*OF 2*] *Times*(*3*) *1 2* **show** *?case* **by** *auto*
next
  case (*Div a1 a2*)
  from *Div*(*3*) **obtain** *v1* **where** *1*: *paval′ a1 ps* = *Some v1*
    by *fastforce*
  with *Div*(*3*) **obtain** *v2* **where** *2*: *paval′ a2 ps* = *Some v2*
    by *fastforce*
  from *Div*(*1*)[*OF 1*] *Div*(*2*)[*OF 2*] *Div*(*3*) *1 2* **show** *?case* **by** *auto*
qed *simp_all*

fun *pbval* :: *bexp* ⇒ *partstate* ⇒ *bool* **where**
pbval (*Bc v*) *s* = *v* |
pbval (*Not b*) *s* = (¬ *pbval b s*) |
pbval (*And* b_1 b_2) *s* = (*pbval* b_1 *s* ∧ *pbval* b_2 *s*) |
pbval (*Less* a_1 a_2) *s* = (*paval* a_1 *s* < *paval* a_2 *s*)

abbreviation *optnot* **where** *optnot a* ≡ (*case a of None* ⇒ *None* | *Some a′* ⇒ *Some* (~*a′*))

abbreviation *optand* **where** *optand a b* ≡ (*case a of None* ⇒ *None* | *Some a′* ⇒ (*case b of None* ⇒ *None* | *Some b′* ⇒ *Some* (*a′* ∧ *b′*)))
abbreviation *optless* **where** *optless a b* ≡ (*case a of None* ⇒ *None* | *Some a′* ⇒ (*case b of None* ⇒ *None* | *Some b′* ⇒ *Some* (*a′* < *b′*)))

fun *pbval′* :: *bexp* ⇒ *partstate* ⇒ *bool option* **where**
pbval′ (*Bc v*) *s* = *Some v* |
pbval′ (*Not b*) *s* = (*optnot* (*pbval′ b s*)) |
pbval′ (*And* b_1 b_2) *s* = (*optand* (*pbval′* b_1 *s*) (*pbval′* b_2 *s*)) |
pbval′ (*Less* a_1 a_2) *s* = (*optless* (*paval′* a_1 *s*) (*paval′* a_2 *s*))

lemma *pbval′_pbval*: *vars a* ⊆ *dom ps* ⟹ *pbval′ a ps* = *Some* (*pbval a ps*)
  apply(*induct a*) **apply** (*auto simp*: *paval′_paval*) **done**

lemma *paval_aval_vars*: *vars a* ⊆ *dom ps* ⟹ *paval a ps* = *aval a* (*emb ps s*)
  apply(*induct a*) **by**(*auto simp*: *emb_def*)

lemma *pbval_bval_vars*: *vars b* ⊆ *dom ps* ⟹ *pbval b ps* = *bval b* (*emb ps s*)
  apply(*induct b*) **apply** (*simp_all*)
  using *paval_aval_vars*[**where** *s*=*s*] **by** *auto*

lemma *paval'dom*: *paval' a ps* = *Some v* ⟹ *vars a* ⊆ *dom ps*
proof (*induct a arbitrary*: *v*)
  case (*Plus a1 a2*)
  then show *?case* **apply** *auto*
    apply *fastforce*
     by (*metis* (*no_types*, *lifting*) *domD option.case_eq_if option.collapse subset_iff*)
next
  case (*Times a1 a2*)
  then show *?case* **apply** *auto*
    apply *fastforce*
     by (*metis* (*no_types*, *lifting*) *domD option.case_eq_if option.collapse subset_iff*)
next
  case (*Div a1 a2*)
  then show *?case* **apply** *auto*
    apply *fastforce*
     by (*metis* (*no_types*, *lifting*) *domD option.case_eq_if option.collapse subset_iff*)
qed *auto*

end
theory *Product_Separation_Algebra*
imports *Separation_Algebra.Separation_Algebra*
begin

instantiation *prod* :: (*sep_algebra*, *sep_algebra*) *sep_algebra*
begin

definition
  zero_prod_def: *0* ≡ (*0*, *0*)

definition
  plus_prod_def: *m1* + *m2* ≡ (*fst m1* + *fst m2* , *snd m1* + *snd m2*)

definition
sep_disj_prod_def: *sep_disj m1 m2* ≡ *sep_disj* (*fst m1*) (*fst m2*) ∧ *sep_disj* (*snd m1*) (*snd m2*)

instance
apply *standard* **unfolding** *sep_disj_prod_def zero_prod_def plus_prod_def*

subgoal by *auto*
subgoal by (*auto simp*: *sep_disj_commuteI*)
subgoal by (*auto*)
subgoal using *sep_add_commute* **by** *metis*
subgoal by (*auto simp*: *sep_add_assoc*)
subgoal apply *auto* **using** *sep_disj_addD1* **by** *metis+*
subgoal apply *auto* **using** *sep_disj_addI1* **apply** *auto* **done**
done

end

lemma *sep_disj_prod_commute*[*simp*]: (*ps*, *0*) ## (*0*, *n*)   (*0*, *n*) ## (*ps*, *0*) **unfolding** *sep_disj_prod_def* **by** *auto*

lemma *sep_disj_prod_conv*[*simp*]: (*a*, *x*) ## (*b*, *y*) = (*a*##*b* ∧ *x*##*y*) **unfolding** *sep_disj_prod_def* **by** *auto*

lemma *sep_plus_prod_conv*[*simp*]: (*ps*, *n*) + (*ps*′, *n*′) = (*ps* + *ps*′, *n* + *n*′) **unfolding** *plus_prod_def* **by** *auto*

lemma
fixes *h* :: (′*a*::*sep_algebra*) ∗ (′*b*::*sep_algebra*)
shows ((%(*a*,*b*). *P a* ∧ *b* = *0*) ∗∗ (%(*a*,*b*). *a* = *0* ∧ *Q b*)) =
(%(*a*,*b*). *P a* ∧ *Q b*) **unfolding** *sep_conj_def sep_disj_prod_def plus_prod_def*
apply *auto* **apply**(*rule ext*) **apply** *auto* **by** *force*

instantiation *nat* :: *sep_algebra*
begin

definition
sep_disj_nat_def[*simp*]: *sep_disj* (*m1*::*nat*) *m2* ≡ *True*

instance
apply *standard* **by**(*auto*)
end

lemma
  fixes *h* :: *nat*
  shows $(P ** Q ** H)\ h = (Q ** H ** P)\ h$
  by (*simp add*: *sep_conj_ac*)

lemma
  fixes *h* :: (*'a*::*sep_algebra*) ∗ (*'b*::*sep_algebra*)
  shows $(P ** Q ** H)\ h = (Q ** H ** P)\ h$
  by (*simp add*: *sep_conj_ac*)

lemma
  fixes *h* :: *nat* ∗ *nat*
  shows $(P ** Q ** H)\ h = (Q ** H ** P)\ h$
  by (*simp add*: *sep_conj_ac*)

end
theory *Sep_Algebra_Add*
 imports *Separation_Algebra.Separation_Algebra Separation_Algebra.Sep_Heap_Instance*
   Product_Separation_Algebra
begin

definition *puree* :: *bool* ⇒ *'h*::*sep_algebra* ⇒ *bool* (‹↑›) **where** *puree P* ≡
λ*h*. *h*=*0* ∧ *P*

lemma *puree_alt*: ↑Φ = (⟨Φ⟩ *and* □)
  by (*auto simp*: *puree_def sep_empty_def*)

lemma *pure_alt*: ⟨Φ⟩ = (↑Φ ∗∗ *sep_true*)
  apply (*clarsimp simp*: *puree_def*)
proof −
  { fix *aa* :: *'a*
    obtain *aaa* :: (*'a* ⇒ *bool*) ⇒ (*'a* ⇒ *bool*) ⇒ *'a* **where**
      ff1: ⋀*p pa a pb pc aa*. (¬ (*p* ∧∗ *pa*) *a* ∨ *p* (*aaa p pb*) ∨ (*pb* ∧∗
pa) *a*) ∧ (¬ *pb* (*aaa p pb*) ∨ ¬ (*p* ∧∗ *pc*) *aa* ∨ (*pb* ∧∗ *pc*) *aa*)
      by (*metis* (*no_types*) *sep_globalise*)
    then have ∃ *p*. ((λ*a*. *a* = *0*) ∧∗ *p*) *aa*
      by (*metis* (*full_types*) *sep_conj_commuteI sep_conj_sep_emptyE*
sep_empty_def)
    then have ¬ Φ ∨ Φ ∧ ((λ*a*. *a* = *0*) ∧∗ (λ*a*. *True*)) *aa*
      using *ff1* **by** (*metis* (*no_types*) *sep_conj_commuteI*) **}**

then show $(\lambda a.\ \Phi) = (\lambda a.\ \Phi \wedge ((\lambda a.\ (a{::}'a) = 0) \wedge{*}\ (\lambda a.\ True))\ a)$
by *blast*
qed

abbreviation *NO_PURE* :: *bool* ⇒ (*'h::sep_algebra* ⇒ *bool*) ⇒ *bool*
where *NO_PURE X Q* ≡ (*NO_MATCH* (⟨*X*⟩::*'h*⇒*bool*) *Q* ∧ *NO_MATCH* ((↑*X*)::*'h*⇒*bool*) *Q*)

named_theorems *sep_simplify* ‹*Assertion simplifications*›

lemma *sep_reorder*[*sep_simplify*]:
$((a \wedge{*}\ b) \wedge{*}\ c) = (a \wedge{*}\ b \wedge{*}\ c)$
$(NO\_PURE\ X\ a) \Longrightarrow (a ** b) = (b ** a)$
$(NO\_PURE\ X\ b) \Longrightarrow (b \wedge{*}\ a \wedge{*}\ c) = (a \wedge{*}\ b \wedge{*}\ c)$
$(Q ** \langle P\rangle) = (\langle P\rangle ** Q)$
$(Q ** \uparrow P) = (\uparrow P ** Q)$
$NO\_PURE\ X\ Q \Longrightarrow (Q ** \langle P\rangle ** F) = (\langle P\rangle ** Q ** F)$
$NO\_PURE\ X\ Q \Longrightarrow (Q ** \uparrow P ** F) = (\uparrow P ** Q ** F)$
by (*simp_all add*: *sep.add_ac*)

lemma *sep_combine1*[*simp*]:
$(\uparrow P ** \uparrow Q) = \uparrow(P \wedge Q)$
$(\langle P\rangle ** \langle Q\rangle) = \langle P \wedge Q\rangle$
$(\uparrow P ** \langle Q\rangle) = \langle P \wedge Q\rangle$
$(\langle P\rangle ** \uparrow Q) = \langle P \wedge Q\rangle$
apply (*auto simp add*: *sep_conj_def puree_def intro*!: *ext*)
apply (*rule_tac x=0* **in** *exI*)
apply *simp*
done

lemma *sep_combine2*[*simp*]:
$(\uparrow P ** \uparrow Q ** F) = (\uparrow(P \wedge Q) ** F)$
$(\langle P\rangle ** \langle Q\rangle ** F) = (\langle P \wedge Q\rangle ** F)$
$(\uparrow P ** \langle Q\rangle ** F) = (\langle P \wedge Q\rangle ** F)$
$(\langle P\rangle ** \uparrow Q ** F) = (\langle P \wedge Q\rangle ** F)$
apply (*subst sep.add_assoc*[*symmetric*]; *simp*)+
done

lemma *sep_extract_pure*[*simp*]:
$NO\_MATCH\ True\ P \Longrightarrow (\langle P\rangle ** Q)\ h = (P \wedge (sep\_true ** Q)\ h)$
$(\uparrow P ** Q)\ h = (P \wedge Q\ h)$
$\uparrow True = \square$
$\uparrow False = sep\_false$
using *sep_conj_sep_true_right* **apply** *fastforce*

by (*auto simp*: *puree_def sep_empty_def*[*symmetric*])

lemma *sep_pure_front2*[*simp*]:
$(\uparrow P ** A ** \uparrow Q ** F) = (\uparrow(P \wedge Q) ** F ** A)$
apply (*simp add*: *sep_reorder*)
done

lemma *ex_h_simps*[*simp*]:
$Ex\ (\uparrow\Phi) \longleftrightarrow \Phi$
$Ex\ (\uparrow\Phi ** P) \longleftrightarrow (\Phi \wedge Ex\ P)$
apply (*cases* Φ; *auto*)
apply *auto*
done

lemma
fixes $h :: ('a \Rightarrow 'b\ option) * nat$
shows $(P ** Q ** H)\ h = (Q ** H ** P)\ h$
by (*simp add*: *sep_conj_ac*)

lemma *map_le_substate_conv*: *map_le* = *sep_substate*
unfolding *map_le_def sep_substate_def sep_disj_fun_def plus_fun_def domain_def dom_def none_def* **apply** (*auto intro*!: *ext*)
subgoal for *m1 m2* **apply**(*rule exI*[**where** *x*=%*x*. *if* (∃ *y*. *m1 x* = *Some y*) *then None else m2 x*])
by *auto*
by *blast*

end

7.2 Big step Semantics on partial states

theory *Big_StepT_Partial*
imports *Partial_Evaluation Big_StepT SepLogAdd/Sep_Algebra_Add HOL−Eisbach.Eisbach*
begin

type_synonym *lvname* = *string*
type_synonym *pstate_t* = *partstate* ∗ *nat*
type_synonym *assnp* = *partstate* ⇒ *bool*
type_synonym *assn2* = *pstate_t* ⇒ *bool*

7.2.1 helper functions

restrict **definition** *restrict* **where** *restrict S s* = (%*x*. *if x*:*S then Some* (*s x*) *else None*)

lemma *restrictI*: ∀ *x*∈*S*. *s1 x* = *s2 x* ⟹ *restrict S s1* = *restrict S s2*
unfolding *restrict_def* **by** *fastforce*

lemma *restrictE*: *restrict S s1* = *restrict S s2* ⟹ *s1* = *s2 on S*
unfolding *restrict_def* **by** (*meson option.inject*)

lemma *dom_restrict*[*simp*]: *dom* (*restrict S s*) = *S*
unfolding *restrict_def*
using *domIff* **by** *fastforce*

lemma *restrict_less_part*: *restrict S t* ⪯ *part t*
unfolding *restrict_def map_le_substate_conv*[*symmetric*] *map_le_def part_def* **apply** *auto*
by (*metis option.simps*(*3*))

Heap helper functions **fun** *lmaps_to_expr* :: *aexp* ⇒ *val* ⇒ *assn2* **where**
lmaps_to_expr a v = (%(*s*,*c*). *dom s* = *vars a* ∧ *paval a s* = *v* ∧ *c* = *0*)

fun *lmaps_to_expr_x* :: *vname* ⇒ *aexp* ⇒ *val* ⇒ *assn2* **where**
lmaps_to_expr_x x a v = (%(*s*,*c*). *dom s* = *vars a* ∪ {*x*} ∧ *paval a s* = *v* ∧ *c* = *0*)

lemma *subState*: *x* ⪯ *y* ⟹ *v* ∈ *dom x* ⟹ *x v* = *y v* **unfolding** *map_le_substate_conv*[*symmetric*] *map_le_def*
by *blast*

lemma **fixes** *ps*:: *partstate*
and *s*::*state*
assumes *vars a* ⊆ *dom ps ps* ⪯ *part s*
shows *emb_update*: *emb* [*x* ↦ *paval a ps*] *s* = (*emb ps s*) (*x* := *aval a* (*emb ps s*))
using *assms*
unfolding *emb_def* **apply** *auto* **apply** (*rule ext*)
apply(*case_tac v*=*x*)
apply(*simp add*: *paval_aval*)
apply(*simp*) **unfolding** *part_def* **apply**(*case_tac v* ∈ *dom ps*)
using *subState* **apply** *fastforce*
by (*simp add*: *domIff*)

lemma *paval_aval2*: *vars a* $\subseteq$ *dom ps* $\Longrightarrow$ *ps* $\preceq$ *part s* $\Longrightarrow$ *paval a ps* = *aval a s*
apply(*induct a*) **using** *subState* **unfolding** *part_def* **apply** *auto*
by *fastforce*

lemma **fixes** *ps*:: *partstate*
and *s*::*state*
assumes *vars a* $\subseteq$ *dom ps ps* $\preceq$ *part s*
shows *emb_update2*: *emb* (*ps*(*x* $\mapsto$ *paval a ps*)) *s* = (*emb ps s*)(*x* := *aval a* (*emb ps s*))
using *assms*
unfolding *emb_def* **apply** *auto* **apply** (*rule ext*)
apply(*case_tac v*=*x*)
apply(*simp add*: *paval_aval*)
by (*simp*)

7.2.2 Big step Semantics on partial states

inductive
big_step_t_part :: *com* × *partstate* ⇒ *nat* ⇒ *partstate* ⇒ *bool* (‹_ $\Rightarrow_A$ _ $\Downarrow$ _› *55*)
where
Skip: (*SKIP*,*s*) $\Rightarrow_A$ *Suc 0* $\Downarrow$ *s* |
Assign: $[\![$ *vars a* $\cup$ {*x*} $\subseteq$ *dom ps*; *paval a ps* = *v* ; *ps'* = *ps*(*x* $\mapsto$ *v*) $]\!]$ $\Longrightarrow$ (*x* ::= *a*,*ps*) $\Rightarrow_A$ *Suc 0* $\Downarrow$ *ps'* |

Seq: $[\![$ (*c1*,*s1*) $\Rightarrow_A$ *x* $\Downarrow$ *s2*; (*c2*,*s2*) $\Rightarrow_A$ *y* $\Downarrow$ *s3* ; *z*=*x*+*y* $]\!]$ $\Longrightarrow$ (*c1*;;*c2*, *s1*) $\Rightarrow_A$ *z* $\Downarrow$ *s3* |

IfTrue: $[\![$ *vars b* $\subseteq$ *dom ps* ; *dom ps'* = *dom ps* ; *pbval b ps*; (*c1*,*ps*) $\Rightarrow_A$ *x* $\Downarrow$ *ps'*; *y*=*x*+*1* $]\!]$ $\Longrightarrow$ (*IF b THEN c1 ELSE c2*, *ps*) $\Rightarrow_A$ *y* $\Downarrow$ *ps'* |
IfFalse: $[\![$ *vars b* $\subseteq$ *dom ps* ; *dom ps'* = *dom ps* ; ¬ *pbval b ps*; (*c2*,*ps*) $\Rightarrow_A$ *x* $\Downarrow$ *ps'*; *y*=*x*+*1* $]\!]$ $\Longrightarrow$ (*IF b THEN c1 ELSE c2*, *ps*) $\Rightarrow_A$ *y* $\Downarrow$ *ps'* |
WhileFalse: $[\![$ *vars b* $\subseteq$ *dom s*; ¬ *pbval b s* $]\!]$ $\Longrightarrow$ (*WHILE b DO c*,*s*) $\Rightarrow_A$ *Suc 0* $\Downarrow$ *s* |
WhileTrue: $[\![$ *pbval b s1*; *vars b* $\subseteq$ *dom s1*; (*c*,*s1*) $\Rightarrow_A$ *x* $\Downarrow$ *s2*; (*WHILE b DO c*, *s2*) $\Rightarrow_A$ *y* $\Downarrow$ *s3*; *1*+*x*+*y*=*z* $]\!]$
$\Longrightarrow$ (*WHILE b DO c*, *s1*) $\Rightarrow_A$ *z* $\Downarrow$ *s3*

declare *big_step_t_part.intros* [*intro*]

inductive_cases *Skip_tE3*[*elim*!]: (*SKIP*,*s*) $\Rightarrow_A$ *x* ⇓ *t*
thm *Skip_tE3*
inductive_cases *Assign_tE3*[*elim*!]: (*x* ::= *a*,*s*) $\Rightarrow_A$ *p* ⇓ *t*
thm *Assign_tE3*
inductive_cases *Seq_tE3*[*elim*!]: (*c1*;;*c2*,*s1*) $\Rightarrow_A$ *p* ⇓ *s3*
thm *Seq_tE3*
inductive_cases *If_tE3*[*elim*!]: (*IF b THEN c1 ELSE c2*,*s*) $\Rightarrow_A$ *x* ⇓ *t*
thm *If_tE3*
inductive_cases *While_tE3*[*elim*]: (*WHILE b DO c*,*s*) $\Rightarrow_A$ *x* ⇓ *t*
thm *While_tE3*

lemmas *big_step_t_part_induct* = *big_step_t_part.induct*[*split_format*(*complete*)]

lemma *big_step_t3_post_dom_conv*: (*c*,*ps*) $\Rightarrow_A$ *t* ⇓ *ps'* ⟹ *dom ps'* = *dom ps*
apply(*induct rule*: *big_step_t_part_induct*) **apply** (*auto simp*: *sep_disj_fun_def plus_fun_def*)
  apply *metis* **done**

lemma *add_update_distrib*: *ps1 x1* = *Some y* ⟹ *ps1* ## *ps2* ⟹ *vars x2* ⊆ *dom ps1* ⟹ *ps1*(*x1* ↦ *paval x2 ps1*) + *ps2* = (*ps1* + *ps2*)(*x1* ↦ *paval x2 ps1*)
  apply (*rule ext*)
  apply (*auto simp*: *sep_disj_fun_def plus_fun_def*)
  by (*metis disjoint_iff_not_equal domI domain_conv*)

lemma *paval_extend*: *ps1* ## *ps2* ⟹ *vars a* ⊆ *dom ps1* ⟹ *paval a* (*ps1* + *ps2*) = *paval a ps1*
apply(*induct a*) **apply** (*auto simp*: *sep_disj_fun_def domain_conv*)
 by (*metis domI map_add_comm map_add_dom_app_simps*(*1*) *option.sel plus_fun_conv*)

lemma *pbval_extend*: *ps1* ## *ps2* ⟹ *vars b* ⊆ *dom ps1* ⟹ *pbval b* (*ps1* + *ps2*) = *pbval b ps1*
apply(*induct b*) **by** (*auto simp*: *paval_extend*)

lemma *Framer*: (*C*, *ps1*) $\Rightarrow_A$ *m* ⇓ *ps1'* ⟹ *ps1* ## *ps2* ⟹ (*C*, *ps1* + *ps2*) $\Rightarrow_A$ *m* ⇓ *ps1'*+*ps2*
proof (*induct rule*: *big_step_t_part_induct*)
  case (*Skip s*)
  then show *?case* **by** (*auto simp*: *big_step_t_part.Skip*)

```
next
  case (Assign a x ps v ps')
  show ?case apply(rule big_step_t_part.Assign)
    using Assign
     apply (auto simp: plus_fun_def)
    apply(rule ext)
    apply(case_tac xa=x)
   subgoal apply auto subgoal using paval_extend[unfolded plus_fun_def]
by auto
       unfolding sep_disj_fun_def
       by (metis disjoint_iff_not_equal domI domain_conv)
   subgoal by auto
     done
next
  case (IfTrue b ps ps' c1 x y c2)
  then show ?case apply (auto ) apply(subst big_step_t_part.IfTrue)
         apply (auto simp: pbval_extend)
      subgoal      by (auto simp: plus_fun_def)
      subgoal      by (auto simp: plus_fun_def)
      subgoal      by (auto simp: plus_fun_def)
          done
next
  case (IfFalse b ps ps' c2 x y c1)
  then show ?case apply (auto ) apply(subst big_step_t_part.IfFalse)
         apply (auto simp: pbval_extend)
      subgoal      by (auto simp: plus_fun_def)
      subgoal      by (auto simp: plus_fun_def)
      subgoal      by (auto simp: plus_fun_def)
          done
next
  case (WhileFalse b s c)
  then show ?case apply(subst big_step_t_part.WhileFalse)
      subgoal      by (auto simp: plus_fun_def)
      subgoal      by (auto simp: pbval_extend)
          by auto
next
  case (WhileTrue b s1 c x s2 y s3 z)
  from big_step_t3_post_dom_conv[OF WhileTrue(3)]  have dom s2 =
dom s1 by auto
  with WhileTrue(8) have s2 ## ps2 unfolding sep_disj_fun_def do-
main_conv by auto
 with WhileTrue show ?case apply auto apply(subst big_step_t_part.WhileTrue)
      subgoal      by (auto simp: pbval_extend)
      subgoal      by (auto simp: plus_fun_def)
```

apply (*auto*) **done**
next
case (*Seq c1 s1 x s2 c2 y s3 z*)
from *big_step_t3_post_dom_conv*[*OF Seq*(*1*)] **have** *dom s2* = *dom s1* **by** *auto*
with *Seq*(*6*) **have** *s2* ## *ps2* **unfolding** *sep_disj_fun_def domain_conv* **by** *auto*
with *Seq* **show** *?case* **apply** (*subst big_step_t_part.Seq*)
by *auto*
qed

lemma *Framer2*: $(C,\ ps1) \Rightarrow_A m \Downarrow ps1' \Longrightarrow ps1\ \#\#\ ps2 \Longrightarrow ps = ps1 + ps2 \Longrightarrow ps' = ps1'{+}ps2 \Longrightarrow (C,\ ps) \Rightarrow_A m \Downarrow ps'$
using *Framer* **by** *auto*

7.2.3 Relation to BigStep Semantic on full states

lemma *paval_aval_part*: *paval a* (*part s*) = *aval a s*
apply(*induct a*) **by** (*auto simp*: *part_def*)
lemma *pbval_bval_part*: *pbval b* (*part s*) = *bval b s*
apply(*induct b*) **by** (*auto simp*: *paval_aval_part*)

lemma *part_paval_aval*: *part* (*s*(*x* := *aval a s*)) = (*part s*)(*x* ↦ *paval a* (*part s*))
apply(*rule ext*)
apply(*case_tac xa=x*)
unfolding *part_def* **apply** *auto* **by** (*metis* (*full_types*) *domIff map_le_def map_le_substate_conv option.distinct*(*1*) *part_def paval_aval2 subsetI*)

lemma *full_to_part*: $(C,\ s) \Rightarrow m \Downarrow s' \Longrightarrow (C,\ part\ s) \Rightarrow_A m \Downarrow part\ s'$
apply(*induct rule*: *big_step_t_induct*)
using *Skip* **apply** *simp*
apply (*subst Assign*)
using *part_paval_aval* **apply**(*simp_all add*:)
apply(*rule Seq*) **apply** *auto*
apply(*rule IfTrue*) **apply** (*auto simp*: *pbval_bval_part*)
apply(*rule IfFalse*) **apply** (*auto simp*: *pbval_bval_part*)
apply(*rule WhileFalse*) **apply** (*auto simp*: *pbval_bval_part*)
apply(*rule WhileTrue*) **apply** (*auto simp*: *pbval_bval_part*)
done

lemma $part\_to\_full'$: $(C,\ ps) \Rightarrow_A m \Downarrow ps' \Longrightarrow (C,\ emb\ ps\ s) \Rightarrow m \Downarrow emb\ ps'\ s$
proof (*induct rule*: *big_step_t_part_induct*)
  case (*Assign a x ps v ps'*)
  have *z*: $paval\ a\ ps = aval\ a\ (emb\ ps\ s)$
    apply(*rule paval_aval_vars*) **using** *Assign*(*1*) **by** *auto*
  have *g* :$emb\ ps'\ s = (emb\ ps\ s)(x{:=}aval\ a\ (emb\ ps\ s)\ )$
    apply(*simp only*: *Assign z*[*symmetric*])
      unfolding *emb_def* **by** *auto*
  show *?case* **apply**(*simp only*: *g*) **by**(*rule big_step_t.Assign*)
qed (*auto simp*: *pbval_bval_vars*[*symmetric*])

lemma $part\_to\_full$: $(C,\ part\ s) \Rightarrow_A m \Downarrow part\ s' \Longrightarrow (C,\ s) \Rightarrow m \Downarrow s'$
proof −
  assume $(C,\ part\ s) \Rightarrow_A m \Downarrow part\ s'$
  then have $(C,\ emb\ (part\ s)\ s) \Rightarrow m \Downarrow emb\ (part\ s')\ s$ **by** (*rule part_to_full'*)
  then show $(C,\ s) \Rightarrow m \Downarrow s'$ **by** *auto*
qed

lemma $part\_full\_equiv$: $(C,\ s) \Rightarrow m \Downarrow s' \longleftrightarrow (C,\ part\ s) \Rightarrow_A m \Downarrow part\ s'$
using *part_to_full full_to_part* **by** *metis*

7.2.4 more properties

lemma $big\_step\_t3\_gt0$: $(C,\ ps) \Rightarrow_A x \Downarrow ps' \Longrightarrow x > 0$
apply(*induct rule*: *big_step_t_part_induct*) **apply** *auto* **done**

lemma $big\_step\_t3\_same$: $(C,\ ps) \Rightarrow_A m \Downarrow ps'$ ==> $ps = ps'\ on\ UNIV - lvars\ C$
apply(*induct rule*: *big_step_t_part_induct*) **by** (*auto simp*: *sep_disj_fun_def plus_fun_def*)

lemma $avalDirekt3\_correct$: $(x ::= N\ v,\ ps) \Rightarrow_A m \Downarrow ps' \Longrightarrow paval'\ a\ ps = Some\ v \Longrightarrow (x ::= a,\ ps) \Rightarrow_A m \Downarrow ps'$
apply(*auto*) **apply**(*subst Assign*) **by** (*auto simp*: *paval_paval' paval'dom*)

7.3 Partial State

lemma
  fixes $h :: (vname \Rightarrow val\ option) * nat$
  shows $(P ** Q ** H)\ h = (Q ** H ** P)\ h$
  by (*simp add*: *sep_conj_ac*)

lemma *separate_othogonal_commuted'*: **assumes**
$\bigwedge ps\ n.\ P\ (ps,n) \Longrightarrow ps = 0$
$\bigwedge ps\ n.\ Q\ (ps,n) \Longrightarrow n = 0$
shows $(P ** Q)\ s \longleftrightarrow P\ (0,snd\ s) \wedge Q\ (fst\ s,0)$
using *assms* **unfolding** *sep_conj_def* **by** *force*

lemma *separate_othogonal_commuted*: **assumes**
$\bigwedge ps\ n.\ P\ (ps,n) \Longrightarrow ps = 0$
$\bigwedge ps\ n.\ Q\ (ps,n) \Longrightarrow n = 0$
shows $(P ** Q)\ (ps,n) \longleftrightarrow P\ (0,n) \wedge Q\ (ps,0)$
using *assms* **unfolding** *sep_conj_def* **by** *force*

lemma *separate_othogonal*: **assumes**
$\bigwedge ps\ n.\ P\ (ps,n) \Longrightarrow n = 0$
$\bigwedge ps\ n.\ Q\ (ps,n) \Longrightarrow ps = 0$
shows $(P ** Q)\ (ps,n) \longleftrightarrow P\ (ps,0) \wedge Q\ (0,n)$
using *assms* **unfolding** *sep_conj_def* **by** *force*

lemma assumes $((\lambda(s,\ n).\ P\ (s,\ n) \wedge vars\ b \subseteq dom\ s) \wedge* (\lambda(s,\ c).\ s = 0 \wedge c = Suc\ 0))\ (ps,\ n)$
shows $\exists\ n'.\ P\ (ps,\ n') \wedge vars\ b \subseteq dom\ ps \wedge n = Suc\ n'$
proof −
from *assms* **obtain** *x y* **where** $x\ \#\#\ y$ **and** $(ps,\ n) = x + y$
and *2*: $(case\ x\ of\ (s,\ n) \Rightarrow P\ (s,\ n) \wedge vars\ b \subseteq dom\ s)$
and $(case\ y\ of\ (s,\ c) \Rightarrow s = 0 \wedge c = Suc\ 0)$
unfolding *sep_conj_def* **by** *blast*
then have $y = (0,\ Suc\ 0)$ **and** *f*: $fst\ x = ps$ **and** *n*: $n = snd\ x + Suc\ 0$
by *auto*

with *2* **have** $P\ (ps,\ snd\ x) \wedge vars\ b \subseteq dom\ ps \wedge n = Suc\ (snd\ x)$
by *auto*
then show *?thesis* **by** *simp*
qed

7.4 Dollar and Pointsto

definition *dollar* :: $nat \Rightarrow assn2$ (‹\$›) **where**
$dollar\ q = (\%(s,c).\ s = 0 \wedge c=q)$

lemma *sep_reorder_dollar_aux*:

NO_MATCH ($\$X$) $A \Longrightarrow (\$B ** A) = (A ** \$B)$
($\$X ** \$Y) = \$(X+Y)$
apply (*auto simp*: *sep_simplify*)
unfolding *dollar_def sep_conj_def sep_disj_prod_def sep_disj_nat_def*
by *auto*

lemmas *sep_reorder_dollar* = *sep_conj_assoc sep_reorder_dollar_aux*

lemma *stardiff*: **assumes** ($P \wedge* \$m$) ($ps$, n)
shows P: P (ps, $n - m$) **and** $m \leq n$ **using** *assms* **unfolding** *sep_conj_def dollar_def* **by** *auto*

lemma [*simp*]: ($Q ** \$0$) = Q **unfolding** *dollar_def sep_conj_def sep_disj_prod_def sep_disj_nat_def*
by *auto*

definition *embP* :: (*partstate* ⇒ *bool*) ⇒ *partstate* × *nat* ⇒ *bool* **where**
embP P = (%(s,n). $P\ s \wedge n = 0$)

lemma *orthogonal_split*: **assumes** (*embP Q* $\wedge*$ \$ n) = (*embP P* $\wedge*$ \$ m)

shows ($Q = P \wedge n = m$) $\vee$ $Q = (\lambda s.\ False) \wedge P = (\lambda s.\ False)$
using *assms* **unfolding** *embP_def dollar_def* **apply** (*auto intro*!: *ext*)
unfolding *sep_conj_def* **apply** *auto* **unfolding** *sep_disj_prod_def plus_prod_def*
apply (*metis fst_conv snd_conv*)+ **done**

lemma *F*: **assumes** (*embP Q* $\wedge*$ \$ n) = (*embP P* $\wedge*$ \$ m)
obtains (*blub*) $Q = P$ **and** $n = m$ |
(*da*) $Q = (\lambda s.\ False)$ **and** $P = (\lambda s.\ False)$
using *assms orthogonal_split* **by** *auto*

lemma *T*: **assumes** (*embP Q* $\wedge*$ \$ n) = (*embP P* $\wedge*$ \$ m)
obtains (*blub*) x::*nat* **where** $Q = P$ **and** $n = m$ **and** $x=x$ |
(*da*) $Q = (\lambda s.\ False)$ **and** $P = (\lambda s.\ False)$
using *assms orthogonal_split* **by** *auto*

definition *pointsto* :: *vname* ⇒ *val* ⇒ *assn2* (‹_ ↪ _› [*56*,*51*] *56*) **where**
$v \hookrightarrow n$ = (%(s,c). $s = [v \mapsto n] \wedge c=0$)

notation *pred_ex* (**binder** ‹∃› *10*)

definition *maps_to_ex* :: *vname* ⇒ *assn2* (‹_ ↪ −› [*56*] *56*)
  where $x \hookrightarrow - \equiv \exists y.\ x \hookrightarrow y$

fun *lmaps_to_ex* :: *vname set* ⇒ *assn2* **where**
  lmaps_to_ex xs = (%(*s*,*c*). *dom s* = *xs* ∧ *c* = *0*)

lemma $(x \hookrightarrow -)\ (s,n) \Longrightarrow x \in dom\ s$
  unfolding *maps_to_ex_def pointsto_def* **by** *auto*

fun *lmaps_to_axpr* :: *bexp* ⇒ *bool* ⇒ *assnp* **where**
  lmaps_to_axpr b bv = (%*ps*. *vars b* ⊆ *dom ps* ∧ *pbval b ps* = *bv*)

definition *lmaps_to_axpr′* :: *bexp* ⇒ *bool* ⇒ *assnp* **where**
  lmaps_to_axpr′ b bv = *lmaps_to_axpr b bv*

7.5 Frame Inference

definition *Frame* **where** *Frame P Q F* ≡ ∀ *s*. (*P imp* (*Q* ** *F*)) *s*
definition *Frame′* **where** *Frame′ P P′ Q F* ≡ ∀ *s*. ((*P′* ** *P*) *imp* (*Q* ** *F*)) *s*

definition *cnv* **where** *cnv x y* == *x* = *y*

lemma *cnv_I*: *cnv x x*
  unfolding *cnv_def* **by** *simp*

lemma *Frame′_conv*: *Frame P Q F* = *Frame′* (*P* ** □) □ (*Q* ** □) *F*
  unfolding *Frame_def Frame′_def* **apply** *auto* **done**

lemma *Frame′I*: *Frame′* (*P* ** □) □ (*Q* ** □) *F* ⟹ *cnv F F′* ⟹ *Frame P Q F′*
  unfolding *Frame_def Frame′_def cnv_def* **apply** *auto* **done**

lemma *FrameD*: **assumes** *Frame P Q F P s*
  shows (*F* ** *Q*) *s*
  using *assms* **unfolding** *Frame_def* **by** (*auto simp*: *sep_conj_commute*)

lemma *Frame′_match*: *Frame′* (*P* ** *P′*) □ *Q F* ⟹ *Frame′* (*x* ↪ *v* ** *P*) *P′* (*x* ↪ *v* ** *Q*) *F*

unfolding *Frame_def Frame'_def* **apply** (*auto simp*: *sep_conj_ac*)
by (*metis* (*no_types*, *opaque_lifting*) *prod.collapse sep.mult_assoc sep_conj_impl1*)

lemma *R*: **assumes** $\bigwedge s.$ (*A imp B*) *s* **shows** ((*A* ∗∗ \$*n*) *imp* (*B* ∗∗ \$*n*)) *s*

proof (*safe*)
assume (*A* ∧∗ \$ *n*) *s*
then obtain *h1 h2* **where** *A*: *A h1* **and** *n*: \$*n h2* **and** *disj*: *h1* ## *h2*
s = *h1*+*h2* **unfolding** *sep_conj_def* **by** *blast*
from *assms A* **have** *B*: *B h1* **by** *auto*
show (*B* ∗∗ \$*n*) *s* **using** *B n disj* **unfolding** *sep_conj_def* **by** *blast*
qed

lemma *Frame'_matchdollar*: **assumes** *Frame'* (*P* ∗∗ *P'* ∗∗ \$(*n*−*m*)) □ *Q*
F **and** *nm*: *n*≥*m*
shows *Frame'* (\$*n* ∗∗ *P*) *P'* (\$*m* ∗∗ *Q*) *F*
using *assms*(*1*) **unfolding** *Frame_def Frame'_def* **apply** (*auto simp*:
sep_conj_ac)
proof (*goal_cases*)
case (*1 a b*)
have *g*: ((*P* ∧∗ *P'* ∧∗ \$ *n*) *imp* (*F* ∧∗ *Q* ∧∗ \$ *m*)) (*a*, *b*)
⟷ (((*P* ∧∗ *P'* ∧∗ \$(*n*−*m*)) ∗∗ \$*m*) *imp* ((*F* ∧∗ *Q*) ∧∗ \$ *m*)) (*a*, *b*)
by(*simp add*: *nm sep_reorder_dollar*)
have ((*P* ∧∗ *P'* ∧∗ \$ *n*) *imp* (*F* ∧∗ *Q* ∧∗ \$ *m*)) (*a*, *b*)
apply(*subst g*)
apply(*rule R*) **using** *1*(*1*) **by** *auto*
then have (*P* ∧∗ *P'* ∧∗ \$ *n*) (*a*, *b*) ⟶ (*F* ∧∗ *Q* ∧∗ \$ *m*) (*a*, *b*)
by *blast*
then show *?case* **using** *1*(*2*) **by** *auto*
qed

lemma *Frame'_nomatch*: *Frame'* *P* (*p* ∗∗ *P'*) (*x* ↪ *v* ∗∗ *Q*) *F* ⟹ *Frame'*
(*p* ∗∗ *P*) *P'* (*x* ↪ *v* ∗∗ *Q*) *F*
unfolding *Frame'_def* **by** (*auto simp*: *sep_conj_ac*)

lemma *Frame'_nomatchempty*: *Frame'* *P* *P'* (*x* ↪ *v* ∗∗ *Q*) *F* ⟹ *Frame'*
(□ ∗∗ *P*) *P'* (*x* ↪ *v* ∗∗ *Q*) *F*
unfolding *Frame'_def* **by** (*auto simp*: *sep_conj_ac*)

lemma *Frame'_end*: *Frame'* *P* □ □ *P*
unfolding *Frame'_def* **by** (*auto simp*: *sep_conj_ac*)

schematic_goal *Frame* $(x \hookrightarrow v1 \wedge\!* \ y \hookrightarrow v2)$ $(x \hookrightarrow ?v)$ *?F*
apply(*rule Frame'I*) **apply**(*simp only*: *sep_conj_assoc*)
apply(*rule Frame'_match*)
apply(*rule Frame'_end*) **apply**(*simp only*: *sep_conj_ac sep_conj_empty'* *sep_conj_empty*) **apply**(*rule cnv_I*) **done**

schematic_goal *Frame* $(x \hookrightarrow v1 \wedge\!* \ y \hookrightarrow v2)$ $(y \hookrightarrow ?v)$ *?F*
apply(*rule Frame'I*) **apply**(*simp only*: *sep_conj_assoc*)
apply(*rule Frame'_end Frame'_match Frame'_nomatchempty Frame'_nomatch*; (*simp only*: *sep_conj_assoc*)*?*)+
apply(*simp only*: *sep_conj_ac sep_conj_empty' sep_conj_empty*) **apply**(*rule cnv_I*)
done

method *frame_inference_init* = (*rule Frame'I*, (*simp only*: *sep_conj_assoc*)*?*)

method *frame_inference_solve* = (*rule Frame'_matchdollar Frame'_end Frame'_match Frame'_nomatchempty Frame'_nomatch*; (*simp only*: *sep_conj_assoc*)*?*)+

method *frame_inference_cleanup* = ((*simp only*: *sep_conj_ac sep_conj_empty'* *sep_conj_empty*)*?*; *rule cnv_I*)

method *frame_inference* = (*frame_inference_init*, (*frame_inference_solve*; *fail*), (*frame_inference_cleanup*; *fail*))
method *frame_inference_debug* = (*frame_inference_init*, *frame_inference_solve*)

7.5.1 tests

schematic_goal *Frame* $(x \hookrightarrow v1 \wedge\!* \ y \hookrightarrow v2)$ $(y \hookrightarrow ?v)$ *?F*
by *frame_inference*

schematic_goal *Frame* $(x \hookrightarrow v1 ** P ** \square ** y \hookrightarrow v2 \wedge\!* \ z \hookrightarrow v2 ** Q)$ $(z \hookrightarrow ?v ** y \hookrightarrow ?v2)$ *?F*
by *frame_inference*

schematic_goal $1 \leq v \Longrightarrow$ *Frame* $(\$ \ (2 * v) \wedge\!* \ ''x'' \hookrightarrow int \ v)$ $(\$ \ 1 \wedge\!* \ ''x'' \hookrightarrow ?d)$ *?F*
apply(*rule Frame'I*) **apply**(*simp only*: *sep_conj_assoc*)
apply(*rule Frame'_matchdollar Frame'_end Frame'_match Frame'_nomatchempty Frame'_nomatch*; (*simp only*: *sep_conj_assoc*)*?*)+
apply (*simp only*: *sep_conj_ac sep_conj_empty' sep_conj_empty*)*?*

apply (*rule cnv_I*) **done**

schematic_goal $0 < v \Longrightarrow$ *Frame* (\$ ($2 * v$) $\wedge*$ $''x'' \hookrightarrow$ *int v*) (\$ 1 $\wedge*$ $''x'' \hookrightarrow$ *?d*) *?F*
  apply *frame_inference* **done**

7.6 Expression evaluation

definition *symeval* **where** *symeval P e v* $\equiv$ ($\forall$ *s n. P* (*s*,*n*) $\longrightarrow$ *paval′ e s* = *Some v*)
definition *symevalb* **where** *symevalb P e v* $\equiv$ ($\forall$ *s n. P* (*s*,*n*) $\longrightarrow$ *pbval′ e s* = *Some v*)

lemma *symeval_c*: *symeval P* (*N v*) *v*
 unfolding *symeval_def* **apply** *auto* **done**

lemma *symeval_v*: **assumes** *Frame P* ($x \hookrightarrow v$) *F*
 shows *symeval P* (*V x*) *v*
 unfolding *symeval_def* **apply** *auto*
 apply (*drule FrameD*[*OF assms*]) **unfolding** *sep_conj_def pointsto_def*

 apply (*auto simp*: *plus_fun_conv*) **done**

lemma *symeval_plus*: **assumes** *symeval P e1 v1 symeval P e2 v2*
 shows *symeval P* (*Plus e1 e2*) (*v1*+*v2*)
 using *assms* **unfolding** *symeval_def* **by** *auto*

lemma *symevalb_c*: *symevalb P* (*Bc v*) *v*
 unfolding *symevalb_def* **apply** *auto* **done**

lemma *symevalb_and*: **assumes** *symevalb P e1 v1 symevalb P e2 v2*
 shows *symevalb P* (*And e1 e2*) (*v1* $\wedge$ *v2*)
 using *assms* **unfolding** *symevalb_def* **by** *auto*

lemma *symevalb_not*: **assumes** *symevalb P e v*
 shows *symevalb P* (*Not e*) ($\neg$ *v*)
 using *assms* **unfolding** *symevalb_def* **by** *auto*

lemma *symevalb_less*: **assumes** *symeval P e1 v1 symeval P e2 v2*
 shows *symevalb P* (*Less e1 e2*) (*v1* < *v2*)
 using *assms* **unfolding** *symevalb_def symeval_def* **by** *auto*

lemmas *symeval* = *symeval_c symeval_v symeval_plus symevalb_c symevalb_and symevalb_not symevalb_less*

schematic_goal *symevalb* (($x \hookrightarrow v1$) ** ($y \hookrightarrow v2$)) (*Less* (*Plus* (*V x*) (*V y*)) (*N 5*)) *?g*
  apply(*rule symeval* | *frame_inference*)+ **done**

end

8 Hoare Logic based on Separation Logic and Time Credits

theory *SepLog_Hoare*
 imports *Big_StepT_Partial SepLogAdd/Sep_Algebra_Add*
begin

8.1 Definition of Validity

definition *hoare3_valid* :: *assn2* ⇒ *com* ⇒ *assn2* ⇒ *bool*
 (‹$\models_3$ {(1_)}/ (_)/ { (1_)}› *50*) **where**
$\models_3$ { P } c { Q } $\longleftrightarrow$
 ($\forall ps\ n.\ P\ (ps,n)$
 $\longrightarrow (\exists ps'\ m.\ ((c,ps) \Rightarrow_A m \Downarrow ps')$
 $\wedge\ n \geq m \wedge Q\ (ps',\ n-m))$)

lemma *alternative*: $\models_3$ { P } c { Q } $\longleftrightarrow$
 ($\forall ps\ n.\ P\ (ps,n)$
 $\longrightarrow (\exists ps'\ t\ n'.\ ((c,ps) \Rightarrow_A t \Downarrow ps')$
 $\wedge\ n=n'+t \wedge Q\ (ps',\ n'))$)
proof *rule*
 assume $\models_3$ {P} c { Q}
 then have P: ($\forall ps\ n.\ P\ (ps,n) \longrightarrow (\exists ps'\ m.\ ((c,ps) \Rightarrow_A m \Downarrow ps') \wedge n \geq m \wedge Q\ (ps',\ n-m))$) **unfolding** *hoare3_valid_def*.
 show $\forall ps\ n.\ P\ (ps,\ n) \longrightarrow (\exists ps'\ m\ e.\ (c,\ ps) \Rightarrow_A m \Downarrow ps' \wedge n = e + m \wedge Q\ (ps',\ e))$
 proof (*safe*)
  fix *ps n*
  assume $P\ (ps,\ n)$
   with P **obtain** ps' m **where** Z: $((c,ps) \Rightarrow_A m \Downarrow ps')$ $n \geq m$ $Q\ (ps',\ n-m)$ **by** *blast*
  show $\exists ps'\ m\ e.\ (c,\ ps) \Rightarrow_A m \Downarrow ps' \wedge n = e + m \wedge Q\ (ps',\ e)$
  apply(*rule exI*[**where** $x=ps'$])

apply(*rule exI*[**where** $x{=}m$])
apply(*rule exI*[**where** $x{=}n{-}m$]) **using** *Z* **by** *auto*
qed
next
assume $\forall ps\ n.\ P\ (ps,\ n) \longrightarrow (\exists ps'\ m\ e.\ (c,\ ps) \Rightarrow_A m \Downarrow ps' \wedge n = e + m \wedge Q\ (ps',\ e))$
then show $\models_3 \{\ P\ \}\ c\ \{\ Q\ \}$ **unfolding** *hoare3_valid_def*
by *fastforce*
qed

8.2 Hoare Rules

inductive
hoareT3 :: *assn2* ⇒ *com* ⇒ *assn2* ⇒ *bool* (‹⊢₃ ({(1_)}/ (_)/ { (1_)})› 50)
where

Skip: $\vdash_3 \{\$1\}\ SKIP\ \{\ \$0\}$ |

Assign: $\vdash_3 \{lmaps\_to\_expr\_x\ x\ a\ v\ {**}\ \$1\}\ x{::=}a\ \{\ (\%(s,c).\ dom\ s = vars\ a - \{x\} \wedge c = 0)\ {**}\ x \hookrightarrow v\ \}$ |

If: $[\![\ \vdash_3 \{\ \lambda(s,n).\ P\ (s,n) \wedge lmaps\_to\_axpr\ b\ True\ s\ \}\ c_1\ \{\ Q\ \};$
$\vdash_3 \{\ \lambda(s,n).\ P\ (s,n) \wedge lmaps\_to\_axpr\ b\ False\ s\ \}\ c_2\ \{\ Q\ \}\ ]\!]$
$\Longrightarrow \vdash_3 \{\ (\lambda(s,n).\ P\ (s,n) \wedge vars\ b \subseteq dom\ s)\ {**}\ \$1\}\ IF\ b\ THEN\ c_1\ ELSE\ c_2\ \{\ Q\}$ |

Frame: $[\![\ \ \vdash_3 \{\ P\ \}\ C\ \{\ Q\ \ \}\ ]\!]$
$\Longrightarrow \vdash_3 \{\ P\ {**}\ F\ \}\ C\ \{\ Q\ {**}\ F\ \}$ |

Seq: $[\![\ \vdash_3 \{\ P\ \}\ C_1\ \{\ Q\ \}\ ;\ \vdash_3 \{\ Q\ \}\ C_2\ \{\ R\ \ \}\ ]\!]$
$\Longrightarrow \vdash_3 \{\ P\ \}\ C_1\ ;;\ C_2\ \{\ R\ \}$ |

While: $[\![\ \vdash_3 \{\ (\lambda(s,n).\ P\ (s,n) \wedge lmaps\_to\_axpr\ b\ True\ s)\ \}\ C\ \{\ P\ \ {**}\ \$1\ \}\ ]\!]$
$\Longrightarrow \vdash_3 \{\ (\lambda(s,n).\ P\ (s,n) \wedge\ \ vars\ b \subseteq dom\ s)\ {**}\ \$1\ \}\ WHILE\ b\ DO\ C\ \{\ \ \lambda(s,n).\ P\ (s,n) \wedge lmaps\_to\_axpr\ b\ False\ s\ \}$ |

conseq: $[\![\ \vdash_3 \{P\}c\{Q\}\ ;\ \bigwedge s.\ P'\ s \Longrightarrow P\ s\ ;\ \bigwedge s.\ Q\ s \Longrightarrow Q'\ s\ ]\!] \Longrightarrow$
$\vdash_3 \{P'\}c\{\ Q'\}$ |

normalize: $[\![\ \ \vdash_3 \{\ P\ {**}\ \$m\ \}\ C\ \{\ Q\ \ {**}\ \$n\ \};\ n{\le}m\ ]\!]$
$\Longrightarrow \vdash_3 \{\ P\ {**}\ \$(m{-}n)\ \}\ C\ \{\ Q\ \}$ |

constancy: $\llbracket \quad \vdash_3 \{ P \} C \{ Q \}; \bigwedge ps\ ps'.\ ps = ps'\ on\ UNIV - lvars\ C \Longrightarrow R\ ps = R\ ps' \rrbracket$
$\Longrightarrow \vdash_3 \{ \%(ps,n).\ P\ (ps,n) \wedge R\ ps \}\ C\ \{ \%(ps,n).\ Q\ (ps,n) \wedge R\ ps \}$ |

Assign''': $\vdash_3 \{ \ \$1 ** (x \hookrightarrow ds) \}\ x ::= (N\ v)\ \{ \ (x \hookrightarrow v) \}$ |

Assign'''': $\llbracket\ symeval\ P\ a\ v; \vdash_3 \{P\}\ x ::= (N\ v)\ \{Q'\} \ \rrbracket \Longrightarrow \vdash_3 \{P\}\ x ::= a\ \{Q'\}$ |

Assign4: $\vdash_3 \{ (\lambda(ps,t).\ x{\in}dom\ ps \wedge vars\ a \subseteq dom\ ps \wedge Q\ (ps(x{\mapsto}(paval\ a\ ps)),t)\ ) ** \$1\}\ x{::=}a\ \{ Q \}$ |

False: $\vdash_3 \{ \ \lambda(ps,n).\ False \}\ c\ \{ \ Q \}$ |

pureI: $(P \Longrightarrow \ \vdash_3 \{ Q\}\ c\ \{ R\}) \Longrightarrow \ \vdash_3 \{\uparrow P ** Q\}\ c\ \{ R\}$

Derived Rules

lemma *Frame_R*: **assumes** $\vdash_3 \{ P \}\ C\ \{ Q \}$ *Frame P' P F*
shows $\vdash_3 \{ P' \}\ C\ \{ Q ** F \}$
apply(*rule conseq*) **apply**(*rule Frame*) **apply**(*rule assms*(*1*))
using *assms*(*2*) **unfolding** *Frame_def* **by** *auto*

lemma *strengthen_post*: **assumes** $\vdash_3 \{P\}c\{Q\}\ \bigwedge s.\ Q\ s \Longrightarrow Q'\ s$
shows $\vdash_3 \{P\}c\{ Q'\}$
apply(*rule conseq*)
apply (*rule assms*(*1*))
apply *simp* **apply** *fact* **done**

lemma *weakenpre*: $\llbracket \vdash_3 \{P\}c\{Q\}\ ;\ \bigwedge s.\ P'\ s \Longrightarrow P\ s\ \rrbracket \Longrightarrow$
$\vdash_3 \{P'\}c\{ Q\}$
using *conseq* **by** *auto*

8.3 Soundness Proof

lemma *exec_preserves_disj*: $(c,ps) \Rightarrow_A t \Downarrow ps' \Longrightarrow ps''\ \#\#\ ps \Longrightarrow ps''\ \#\#\ ps'$
apply(*drule big_step_t3_post_dom_conv*)
unfolding *sep_disj_fun_def domain_conv* **by** *auto*

lemma *FrameRuleSound*: **assumes** $\models_3 \{ P \}\ C\ \{ Q \}$
shows $\models_3 \{ P ** F \}\ C\ \{ Q ** F \}$

proof –
  {
    fix *ps n*
    assume $(P \wedge\!* \ F)$ (ps, n)
    then obtain *pP nP pF nF* **where** *orth*: (pP, nP) ## (pF, nF) **and** *add*: $(ps, n) = (pP, nP) + (pF, nF)$
        and *P*: P (pP, nP) **and** *F*: F (pF, nF) **unfolding** *sep_conj_def* **by** *auto*
    from *assms*[*unfolded hoare3_valid_def*] *P*
      obtain pP' *m* **where** *ex*: $(C, pP) \Rightarrow_A m \Downarrow pP'$ **and** *m*: $m \le nP$ **and** *Q*: Q $(pP', nP - m)$ **by** *blast*

    have *exF*: $(C, ps) \Rightarrow_A m \Downarrow pP' + pF$
      using *Framer2 ex orth add* **by** *auto*
    have *QF*: $(Q \wedge\!* \ F)$ $(pP' + pF, n - m)$
      unfolding *sep_conj_def*
        apply(*rule exI*[**where** $x{=}(pP'{,}nP{-}m)$])
        apply(*rule exI*[**where** $x{=}(pF{,}nF)$])
        using *orth exec_preserves_disj*[*OF ex*] *add m F Q* **by** (*auto simp add*: *sep_add_ac*)
    have $(C, ps) \Rightarrow_A m \Downarrow pP'{+}pF \wedge m \le n \wedge (Q \wedge\!* \ F)$ $(pP'{+}pF, n - m)$
      using *QF exF add m* **by** *auto*
    hence $\exists ps'\ m.\ (C, ps) \Rightarrow_A m \Downarrow ps' \wedge m \le n \wedge (Q \wedge\!* \ F)$ $(ps', n - m)$ **by** *auto*
  }
  thus *?thesis* **unfolding** *hoare3_valid_def* **by** *auto*
qed

theorem *hoare3_sound*: **assumes** $\vdash_3 \{\ P\ \} c \{\ Q\ \}$
  shows $\models_3 \{\ P\ \}\ c\ \{\ Q\ \}$ **using** *assms*
proof(*induction rule*: *hoareT3.induct*)
  case (*False c Q*)
  then show *?case* **by** (*auto simp*: *hoare3_valid_def*)
next
  case *Skip*
  then show *?case* **by** (*auto simp*: *hoare3_valid_def dollar_def*)
next
  case (*Assign4 x a Q*)
  then show *?case*
    apply (*auto simp*: *dollar_def sep_conj_def hoare3_valid_def*)
      subgoal for *ps b y*
        apply(*rule exI*[**where** $x{=}ps(x \mapsto paval\ a\ ps)$])
          apply(*rule exI*[**where** $x{=}Suc\ 0$]) **by** *auto*

done
next
case (*Assign x a v*)
then show *?case* **unfolding** *hoare3_valid_def* **apply** *auto* **apply** (*auto simp*: *dollar_def*) **apply** (*subst* (*asm*) *separate_othogonal*)
apply *simp_all* **apply**(*intro exI conjI*)
apply(*rule big_step_t_part.Assign*)
apply (*auto simp*: *pointsto_def*) **unfolding** *sep_conj_def*
subgoal for *ps* **apply**(*rule exI*[**where** *x*=((%*y*. *if y*=*x then None else ps y*) , *0*)])
apply(*rule exI*[**where** *x*=((%*y*. *if y* = *x then Some* (*paval a ps*) *else None*),*0*)])
apply (*auto simp*: *sep_disj_prod_def sep_disj_fun_def plus_fun_def*)
apply (*smt domIff domain_conv*)
apply (*metis domI insertE option.simps*(*3*))
using *domIff* **by** *fastforce*
done
next
case (*If P b* c_1 *Q* c_2)
from *If*(*3*)[*unfolded hoare3_valid_def*]
have *T*: $\bigwedge ps\ n.\ P\ (ps,\ n) \Longrightarrow vars\ b \subseteq dom\ ps \Longrightarrow pbval\ b\ ps$
$\Longrightarrow (\exists ps'\ m.\ (c_1,\ ps) \Rightarrow_A m \Downarrow ps' \wedge m \le n \wedge Q\ (ps',\ n{-}m))$ **by** *auto*
from *If*(*4*)[*unfolded hoare3_valid_def*]
have *F*: $\bigwedge ps\ n.\ P\ (ps,\ n) \Longrightarrow vars\ b \subseteq dom\ ps \Longrightarrow \neg\ pbval\ b\ ps$
$\Longrightarrow (\exists ps'\ m.\ (c_2,\ ps) \Rightarrow_A m \Downarrow ps' \wedge m \le n \wedge Q\ (ps',\ n{-}m))$ **by** *auto*
show *?case* **unfolding** *hoare3_valid_def* **apply** *auto* **apply** (*auto simp*: *dollar_def*)
proof (*goal_cases*)
case (*1 ps n*)
then obtain n' **where** *P*: $P\ (ps,\ n')$ **and** *dom*: $vars\ b \subseteq dom\ ps$ **and** *Suc*: $n = Suc\ n'$ **unfolding** *sep_conj_def*
by *force*
show *?case*
proof(*cases pbval b ps*)
case *True*
with *T*[*OF P dom*] **obtain** ps' *m* **where** *d*: $(c_1,\ ps) \Rightarrow_A m \Downarrow ps'$
and *m1*: $m \le n'$ **and** *Q*: $Q\ (ps',\ n'{-}m)$ **by** *blast*
from *big_step_t3_post_dom_conv*[*OF d*] **have** *klong*: $dom\ ps' = dom\ ps$ **.**
show *?thesis*
apply(*rule exI*[**where** $x{=}ps'$]) **apply**(*rule exI*[**where** *x*=*m*+*1*])
apply *safe*
apply(*rule big_step_t_part.IfTrue*)
apply (*rule dom*)

apply *fact*
apply (*rule True*)
apply (*rule d*)
apply *simp*
using *m1 Suc* **apply** *simp*
using *Q Suc* **by** *force*
next
case *False*
with *F*[*OF P dom*] **obtain** *ps'* *m* **where** *d*: $(c_2, ps) \Rightarrow_A m \Downarrow ps'$
and *m1*: $m \leq n'$ **and** *Q*: $Q\ (ps', n'{-}m)$ **by** *blast*
from *big_step_t3_post_dom_conv*[*OF d*] **have** *dom ps'* = *dom ps* **.**
show *?thesis*
apply(*rule exI*[**where** *x=ps'*]) **apply**(*rule exI*[**where** *x=m+1*])
apply *safe*
apply(*rule big_step_t_part.IfFalse*)
apply *fact*
apply *fact*
apply (*rule False*)
apply (*rule d*)
apply *simp*
using *m1 Suc* **apply** *simp*
using *Q Suc* **by** *force*
qed
qed
next
case (*Frame P C Q F*)
then show *?case* **using** *FrameRuleSound* **by** *auto*
next
case (*Seq P* C_1 *Q* C_2 *R*)
show *?case* **unfolding** *hoare3_valid_def*
proof (*safe*, *goal_cases*)
case (*1 ps n*)
with *Seq*(*3*)[*unfolded hoare3_valid_def*] **obtain** *ps'* *m* **where** *C1*: $(C_1, ps) \Rightarrow_A m \Downarrow ps'$
and *m*: $m \leq n$ **and** *Q*: $Q\ (ps', n - m)$ **by** *blast*
with *Seq*(*4*)[*unfolded hoare3_valid_def*] **obtain** *ps''* *m'* **where** *C2*: $(C_2, ps') \Rightarrow_A m' \Downarrow ps''$
and *m'*: $m' \leq n - m$ **and** *R*: $R\ (ps'', n - m - m')$ **by** *blast*
have *a*: $(C_1;; C_2, ps) \Rightarrow_A m + m' \Downarrow ps''$ **apply**(*rule big_step_t_part.Seq*)
apply *fact+* **by** *simp*
have *b*: $m + m' \leq n$ **using** *m' m* **by** *auto*
have *c*: $R\ (ps'', n - (m + m'))$ **using** *R* **by** *simp*
show *?case* **apply**(*rule exI*[**where** *x=ps''*]) **apply**(*rule exI*[**where** *x=m+m'*])

using *a b c* **by** *auto*
qed
next
case (*While P b C*)
show *?case* **unfolding** *hoare3_valid_def* **apply** *auto* **apply** (*auto simp*: *dollar_def*)
proof (*goal_cases*)
case (*1 ps n*)
from *1* **show** *?case*
proof(*induct n arbitrary*: *ps rule*: *less_induct*)
case (*less x ps3*)

show *?case*
proof(*cases pbval b ps3*)
case *True*
— prepare premise to obtain ...
from *less*(*2*) **obtain** x' **where** *P*: *P* (*ps3*, x') **and** *dom*: *vars b* $\subseteq$ *dom ps3* **and** *Suc*: x = *Suc* x' **unfolding** *sep_conj_def dollar_def* **by** *auto*
from *P dom True* **have**
g: ((λ(*s*, *n*). *P* (*s*, *n*) $\wedge$ *lmaps_to_axpr b True s*)) (*ps3*, x')
unfolding *dollar_def* **by** *auto*
— ... the loop body from the outer IH
from *While*(*2*)[*unfolded hoare3_valid_def*] *g* **obtain** *ps3'* x'' **where** *C*: (*C*, *ps3*) $\Rightarrow_A$ x'' $\Downarrow$ *ps3'* **and** *x*: $x'' \leq x'$ **and** P': (*P* $\wedge$∗ \$ *1*) (*ps3'*, $x' - x''$) **by** *blast*
then obtain x''' **where** P'': *P* (*ps3'*, x''') **and** Suc'': $x' - x''$ = *Suc* x''' **unfolding** *sep_conj_def dollar_def* **by** *auto*

from *C big_step_t3_post_dom_conv* **have** *dom ps3* = *dom ps3'* **by** *simp*
with *dom* **have** *dom'*: *vars b* $\subseteq$ *dom ps3'* **by** *auto*

— prepare premises to ...
from *C big_step_t3_gt0* **have** *gt0*: $x'' > 0$ **by** *auto*
have $\exists$ *ps' m*. (*WHILE b DO C*, *ps3'*) $\Rightarrow_A$ *m* $\Downarrow$ *ps'* $\wedge$ $m \leq (x - (1 + x''))$ $\wedge$ *P* (*ps'*, $(x - (1 + x'')) - m$) $\wedge$ *vars b* $\subseteq$ *dom ps'* $\wedge$ $\neg$ *pbval b ps'*
apply(*rule less*(*1*))
using *gt0 x Suc* **apply** *simp*
using *dom' Suc P'* **unfolding** *dollar_def sep_conj_def*
by *force*
— ... obtain the tail of the While loop from the inner IH
then obtain *ps3''* *m* **where** *w*: ((*WHILE b DO C*, *ps3'*) $\Rightarrow_A$ *m* $\Downarrow$ *ps3''*)

and m'': $m \leq (x - (1 + x''))$ **and** P'': P $(ps3'', (x - (1 + x'')) - m)$
and dom'': *vars* $b \subseteq$ *dom* $ps3''$ **and** b'': $\neg$ *pbval* b $ps3''$ **by** *auto*

— combine body and tail to one loop unrolling:
— - the Bigstep Semantic
have *BigStep*: $(WHILE\ b\ DO\ C,\ ps3) \Rightarrow_A 1 + x'' + m \Downarrow ps3''$
apply(*rule big_step_t_part.WhileTrue*)
apply (*fact True*) **apply** (*fact dom*) **apply** (*fact C*) **apply** (*fact w*) **by** *simp*
— - the TimeBound
have *TimeBound*: $1 + x'' + m \leq x$
using m'' Suc'' *Suc* **by** *simp*
— - the invariantPreservation
have *invariantPreservation*: P $(ps3'', x - (1 + x'' + m))$ **using** P'' m'' **by** *auto*

— finally combine BigStep Semantic, TimeBound, invariantPreservation
show *?thesis*
apply(*rule exI*[**where** $x=ps3''$])
apply(*rule exI*[**where** $x=1 + x'' + m$])
using *BigStep TimeBound invariantPreservation* dom'' b'' **by** *blast*
next
case *False*
from *less*(*2*) **obtain** x' **where** P: P $(ps3, x')$ **and** *dom*: *vars* $b \subseteq$ *dom ps3* **and** *Suc*: $x = Suc\ x'$ **unfolding** *sep_conj_def*
by *force*
show *?thesis*
apply(*rule exI*[**where** $x=ps3$])
apply(*rule exI*[**where** $x=Suc\ 0$]) **apply** *safe*
apply (*rule big_step_t_part.WhileFalse*)
subgoal using *dom* **by** *simp*
apply *fact*
using *Suc* **apply** *simp*
using *P Suc* **apply** *simp*
using *dom* **apply** *auto*
using *False* **apply** *auto* **done**
qed
qed
qed

next
  case (*conseq P c Q P′ Q′*)
  then show *?case* **unfolding** *hoare3_valid_def* **by** *metis*
next
  case (*normalize P m C Q n*)
  then show *?case* **unfolding** *hoare3_valid_def*
  apply(*safe*) **proof** (*goal_cases*)
    case (*1 ps N*)
    have *Q2*: P (*ps*, $N - (m - n)$) **apply**(*rule stardiff*) **by** *fact*
    have *mn*: $m - n \leq N$ **apply**(*rule stardiff*(*2*)) **by** *fact*
    have *P*: (P ∧∗ \$ m) (*ps*, $N - (m - n) + m$) **unfolding** *sep_conj_def dollar_def*
      apply(*rule exI*[**where** *x*=(*ps*,$N - (m - n)$)]) **apply**(*rule exI*[**where** *x*=(*0*,*m*)])
      apply(*auto simp*: *sep_disj_prod_def sep_disj_nat_def*) **by** *fact*
    have $N - (m - n) + m = N + n$ **using** *normalize*(*2*)
      using *mn* **by** *auto*

    from *P 1*(*3*) **obtain** *ps′ m′* **where** (C, *ps*) $\Rightarrow_A$ $m' \Downarrow$ *ps′* **and** *m′*: $m' \leq N - (m - n) + m$ **and** *Q*: (Q ∧∗ \$ n) (*ps′*, $N - (m - n) + m - m'$) **by** *blast*
    have *Q2*: Q (*ps′*, ($N - (m - n) + m - m'$) $- n$) **apply**(*rule stardiff*) **by** *fact*
    have *nm2*: $n \leq (N - (m - n) + m - m')$ **apply**(*rule stardiff*(*2*)) **by** *fact*
    show *?case*
      apply(*rule exI*[**where** *x*=*ps′*]) **apply**(*rule exI*[**where** *x*=*m′*])
      apply(*safe*)
        apply *fact*
        using *Q2*
        using ‹$N - (m - n) + m = N + n$› *m′ nm2* **apply** *linarith*
        using *Q2* ‹$N - (m - n) + m = N + n$› **by** *auto*
    qed
next
  case (*constancy P C Q R*)
  from *constancy*(*3*) **show** *?case* **unfolding** *hoare3_valid_def*
 apply *safe* **proof** (*goal_cases*)
    case (*1 ps n*)
    then obtain *ps′ m* **where** *C*: (C, *ps*) $\Rightarrow_A$ $m \Downarrow$ *ps′* **and** *m*: $m \leq n$ **and** *Q*: Q (*ps′*, $n - m$) **by** *blast*
    from *C big_step_t3_same* **have** *ps = ps′ on UNIV* − *lvars C* **by** *auto*
    with *constancy*(*2*) *1*(*3*) **have** *R ps′* **by** *auto*

   show *?case* **apply**(*rule exI*[**where** *x*=*ps′*]) **apply**(*rule exI*[**where** *x*=*m*])

apply(*safe*)
apply *fact+* **done**
qed
next
case (*Assign''' x ds v*)
then show *?case*
unfolding *hoare3_valid_def* **apply** *auto*
subgoal for *ps n* **apply**(*rule exI*[**where** *x=ps(x↦v)*])
apply(*rule exI*[**where** *x=Suc 0*])
apply *safe*
apply(*rule big_step_t_part.Assign*)
apply (*auto*)
subgoal apply(*subst* (*asm*) *separate_othogonal_commuted'*) **by**(*auto simp*: *dollar_def pointsto_def*)
subgoal apply(*subst* (*asm*) *separate_othogonal_commuted'*) **by**(*auto simp*: *dollar_def pointsto_def*)
subgoal apply(*subst* (*asm*) *separate_othogonal_commuted'*) **by**(*auto simp*: *dollar_def pointsto_def*)
done
done

next
case (*Assign'''' P a v x Q'*)
show *?case*

unfolding *hoare3_valid_def* **apply** *auto*
proof (*goal_cases*)
case (*1 ps n*)
with *Assign''''*(*3*)[*unfolded hoare3_valid_def*] **obtain** *ps' m*
where $(x ::= N\ v,\ ps) \Rightarrow_A m \Downarrow ps'\ m \leq n\ Q'\ (ps',\ n - m)$ **by** *metis*
from *1*(*1*) *Assign''''*(*1*)[*unfolded symeval_def*] **have** *paval' a ps = Some v* **by** *auto*
show *?case* **apply**(*rule exI*[**where** *x=ps'*]) **apply**(*rule exI*[**where** *x=m*])
apply *safe*
apply(*rule avalDirekt3_correct*)
apply *fact+* **done**
qed
next
case (*pureI P Q c R*)
then show *?case* **unfolding** *hoare3_valid_def* **by** *auto*
qed

8.4 Completeness

definition *wp3* :: *com* ⇒ *assn2* ⇒ *assn2* (‹wp_3›) **where**
wp_3 *c Q* = (λ(*s*,*n*). ∃ *t m*. *n*≥*m* ∧ (*c*,*s*) $\Rightarrow_A$ *m* ⇓ *t* ∧ *Q* (*t*,*n*−*m*))

lemma *wp3_SKIP*[*simp*]: wp_3 *SKIP Q* = (*Q* ∗∗ $*1*)
 apply (*auto intro*!: *ext simp*: *wp3_def*)
  unfolding *sep_conj_def dollar_def sep_disj_prod_def sep_disj_nat_def*
apply *auto* **apply** *force*
   subgoal for *t n* **apply**(*rule exI*[**where** *x*=*t*]) **apply**(*rule exI*[**where**
x=*Suc 0*])
    using *big_step_t_part.Skip* **by** *auto*
   done

lemma *wp3_Assign*[*simp*]: wp_3 (*x* ::= *e*) *Q* = ((λ(*ps*,*t*). *vars e* ∪ {*x*} ⊆
dom ps ∧ *Q* (*ps*(*x* ↦ *paval e ps*),*t*)) ∗∗ $*1*)
 apply (*auto intro*!: *ext simp*: *wp3_def*)
 unfolding *sep_conj_def* **apply** (*auto simp*: *sep_disj_prod_def sep_disj_nat_def*
dollar_def) **apply** *force*
   by *fastforce*

lemma *wpt_Seq*[*simp*]: wp_3 (c_1;;c_2) *Q* = wp_3 c_1 (wp_3 c_2 *Q*)
 apply (*auto simp*: *wp3_def fun_eq_iff*)
 subgoal for *a b t m1 s2 m2*
    apply(*rule exI*[**where** *x*=*s2*])
    apply(*rule exI*[**where** *x*=*m1*])
     apply *simp*
    apply(*rule exI*[**where** *x*=*t*])
    apply(*rule exI*[**where** *x*=*m2*])
    apply *simp* **done**
 subgoal for *s m t′ m1 t m2*
    apply(*rule exI*[**where** *x*=*t*])
    apply(*rule exI*[**where** *x*=*m1*+*m2*])
   apply (*auto simp*: *big_step_t_part.Seq*) **done**
    done

lemma *wp3_If*[*simp*]:
 wp_3 (*IF b THEN* c_1 *ELSE* c_2) *Q* = ((λ(*ps*,*t*). *vars b* ⊆ *dom ps* ∧ wp_3 (*if*
pbval b ps then c_1 *else* c_2) *Q* (*ps*,*t*)) ∗∗ $*1*)
 apply (*auto simp*: *wp3_def fun_eq_iff*)
 unfolding *sep_conj_def* **apply** (*auto simp*: *sep_disj_prod_def sep_disj_nat_def*
dollar_def)
 subgoal for *a ba t x* **apply**(*rule exI*[**where** *x*=*ba* − *1*]) **apply** *auto*

apply(*rule exI*[**where** *x=t*]) **apply**(*rule exI*[**where** *x=x*]) **apply** *auto*
done
subgoal for *a ba t x* **apply**(*rule exI*[**where** *x=ba − 1*]) **apply** *auto*
apply(*rule exI*[**where** *x=t*]) **apply**(*rule exI*[**where** *x=x*]) **apply** *auto*
done
subgoal for *a ba t m*
apply(*rule exI*[**where** *x=t*]) **apply**(*rule exI*[**where** *x=Suc m*]) **apply** *auto*
apply(*cases pbval b a*)
subgoal apply *simp* **apply**(*subst big_step_t_part.IfTrue*) **using** *big_step_t3_post_dom_conv* **by** *auto*
subgoal apply *simp* **apply**(*subst big_step_t_part.IfFalse*) **using** *big_step_t3_post_dom_conv* **by** *auto*
done
done

lemma *sFTrue*: **assumes** *pbval b ps vars b* $\subseteq$ *dom ps*
shows wp_3 (*WHILE b DO c*) *Q* (*ps*, *n*) = ((λ(*ps*, *n*). *vars b* $\subseteq$ *dom ps* $\wedge$ (*if pbval b ps then* wp_3 *c* (wp_3 (*WHILE b DO c*) *Q*) (*ps*, *n*) *else Q* (*ps*, *n*))) $\wedge*$ \$ *1*) (*ps*, *n*)
(**is** *?wp* = (*?I* $\wedge*$ \$ *1*) _)
proof
assume wp_3 (*WHILE b DO c*) *Q* (*ps*, *n*)
from *this*[*unfolded wp3_def*] **obtain** *ps′′ tt* **where** *tn*: *tt* $\leq$ *n* **and** *w1*: (*WHILE b DO c*, *ps*) $\Rightarrow_A$ *tt* $\Downarrow$ *ps′′* **and** *Q*: *Q* (*ps′′*, *n − tt*) **by** *blast*
with *assms* **obtain** *t t′ ps′* **where** *w2*: (*WHILE b DO c*, *ps′*) $\Rightarrow_A$ *t′* $\Downarrow$ *ps′′* **and** *c*: (*c*, *ps*) $\Rightarrow_A$ *t* $\Downarrow$ *ps′* **and** *tt*: *tt=1+t+t′* **by** *auto*

from *tn* **obtain** *k* **where** *n*: *n=tt+k*
using *le_Suc_ex* **by** *blast*

from *assms* **show** (*?I* $\wedge*$ \$ *1*) (*ps*,*n*)
unfolding *sep_conj_def dollar_def wp3_def* **apply** *auto*
apply(*rule exI*[**where** *x=t+t′+k*])
apply *safe* **subgoal using** *n tt* **by** *auto*
apply(*rule exI*[**where** *x=ps′*])
apply(*rule exI*[**where** *x=t*])
using *c* **apply** *auto*
apply(*rule exI*[**where** *x=ps′′*])
apply(*rule exI*[**where** *x=t′*])
using *w2 Q n* **by** *auto*
next
assume (*?I* $\wedge*$ \$ *1*) (*ps*,*n*)
with *assms* **have** *Q*: wp_3 *c* (wp_3 (*WHILE b DO c*) *Q*) (*ps*, *n−1*) **and** *n*:

n≥*1* **unfolding** *dollar_def sep_conj_def* **by** *auto*
then obtain *t ps′ t′ ps′′* **where** *t*: *t* ≤ *n* − *1*
and *c*: (*c*, *ps*) $\Rightarrow_A$ *t* ⇓ *ps′* **and** *t′*: *t′* ≤ (*n*−*1*) − *t* **and** *w*: (*WHILE b DO c*, *ps′*) $\Rightarrow_A$ *t′* ⇓ *ps′′*
and *Q*: *Q* (*ps′′*, ((*n*−*1*) − *t*) − *t′*)
unfolding *wp3_def* **by** *auto*

show *?wp* **unfolding** *wp3_def*
apply *simp* **apply**(*rule exI*[**where** *x*=*ps′′*]) **apply**(*rule exI*[**where** *x*=*1*+*t*+*t′*])
apply *safe*
subgoal using *t t′ n* **by** *simp*
subgoal using *c w assms* **by** *auto*
subgoal using *Q t t′ n* **by** *simp*
done
qed

lemma *sFFalse*: **assumes** ~ *pbval b ps vars b* ⊆ *dom ps*
shows wp_3 (*WHILE b DO c*) *Q* (*ps*, *n*) = ((λ(*ps*, *n*). *vars b* ⊆ *dom ps* ∧ (*if pbval b ps then* wp_3 *c* (wp_3 (*WHILE b DO c*) *Q*) (*ps*, *n*) *else Q* (*ps*, *n*))) ∧∗ $ *1*) (*ps*, *n*)
(**is** *?wp* = (*?I* ∧∗ $ *1*) _)
proof
assume wp_3 (*WHILE b DO c*) *Q* (*ps*, *n*)
from *this*[*unfolded wp3_def*] **obtain** *ps′ t* **where** *tn*: *t* ≤ *n* **and** *w1*: (*WHILE b DO c*, *ps*) $\Rightarrow_A$ *t* ⇓ *ps′* **and** *Q*: *Q* (*ps′*, *n* − *t*) **by** *blast*
from *assms* **have** *w2*: (*WHILE b DO c*, *ps*) $\Rightarrow_A$ *1* ⇓ *ps* **by** *auto*
from *w1 w2 big_step_t_determ2* **have** *t1*: *t*=*1* **and** *pps*: *ps*=*ps′* **by** *auto*
from *assms* **show** (*?I* ∧∗ $ *1*) (*ps*,*n*)
unfolding *sep_conj_def dollar_def* **using** *t1 tn Q pps* **apply** *auto* **apply**(*rule exI*[**where** *x*=*n*−*1*]) **by** *auto*
next
assume (*?I* ∧∗ $ *1*) (*ps*,*n*)
with *assms* **have** *Q*: *Q*(*ps*,*n*−*1*) *n*≥*1* **unfolding** *dollar_def sep_conj_def* **by** *auto*
from *assms* **have** *w2*: (*WHILE b DO c*, *ps*) $\Rightarrow_A$ *1* ⇓ *ps* **by** *auto*
show *?wp* **unfolding** *wp3_def*
apply *auto* **apply**(*rule exI*[**where** *x*=*ps*]) **apply**(*rule exI*[**where** *x*=*1*])
using *Q w2* **by** *auto*
qed

lemma *sF′*: wp_3 (*WHILE b DO c*) *Q* (*ps*,*n*) = ((λ(*ps*, *n*). *vars b* ⊆ *dom ps* ∧ (*if pbval b ps then* wp_3 *c* (wp_3 (*WHILE b DO c*) *Q*) (*ps*, *n*) *else Q* (*ps*,

n))) $\wedge*$ \$ *1*) (ps,n)
  apply(*cases vars b* $\subseteq$ *dom ps*)
  subgoal apply(*cases pbval b ps*) **using** *sFTrue sFFalse* **by** *auto*
  subgoal **by** (*auto simp add*: *dollar_def wp3_def sep_conj_def*)
    done

lemma *sF*: wp_3 (*WHILE b DO c*) *Q s* = ((λ(*ps*, *n*). *vars b* $\subseteq$ *dom ps* $\wedge$ (*if pbval b ps then* wp_3 *c* (wp_3 (*WHILE b DO c*) *Q*) (*ps*, *n*) *else Q* (*ps*, *n*))) $\wedge*$ \$ *1*) *s*
 using *sF'*
 by (*metis* (*mono_tags*, *lifting*) *prod.case_eq_if prod.collapse sep_conj_impl1*)

lemma assumes $\bigwedge Q$. $\vdash_3$ {wp_3 *c Q*} *c* {*Q*}
  shows *WhileWpisPre*: $\vdash_3$ {wp_3 (*WHILE b DO c*) *Q*} *WHILE b DO c* { *Q*}
proof –
  define *I* **where** *I* $\equiv$ (λ(*ps*, *n*). *vars b* $\subseteq$ *dom ps* $\wedge$ (*if pbval b ps then* wp_3 *c* (wp_3 (*WHILE b DO c*) *Q*) (*ps*, *n*) *else Q* (*ps*, *n*)))

  from *assms*[**where** *Q*=(wp_3 (*WHILE b DO c*) *Q*)] **have**
    c: $\vdash_3$ {wp_3 *c* (wp_3 (*WHILE b DO c*) *Q*)} *c* {(wp_3 (*WHILE b DO c*) *Q*)}
.
  have *c'*: $\vdash_3$ { (λ(*s*,*n*). *I* (*s*,*n*) $\wedge$ *lmaps_to_axpr b True s*) } *c* { *I* $**$ \$*1* }
    apply(*rule conseq*)
      apply(*rule c*)
    subgoal apply *auto* **unfolding** *I_def* **by** *auto*
    subgoal **unfolding** *I_def* **using** *sF* **by** *auto*
    done

  from *hoareT3.While*[**where** *P*=*I*] *c'* **have**
    w: $\vdash_3$ { (λ(*s*,*n*). *I* (*s*,*n*) $\wedge$ *vars b* $\subseteq$ *dom s*) $**$ \$*1* } *WHILE b DO c* { λ(*s*,*n*). *I* (*s*,*n*) $\wedge$ *lmaps_to_axpr b False s* } **.**

  show $\vdash_3$ {wp_3 (*WHILE b DO c*) *Q*} *WHILE b DO c* { *Q*}
      apply(*rule conseq*)
      apply(*rule w*)
    subgoal using *sF I_def*
      by (*smt Pair_inject R case_prodE case_prodI2*)
    subgoal unfolding *I_def* **by** *auto*
    done
qed

lemma *wp3_is_pre*: $\vdash_3$ {wp_3 *c Q*} *c* { *Q*}
proof (*induction c arbitrary*: *Q*)
  case *SKIP*
  then show *?case* **apply** *auto*
    using *Frame*[**where** *F*=*Q* **and** *Q*=$*0* **and** *P*=$*1*, *OF Skip*]
      by (*auto simp*: *sep.add_ac*)
next
  case (*Assign x1 x2*)
  then show *?case* **using** *Assign4* **by** *simp*
next
  case (*Seq c1 c2*)
  then show *?case* **apply** *auto*
    apply(*subst hoareT3.Seq*[*rotated*]) **by** *auto*
next
  case (*If x1 c1 c2*)
  then show *?case* **apply** *auto*
   apply(*rule weakenpre*[*OF hoareT3.If*, **where** *P1*=%(*ps*,*n*). wp_3 (*if pbval x1 ps then c1 else c2*) *Q* (*ps*,*n*)])
      apply *auto*
    subgoal apply(*rule conseq*[**where** *P*=wp_3 *c1 Q* **and** *Q*=*Q*]) **by** *auto*
    subgoal apply(*rule conseq*[**where** *P*=wp_3 *c2 Q* **and** *Q*=*Q*]) **by** *auto*
  proof −
    fix *a b*
    assume ((λ(*ps*, *t*). *vars x1* $\subseteq$ *dom ps* $\wedge$ wp_3 (*if pbval x1 ps then c1 else c2*) *Q* (*ps*, *t*)) $\wedge$* $ (*Suc 0*)) (*a*, *b*)
    then show ((λ(*ps*, *t*). wp_3 (*if pbval x1 ps then c1 else c2*) *Q* (*ps*, *t*) $\wedge$ *vars x1* $\subseteq$ *dom ps*) $\wedge$* $ (*Suc 0*)) (*a*, *b*)
       unfolding *sep_conj_def* **apply** *auto* **apply**(*case_tac pbval x1 aa*) **apply** *auto* **done**
  qed
next
  case (*While b c*)
  with *WhileWpisPre* **show** *?case* **.**
qed

theorem *hoare3_complete*: $\models_3$ {*P*}*c*{*Q*} $\Longrightarrow$ $\vdash_3$ {*P*}*c*{*Q*}
apply(*rule conseq*[*OF wp3_is_pre*, **where** *Q'*=*Q* **and** *Q*=*Q*, *simplified*])
  apply(*auto simp*: *hoare3_valid_def wp3_def*)
  by *fast*

theorem *hoare3_sound_complete*: $\models_3$ {*P*}*c*{*Q*} $\longleftrightarrow$ $\vdash_3$ {*P*}*c*{*Q*}

using *hoare3_complete hoare3_sound* **by** *metis*

8.4.1 What about garbage collection?

definition *F* **where** *F C Q* = (%(*ps*,*n*). ∃ *ps1' ps2' m e1 e2*. (*C*, *ps*) $\Rightarrow_A$ *m* ⇓ *ps1'* + *ps2'* ∧ *ps1'* ## *ps2'* ∧ *n* = *e1* + *e2* + *m* ∧ *Q* (*ps1'*,*e1*))

lemma wp_3 *C* (*Q***(%_. *True*)) = *F C Q*
  apply *rule*
  unfolding *wp3_def sep_conj_def*
  unfolding *F_def* **apply** *auto*
  subgoal for *a b m aaa ba ab bb* **apply**(*rule exI*[**where** *x*=*aaa*])
    apply(*rule exI*[**where** *x*=*ab*]) **apply**(*rule exI*[**where** *x*=*m*])
   apply *auto* **apply**(*rule exI*[**where** *x*=*ba*]) **apply** *auto* **apply**(*rule exI*[**where** *x*=*bb*])
    apply *auto*
  done
  subgoal for *a ps1' ps2' m e1 e2*
    apply(*rule exI*[**where** *x*=*ps1'*+*ps2'*])
    apply(*rule exI*[**where** *x*=*m*]) **by** *auto*
  done

definition *hoareT3_validGC* :: *assn2* ⇒ *com* ⇒ *assn2* ⇒ *bool*
  (‹$\models_G$ {(*1*_)}/ (_)/ { (*1*_)}› *50*) **where**
$\models_G$ { *P* } *c* { *Q* } ⟷ $\models_3$ { *P* } *c* { *Q* ** (%_. *True*) }

end

8.5 Examples

theory *SepLog_Examples*
imports *SepLog_Hoare*
begin

8.5.1 nice example

lemmas *strongAssign* = *Assign''''*[*OF* _ *strengthen_post*, *OF* _ *Frame_R*, *OF* _ *Assign'''*]

lemma *myrule*: **assumes** *case s of* (*s*, *n*) ⇒ ($ (*2* ∗ *x*) ∧∗ ''*x*'' ↪ *int x*) (*s*, *n*) ∧ *lmaps_to_axpr'* (*Less* (*N 0*) (*V* ''*x*'')) *True s*
    and *symevalb* ($ (*2* ∗ *x*) ** ''*x*'' ↪ *int x*) (*Less* (*N 0*) (*V* ''*x*'')) *v*
    shows (↑(*v*=*True*) ** $ (*2* ∗ *x*) ** ''*x*'' ↪ *int x*) *s*
      using *assms* **unfolding** *symevalb_def lmaps_to_axpr'_def* **by** *auto*

```
fun sum :: int ⇒ int where
sum i = (if i ≤ 0 then 0 else sum (i − 1) + i)

abbreviation wsum ==
  WHILE Less (N 0) (V ''x'')
  DO (
     ''x'' ::= Plus (V ''x'') (N (− 1)))



lemma E4_R: ⊢3 {↑(v>0) ** $(2*v) ** pointsto ''x'' (int v) }
           ''x'' ::= Plus (V ''x'') (N (− 1))
          {↑(v>0) ** $(2*v−1) ** pointsto ''x'' (int v−1) }
     apply(rule pureI)
  apply(rule strongAssign)
    apply(rule symeval | frame_inference)+
     by (simp add:  sep_reorder_dollar )



lemma prod_0:
  shows (λ(s::char list ⇒ int option, c::nat). s = Map.empty ∧ c = 0) h
⟹ h = 0 by (auto simp: zero_prod_def zero_fun_def)



lemma example2: ⊢3 { (pointsto ''x'' n) ** (pointsto ''y'' n) ** $1 } ''x''
::= Plus (V ''x'') (N (− 1)) { (pointsto ''x'' (n−1)) ** (pointsto ''y'' n) }
  apply(rule conseq)
  apply(rule Frame[where F=(pointsto ''y'' n) and P=lmaps_to_expr_x
''x'' ( Plus (V ''x'') (N (− 1))) (n−1) ** $1])
    apply (rule Assign)
   apply (simp add: sep_conj_assoc)  apply (rule sep_conj_impl)
     apply auto[1]
    subgoal for s h unfolding pointsto_def apply auto
      by (meson option.distinct(1))
     apply (simp add: sep_conj_commute)
    apply simp apply (rule sep_conj_impl)
      apply auto[1]
     apply auto
      unfolding sep_conj_def
      using prod_0 by fastforce
```

end

9 Hoare Logic based on Separation Logic and Time Credits (big-O style)

theory *SepLogK_Hoare*
imports *Big_StepT Partial_Evaluation Big_StepT_Partial*
begin

9.1 Definition of Validity

definition *hoare3o_valid* :: $assn2 \Rightarrow com \Rightarrow assn2 \Rightarrow bool$
(‹$\models_{3'}$ {(1_)}/ (_)/ { (1_)}› 50) **where**
$\models_{3'}$ { P } c { Q } $\longleftrightarrow$
($\exists$ $k>0$. ($\forall ps\ n.\ P\ (ps,n)$
$\longrightarrow$ ($\exists ps'\ ps''\ m\ e\ e'$. (($c$,$ps$) $\Rightarrow_A$ $m \Downarrow ps' + ps''$)
$\wedge\ ps'$ ## $ps'' \wedge k * n = k * e + e' + m$
$\wedge\ Q\ (ps',e)$)))

9.2 Hoare Rules

inductive
hoare3a :: $assn2 \Rightarrow com \Rightarrow assn2 \Rightarrow bool$ (‹$\vdash_{3a}$ ({(1_)}/ (_)/ { (1_)})› 50)
where

Skip: $\vdash_{3a}$ {\$1} *SKIP* { \$0} |

Assign4: $\vdash_{3a}$ { ($\lambda(ps,t)$. $x \in dom\ ps \wedge vars\ a \subseteq dom\ ps \wedge Q\ (ps(x \mapsto (paval\ a\ ps)),t)$) ** \$1} x::=a { Q } |

If: $[\![$ $\vdash_{3a}$ { $\lambda(s,n)$. $P\ (s,n) \wedge$ *lmaps_to_axpr b True s* } c_1 { Q };
$\vdash_{3a}$ { $\lambda(s,n)$. $P\ (s,n) \wedge$ *lmaps_to_axpr b False s* } c_2 { Q } $]\!]$
$\Longrightarrow \vdash_{3a}$ { ($\lambda(s,n)$. $P\ (s,n) \wedge vars\ b \subseteq dom\ s$) ** \$1} *IF b THEN* c_1 *ELSE* c_2 { Q} |

Frame: $[\![$ $\vdash_{3a}$ { P } C { Q } $]\!]$
$\Longrightarrow \vdash_{3a}$ { P ** F } C { Q ** F } |

Seq: $[\![$ $\vdash_{3a}$ { P } C_1 { Q } ; $\vdash_{3a}$ { Q } C_2 { R } $]\!]$
$\Longrightarrow \vdash_{3a}$ { P } C_1 ;; C_2 { R } |

While: $\llbracket \vdash_{3a}$ { ($\lambda(s,n)$. *P* (*s*,*n*) $\wedge$ *lmaps_to_axpr b True s*) } *C* { ($\lambda(s,n)$. *P* (*s*,*n*) $\wedge$ *vars b* $\subseteq$ *dom s*) ** *$1* } $\rrbracket$
$\Longrightarrow \vdash_{3a}$ { ($\lambda(s,n)$. *P* (*s*,*n*) $\wedge$ *vars b* $\subseteq$ *dom s*) ** *$1* } *WHILE b DO C* { $\lambda(s,n)$. *P* (*s*,*n*) $\wedge$ *lmaps_to_axpr b False s* } |

conseqS: $\llbracket \vdash_{3a}$ {*P*}*c*{*Q*} ; $\bigwedge$*s n*. *P'* (*s*,*n*) $\Longrightarrow$ *P* (*s*,*n*) ; $\bigwedge$*s n*. *Q* (*s*,*n*) $\Longrightarrow$ *Q'* (*s*,*n*) $\rrbracket \Longrightarrow$
$\vdash_{3a}$ {*P'*}*c*{ *Q'*}

inductive
hoare3b :: *assn2* $\Rightarrow$ *com* $\Rightarrow$ *assn2* $\Rightarrow$ *bool* (‹$\vdash_{3b}$ ({(*1*_)}/ (_)/ { (*1*_)})› *50*)
where

import: $\vdash_{3a}$ {*P*} *c* { *Q*} $\Longrightarrow \vdash_{3b}$ {*P*} *c* { *Q*} |

conseq: $\llbracket \vdash_{3b}$ {*P*}*c*{*Q*} ; $\bigwedge$*s n*. *P'* (*s*,*n*) $\Longrightarrow$ *P* (*s*,*k*∗*n*) ; $\bigwedge$*s n*. *Q* (*s*,*n*) $\Longrightarrow$ *Q'* (*s*,*n div k*); *k*>*0* $\rrbracket \Longrightarrow$
$\vdash_{3b}$ {*P'*}*c*{ *Q'*}

inductive
hoare3' :: *assn2* $\Rightarrow$ *com* $\Rightarrow$ *assn2* $\Rightarrow$ *bool* (‹$\vdash_{3'}$ ({(*1*_)}/ (_)/ { (*1*_)})› *50*)
where

Skip: $\vdash_{3'}$ {*$1*} *SKIP* { *$0*} |

Assign: $\vdash_{3'}$ {*lmaps_to_expr_x x a v* ** *$1*} *x*::=*a* { (%(*s*,*c*). *dom s* = *vars a* − {*x*} $\wedge$ *c* = *0*) ** *x* $\hookrightarrow$ *v* } |

Assign': $\vdash_{3'}$ {*pointsto x v'* ** (*pointsto x v* $\longrightarrow$∗ *Q*) ** *$1*} *x*::= *N v* { *Q* } |

Assign2: $\vdash_{3'}$ {$\exists$ *v* . ((($\exists$ *v'*. *pointsto x v'*) ** (*pointsto x v* $\longrightarrow$∗ *Q*) ** *$1*)
and sep_true ** (%(*ps*,*n*). *vars a* $\subseteq$ *dom ps* $\wedge$ *paval a ps* = *v*))} *x*::= *a* { *Q* } |

If: $\llbracket \vdash_{3'} \{ \lambda(s,n).\ P\ (s,n) \wedge lmaps\_to\_axpr\ b\ True\ s \}\ c_1\ \{ Q \}$;
$\vdash_{3'} \{ \lambda(s,n).\ P\ (s,n) \wedge lmaps\_to\_axpr\ b\ False\ s \}\ c_2\ \{ Q \} \rrbracket$
$\Longrightarrow \vdash_{3'} \{ (\lambda(s,n).\ P\ (s,n) \wedge vars\ b \subseteq dom\ s) ** \$1\}\ IF\ b\ THEN\ c_1\ ELSE\ c_2\ \{ Q\}$ |

Frame: $\llbracket \vdash_{3'} \{ P \}\ C\ \{ Q \} \rrbracket$
$\Longrightarrow \vdash_{3'} \{ P ** F \}\ C\ \{ Q ** F \}$ |

Seq: $\llbracket \vdash_{3'} \{ P \}\ C_1\ \{ Q \}\ ;\ \vdash_{3'} \{ Q \}\ C_2\ \{ R \} \rrbracket$
$\Longrightarrow \vdash_{3'} \{ P \}\ C_1 ;;\ C_2\ \{ R \}$ |

While: $\llbracket \vdash_{3'} \{ (\lambda(s,n).\ P\ (s,n) \wedge lmaps\_to\_axpr\ b\ True\ s) \}\ C\ \{ (\lambda(s,n).\ P\ (s,n) \wedge\ vars\ b \subseteq dom\ s) ** \$1 \} \rrbracket$
$\Longrightarrow \vdash_{3'} \{ (\lambda(s,n).\ P\ (s,n) \wedge\ vars\ b \subseteq dom\ s) ** \$1 \}\ WHILE\ b\ DO\ C\ \{ \lambda(s,n).\ P\ (s,n) \wedge lmaps\_to\_axpr\ b\ False\ s \}$ |

conseq: $\llbracket \vdash_{3'} \{P\}c\{Q\}\ ;\ \bigwedge s\ n.\ P'\ (s,n) \Longrightarrow P\ (s,k*n)\ ;\ \bigwedge s\ n.\ Q\ (s,n) \Longrightarrow Q'\ (s,n\ div\ k);\ k>0 \rrbracket \Longrightarrow$
$\vdash_{3'} \{P'\}c\{ Q'\}$ |

normalize: $\llbracket \vdash_{3'} \{ P ** \$m \}\ C\ \{ Q ** \$n \};\ n \leq m \rrbracket$
$\Longrightarrow \vdash_{3'} \{ P ** \$(m-n) \}\ C\ \{ Q \}$ |

Assign''': $\vdash_{3'} \{ \$1 ** (x \hookrightarrow ds) \}\ x ::= (N\ v)\ \{ (x \hookrightarrow v) \}$ |

Assign'''': $\llbracket\ symeval\ P\ a\ v; \vdash_{3'} \{P\}\ x ::= (N\ v)\ \{Q'\} \rrbracket \Longrightarrow \vdash_{3'} \{P\}\ x ::= a\ \{Q'\}$ |

Assign4: $\vdash_{3'} \{ (\lambda(ps,t).\ x \in dom\ ps \wedge vars\ a \subseteq dom\ ps \wedge Q\ (ps(x \mapsto (paval\ a\ ps)),t)\ ) ** \$1\}\ x ::= a\ \{ Q \}$ |

False: $\vdash_{3'} \{ \lambda(ps,n).\ False \}\ c\ \{ Q \}$ |

pureI: $(\ P \Longrightarrow\ \vdash_{3'} \{ Q\}\ c\ \{ R\}) \Longrightarrow\ \vdash_{3'} \{\uparrow P ** Q\}\ c\ \{ R\}$

definition $A4 :: vname \Rightarrow aexp \Rightarrow assn2 \Rightarrow assn2$
where $A4\ x\ a\ Q = ((\lambda(ps,t).\ x \in dom\ ps \wedge vars\ a \subseteq dom\ ps \wedge Q\ (ps(x \mapsto (paval\ a\ ps)),t)\ ) ** \$1)$
definition $A2 :: vname \Rightarrow aexp \Rightarrow assn2 \Rightarrow assn2$
where $A2\ x\ a\ Q = (\exists\, v\ .\ (\ ((\exists\, v'.\ pointsto\ x\ v') ** (\ pointsto\ x\ v \longrightarrow* Q) ** \$1)$

and sep_true ∗∗ (%(*ps*,*n*). *vars a* ⊆ *dom ps* ∧ *paval a ps* = *v*
)))

lemma *A4 x a Q* (*ps*,*n*) ⟹ *A2 x a Q* (*ps*,*n*)
unfolding *A4_def A2_def sep_conj_def dollar_def sep_impl_def pointsto_def*
apply *auto*
  apply(*rule exI*[**where** *x*=*paval a ps*])
  apply *safe*
  subgoal for *n v*
    apply(*rule exI*[**where** *x*=[*x* ↦ *v*]::*partstate*])
    apply(*rule exI*[**where** *x*=*0*])
    apply *auto* **apply**(*rule exI*[**where** *x*=*ps*(*x*:=*None*)])
    apply *auto*
    unfolding *sep_disj_fun_def domain_conv* **apply** *auto*
    unfolding *plus_fun_conv* **apply** *auto*
      by (*simp add*: *map_add_upd_left map_upd_triv*)
  subgoal for *n v*
    apply(*rule exI*[**where** *x*=*0*])
    apply(*rule exI*[**where** *x*=*n*])
    apply(*rule exI*[**where** *x*=*ps*])
    by *auto*
  done

lemma *A2 x a Q* (*ps*,*n*) ⟹ *A4 x a Q* (*ps*,*n*)
 unfolding *A4_def A2_def sep_conj_def dollar_def sep_impl_def pointsto_def*
  apply (*auto simp*: *sep_disj_commute*)
  subgoal for *aa ba ab ac bc xa bd* **apply**(*rule exI*[**where** *x*=*bd*])
    by (*auto simp*: *sep_add_ac domain_conv sep_disj_fun_def*)
  subgoal for *aa ba ab ac bc xa bd* **apply**(*rule exI*[**where** *x*=*bd*])
    apply (*auto simp*: *sep_add_ac*)
    subgoal apply (*auto simp*:   *domain_conv sep_disj_fun_def*)
      by (*metis fun_upd_same none_def plus_fun_def*)
    subgoal
       by (*metis domD map_add_dom_app_simps*(*1*) *plus_fun_conv sub-
setCE*)
    subgoal
    proof −
      assume *a*: *ab* + [*x* ↦ *xa*] = *aa* + *ac*
      assume *b*: *ps* = *aa* + *ac* **and** *o*: *aa* ## *ac*
      then have *b*′: *ps* = *ac* + *aa* **by**(*simp add*: *sep_add_ac*)
      assume *vars*: *vars a* ⊆ *dom ac*
      have *pa*: *paval a ps* = *paval a ac* **unfolding** *b*′
      apply(*rule paval_extend*) **using** *o vars* **by** (*simp_all add*: *sep_add_ac*)

    have f: $\bigwedge f.\ (ab + [x \mapsto xa])(x \mapsto f) = ab + [x \mapsto f]$
      by (*simp add*: *plus_fun_conv*)

    assume $Q\ (ab + [x \mapsto paval\ a\ ac],\ bd)$
    thus $Q\ ((aa + ac)(x \mapsto paval\ a\ (aa + ac)),\ bd)$
      unfolding *b*[*symmetric*] *pa*
      unfolding *b a*[*symmetric*] *pa f* **by** *auto*
   qed
   done
 done

lemma *E_extendsR*: $\vdash_{3a}$ { P } c { F } $\Longrightarrow \vdash_{3'}$ { P } c { F }
 apply (*induct rule*: *hoare3a.induct*)
   apply(*intro hoare3'.Skip*)
   apply(*intro hoare3'.Assign4*)
   subgoal using *hoare3'.If* **by** *auto*
   subgoal using *hoare3'.Frame* **by** *auto*
   subgoal using *hoare3'.Seq* **by** *auto*
   subgoal using *hoare3'.While* **by** *auto*
   subgoal using *hoare3'.conseq*[**where** *k=1*] **by** *simp*
   done

lemma *E_extendsS*: $\vdash_{3b}$ { P } c { F } $\Longrightarrow \vdash_{3'}$ { P } c { F }
 apply (*induct rule*: *hoare3b.induct*)
   apply(*erule E_extendsR*)
   using *hoare3'.conseq* **by** *blast*

lemma *Skip'*: $P = (F ** \$1) \Longrightarrow \vdash_{3'}$ { P } *SKIP* { F }
 apply(*rule conseq*[**where** *k=1*])
   apply(*rule Frame*[**where** *F=F*])
    apply(*rule Skip*)
   by (*auto simp*: *sep_conj_ac*)

9.2.1 experiments with explicit and implicit GarbageCollection

lemma ($(\forall ps\ n.\ P\ (ps,n)$
 $\longrightarrow (\exists ps'\ ps''\ m\ e\ e'.\ ((c,ps) \Rightarrow_A m \Downarrow ps' + ps'')$
   $\wedge\ ps'\ \#\#\ ps'' \wedge\ n = e + e' + m$
   $\wedge\ Q\ (ps',e))))$
 $\longleftrightarrow\ (\forall ps\ n.\ P\ (ps,n) \longrightarrow (\exists ps'\ m\ e\ .\ ((c,ps) \Rightarrow_A m \Downarrow ps')\ \wedge\ n = e$

$+ m \wedge (Q ** (\lambda\_. \textit{True})) (ps',e)))$

proof (*safe*)
fix *ps n*
assume $\forall ps\ n.\ P\ (ps,\ n) \longrightarrow (\exists ps'\ ps''\ m\ e\ e'.\ (c,\ ps) \Rightarrow_A m \Downarrow ps' + ps'' \wedge ps'\ \#\#\ ps'' \wedge n = e + e' + m \wedge Q\ (ps',\ e))$
$P\ (ps,\ n)$
then obtain $ps'\ ps''\ m\ e\ e'$ **where** C: $(c,\ ps) \Rightarrow_A m \Downarrow ps' + ps'' \wedge ps'\ \#\#\ ps'' \wedge n = e + e' + m \wedge Q\ (ps',\ e)$ **by** *blast*
show $\exists ps'\ m\ e.\ (c,\ ps) \Rightarrow_A m \Downarrow ps' \wedge n = e + m \wedge (Q ** (\lambda\_. \textit{True}))\ (ps',e)$ **unfolding** *sep_conj_def*
apply(*rule exI*[**where** $x=ps' + ps''$])
apply(*rule exI*[**where** $x=m$])
apply(*rule exI*[**where** $x=e+e'$]) **using** C **by** *auto*
next
fix *ps n*
assume $\forall ps\ n.\ P\ (ps,n) \longrightarrow (\exists ps'\ m\ e\ .\ ((c,ps) \Rightarrow_A m \Downarrow ps') \wedge n = e + m \wedge (Q ** (\lambda\_. \textit{True}))\ (ps',e))$
$P\ (ps,\ n)$
then obtain $ps'\ m\ e$ **where** C: $((c,ps) \Rightarrow_A m \Downarrow ps') \wedge n = e + m$ **and** Q: $(Q ** (\lambda\_. \textit{True}))\ (ps',e)$ **by** *blast*
from Q **obtain** *ps1 ps2 e1 e2* **where** Q': $Q\ (ps1,e1)$ $ps'=ps1+ps2$ $ps1\#\#ps2$ $e=e1+e2$ **unfolding** *sep_conj_def* **by** *auto*
show $\exists ps'\ ps''\ m\ e\ e'.\ (c,\ ps) \Rightarrow_A m \Downarrow ps' + ps'' \wedge ps'\ \#\#\ ps'' \wedge n = e + e' + m \wedge Q\ (ps',\ e)$
apply(*rule exI*[**where** $x=ps1$])
apply(*rule exI*[**where** $x=ps2$])
apply(*rule exI*[**where** $x=m$])
apply(*rule exI*[**where** $x=e1$])
apply(*rule exI*[**where** $x=e2$]) **using** $C\ Q'$ **by** *auto*
qed

9.3 Soundness

theorem *hoareT_sound2_part*: **assumes** $\vdash_{3'} \{\ P\ \} c \{\ Q\ \}$
shows $\models_{3'} \{\ P\ \}\ c\ \{\ Q\ \}$ **using** *assms*
proof(*induction rule*: *hoare3'.induct*)
case (*conseq* $P\ c\ Q\ P'\ k1\ Q'$)
then obtain k **where** p: $\forall ps\ n.\ P\ (ps,\ n) \longrightarrow (\exists ps'\ ps''\ m\ e\ e'.\ ((c,ps) \Rightarrow_A m \Downarrow ps' + ps'') \wedge ps'\ \#\#\ ps'' \wedge k * n = k * e + e' + m \wedge Q\ (ps',e))$ **and** *gt0*: $k>0$
unfolding *hoare3o_valid_def* **by** *blast*

show *?case* **unfolding** *hoare3o_valid_def*
apply(*rule exI*[**where** $x=k*k1$])
apply *safe*
using *gt0 conseq*(*4*) **apply** *simp*
proof –
fix *ps n*
assume P' (*ps*,*n*)
with *conseq*(*2*) **have** P (*ps*, $k1*n$) **by** *simp*
with *p* **obtain** ps' ps'' *m e* e' **where** *pB*: (*c*, *ps*) $\Rightarrow_A$ *m* $\Downarrow$ ps' + ps'' **and** *orth*: ps' ## ps''
and *m*: $k * (k1 * n) = k*e + e' + m$ **and** *Q*: Q (ps', *e*) **by** *blast*

from *Q conseq*(*3*) **have** Q': Q' (ps', *e div k1*) **by** *auto*

have $k * k1 * n = k*e + e' + m$ **using** *m* **by** *auto*
also have ... = $k*(k1 * (e\ div\ k1) + e\ mod\ k1) + e' + m$ **using** *mod_mult_div_eq* **by** *simp*
also have ... = $k*k1*(e\ div\ k1) + (k*(e\ mod\ k1) + e') + m$
by (*metis add.assoc distrib_left mult.assoc*)
finally have $k * k1 * n = k * k1 * (e\ div\ k1) + (k * (e\ mod\ k1) + e') + m$ **.**

show $\exists ps'\ ps''\ m\ e\ e'.\ (c, ps) \Rightarrow_A m \Downarrow ps' + ps'' \wedge ps' \#\# ps'' \wedge k * k1 * n = k * k1 * e + e' + m \wedge Q' (ps', e)$
apply(*rule exI*[**where** $x=ps'$])
apply(*rule exI*[**where** $x=ps''$])
apply(*rule exI*[**where** $x=m$])
apply(*rule exI*[**where** $x=e\ div\ k1$])
apply(*rule exI*[**where** $x=k * (e\ mod\ k1) + e'$])
apply *safe* **apply** *fact* **apply** *fact* **apply** *fact* **apply** *fact* **done**
qed
next
case (*Frame P c Q F*)
from *Frame*(*2*)[*unfolded hoare3o_valid_def*] **obtain** *k*
where *hyp*: $\forall ps\ n.\ P\ (ps, n) \longrightarrow (\exists ps'\ ps''\ m\ e\ e'.\ ((c,ps) \Rightarrow_A m \Downarrow ps' + ps'') \wedge ps' \#\# ps'' \wedge k * n = k * e + e' + m \wedge Q\ (ps',e))$
and *k*: $k>0$
unfolding *hoare3o_valid_def* **by** *blast*

show *?case* **unfolding** *hoare3o_valid_def* **apply**(*rule exI*[**where** $x=k$]) **using** *k* **apply** *simp*
proof(*safe*)
fix *ps n*

assume $(P \wedge\!* F)$ (ps, n)
then obtain *ps1 ps2* **where** *orth*: *ps1* ## *ps2* **and** *add*: $(ps, n) = ps1 + ps2$
and *P*: *P ps1* **and** *F*: *F ps2* **unfolding** *sep_conj_def* **by** *blast*
from *hyp P* **have** $(\exists\, ps'\ ps''\ m\ e\ e'.\ ((c,fst\ ps1) \Rightarrow_A m \Downarrow ps' + ps'') \wedge ps'\ \#\#\ ps'' \wedge k * snd\ ps1 = k * e + e' + m \wedge Q\ (ps',e))$
by *simp*
then obtain $ps'\ ps''\ m\ e\ e'$ **where** *a*: $(c, fst\ ps1) \Rightarrow_A m \Downarrow ps' + ps''$ **and** *orth2*[*simp*]: $ps'\ \#\#\ ps''$
and *m*: $k * snd\ ps1 = k * e + e' + m$ **and** *Q*: $Q\ (ps', e)$ **by** *blast*

from *big_step_t3_post_dom_conv*[*OF a*] **have** *dom*: $dom\ (ps' + ps'') = dom\ (fst\ ps1)$ **by** *auto*

from *add* **have** *g*: $ps = fst\ ps1 + fst\ ps2$ **and** *h*: $n = snd\ ps1 + snd\ ps2$ **by** (*auto simp add*: *plus_prod_def*)

from *orth* **have** [*simp*]: *fst ps2* ## ps' *fst ps2* ## ps''
apply (*metis dom map_convs*(*1*) *orth2 sep_disj_addD1 sep_disj_commuteI sep_disj_fun_def sep_disj_prod_def*)
by (*metis dom map_convs*(*1*) *orth orth2 sep_add_commute sep_disj_addD1 sep_disj_commuteI sep_disj_fun_def sep_disj_prod_def*)

then have *e*: ps' ## *fst ps2* **unfolding** *sep_disj_fun_def* **using** *dom* **unfolding** *domain_conv* **by** *blast*

have *3*: $(Q \wedge\!* F)$ $(ps'+fst\ ps2,\ e+snd\ ps2)$ **unfolding** *sep_conj_def*
apply(*rule exI*[**where** $x=(ps',e)$])
apply(*rule exI*[**where** $x=ps2$])
apply *safe*
subgoal using *orth* **unfolding** *sep_disj_prod_def* **apply** (*auto simp*: *sep_disj_nat_def*)
apply(*rule e*) **done**
subgoal unfolding *plus_prod_def* **apply** *auto* **done**
apply *fact* **apply** *fact* **done**

show $\exists\, ps'\ ps''\ m.\ (c, ps) \Rightarrow_A m \Downarrow ps' + ps'' \wedge ps'\ \#\#\ ps'' \wedge (\exists\, e.\ (\exists\, e'.\ k * n = k * e + e' + m) \wedge (Q \wedge\!* F)\ (ps', e))$
apply(*rule exI*[**where** $x=ps'+fst\ ps2$])
apply(*rule exI*[**where** $x=ps''$])
apply(*rule exI*[**where** $x=m$])

proof *safe*
show $(c, ps) \Rightarrow_A m \Downarrow ps' + fst\ ps2 + ps''$
apply(*rule Framer2*[*OF* _ _ *g*]) **apply** (*fact a*)
using *orth* **apply** (*auto simp*: *sep_disj_prod_def*)
by (*metis* ‹*fst ps2* ## *ps''*› ‹*fst ps2* ## *ps'*› *orth2 sep_add_assoc sep_add_commute sep_disj_commuteI*)
next
show $ps' + fst\ ps2$ ## ps''
by (*metis dom map_convs*(*1*) *orth orth2 sep_add_disjI1 sep_disj_fun_def sep_disj_prod_def*)
next
show $\exists\, e. (\exists\, e'. k * n = k * e + e' + m) \wedge (Q \wedge* F)\ (ps' + fst\ ps2, e)$
apply(*rule exI*[**where** *x*=*e*+*snd ps2*])
apply *safe*
subgoal proof(*rule exI*[**where** *x*=*e'*])
have $k * n = k * snd\ ps1 + k * snd\ ps2$ **unfolding** *h* **by** (*simp add*: *distrib_left*)
also have $\ldots = k * e + e' + m + k* snd\ ps2$ **unfolding** *m* **by** *auto*
finally show $k * n = k * (e + snd\ ps2) + e' + m$
by *algebra*
qed **apply** *fact* **done**
qed
qed
next
case (*False c Q*)
then show *?case* **by** (*auto simp*: *hoare3o_valid_def*)
next
case (*Assign2 x Q a*)
show *?case*
unfolding *hoare3o_valid_def*
apply (*rule exI*[**where** *x*=*1*], *safe*) **apply** *auto*
proof –
fix *ps n v*
assume *A*: $((\lambda s. \exists\, xa. (x \hookrightarrow xa)\ s) \wedge* (x \hookrightarrow v \longrightarrow* Q) \wedge* \$\ (Suc\ 0))\ (ps, n)$
assume *B*: $((\lambda s.\ True) \wedge* (\lambda(ps, n).\ vars\ a \subseteq dom\ ps \wedge paval\ a\ ps = v))\ (ps, n)$

from *A* **obtain** *ps1 ps2 n1 n2 v'* **where** *ps1* ## *ps2* **and** *add1*: $ps = ps1 + ps2$ **and** *n*: $n = n1 + n2$ **and**
1: $(\exists\, xaa. (x \hookrightarrow xaa)\ (ps1,n1))$

and *2*: $((x \hookrightarrow v \longrightarrow* Q) \wedge* \$ (Suc\ 0))\ (ps2,n2)$ **unfolding** *sep_conj_def*
by *fastforce*

from *2* **obtain** *ps2a ps2b n2a n2b* **where** *ps2a ## ps2b* **and** *add2*: $ps2 = ps2a + ps2b$ **and** *n2*: $n2 = n2a + n2b$
and *Q*: $(x \hookrightarrow v \longrightarrow* Q)\ (ps2a,n2a)$ **and** *ps2b*: $ps2b=0$ **and** *n2b*: $n2b=1$ **unfolding** *dollar_def sep_conj_def*
by *fastforce*

from *1* **obtain** v' **where** *n1*: $n1=0$ **and** *p*: $ps1 = ([x \mapsto v']::partstate)$

and *x*: *x : dom ps1* **by** (*auto simp*: *pointsto_def*)
from *x add1* **have** *x*: *x : dom ps*
by (*simp add*: *plus_fun_conv subset_iff*)

have *f*: $([x \mapsto v'] + ps2a)(x \mapsto v) = ps2a + [x \mapsto v]$
by (*smt* ‹$\bigwedge thesis.\ (\bigwedge v'.\ [\![n1 = 0; ps1 = [x \mapsto v']; x \in dom\ ps1]\!] \Longrightarrow thesis) \Longrightarrow thesis$› ‹*ps1 ## ps2*› *add2 disjoint_iff_not_equal dom_fun_upd domain_conv fun_upd_upd map_add_upd_left option.distinct(1) plus_fun_conv ps2b sep_add_commute sep_add_zero sep_disj_fun_def*)

let $?n' = n2a + n1$
from *n n2 n2b* **have** n': $n=1+?n'$ **by** *simp*
have Q': $Q\ (ps(x \mapsto v),\ ?n')$ **using** *Q n1* **unfolding** *sep_impl_def* **apply** *auto*
unfolding *pointsto_def* **apply** *auto*
subgoal
by (*metis* ‹*ps1 ## ps2*› ‹$ps2 = ps2a + ps2b$› ‹$ps2b = 0$› *dom_fun_upd domain_conv option.distinct(1) p sep_add_zero sep_disj_commute sep_disj_fun_def*)
subgoal unfolding *add1 p add2 ps2b*
by (*auto simp*: *f*)
done

from *B* **obtain** *ps1 ps2 n1 n2* **where** *orth*: *ps1 ## ps2* **and** *add*: $ps = ps2 + ps1$ **and** *n*: $n=n1+n2$
and *vars*: $vars\ a \subseteq dom\ ps2$ **and** *v*: $paval\ a\ ps2 = v$
unfolding *sep_conj_def* **by** (*auto simp*: *sep_add_ac*)

from *vars add* **have** *a*: $vars\ a \subseteq dom\ ps$
by (*simp add*: *plus_fun_conv subset_iff*)

from a x **have** $vars\ a \cup \{x\} \subseteq dom\ ps$ **by** $auto$

have $paval\ a\ ps = v$ **unfolding** add **apply**($subst\ paval\_extend$)
  using $orth\ vars\ v$ **by**($auto\ simp$: $sep\_disj\_commute$)

show $\exists\, ps'\ ps''\ m.\ (x ::= a,\ ps) \Rightarrow_A m \Downarrow ps' + ps'' \wedge ps'\ \#\#\ ps'' \wedge (\exists\, e.\ (\exists\, e'.\ n = e + e' + m) \wedge Q\ (ps', e))$
  apply($rule\ exI$[**where** $x{=}ps(x{\mapsto}v)$])
  apply($rule\ exI$[**where** $x{=}0$])
  apply($rule\ exI$[**where** $x{=}Suc\ 0$])
  apply $auto$
  apply($rule\ big\_step\_t\_part.Assign$)
    apply $fact+$ **apply** $simp$
  apply($rule\ exI$[**where** $x{=}?n'$])
   apply $safe$
   apply($rule\ exI$[**where** $x{=}0$]) **using** n' **apply** $simp$
   using Q' **by** $auto$

qed
next
 case $Skip$
 then show $?case$ **by** ($auto\ simp$: $hoare3o\_valid\_def\ dollar\_def$)
next
 case ($Assign4\ x\ a\ Q$)
 then show $?case$
  apply ($auto\ simp$: $dollar\_def\ sep\_conj\_def\ \ hoare3o\_valid\_def$)
   apply($rule\ exI$[**where** $x{=}1$]) **apply** $auto$
   subgoal for $ps\ b\ y$
    apply($rule\ exI$[**where** $x{=}ps(x \mapsto paval\ a\ ps)$])
    apply($rule\ exI$[**where** $x{=}0$])
    apply($rule\ exI$[**where** $x{=}Suc\ 0$]) **apply** $auto$
     apply($rule\ exI$[**where** $x{=}b$]) **by** $auto$
   done
next
 case ($Assign'\ x\ v'\ v\ Q$)
 have $\bigwedge aa.\ aa\ \#\#\ [x \mapsto v'] \Longrightarrow$
   $\neg\ aa\ \#\#\ [x \mapsto v] \Longrightarrow False$ **unfolding** $sep\_disj\_fun\_def\ domain\_def$
  apply $auto$ **by** ($smt\ Collect\_conj\_eq\ Collect\_empty\_eq$)
 have f: $\bigwedge v'.\ domain\ [x \mapsto v'] = \{x\}$ **unfolding** $domain\_conv$ **by** $auto$

 { fix ps
  assume u: $ps\ \#\#\ [x \mapsto v']$

have *2*: $[x \mapsto v'] + ps = ps + [x \mapsto v']$
$[x \mapsto v] + ps = ps + [x \mapsto v]$
subgoal apply (*subst sep_add_commute*) **using** *u* **by** (*auto simp*: *sep_add_ac*)
subgoal apply (*subst sep_add_commute*) **using** *u* **apply** (*auto simp*: *sep_add_ac*)
unfolding *sep_disj_fun_def f* **by** *auto* **done**
have $(x ::= N\ v,\ [x \mapsto v'] + ps) \Rightarrow_A Suc\ 0 \Downarrow [x \mapsto v] + ps$
apply(*rule Framer*[*OF big_step_t_part.Assign*])
apply *simp_all* **using** *u* **by** (*auto simp*: *sep_add_ac*)
then have $(x ::= N\ v,\ ps + [x \mapsto v']) \Rightarrow_A Suc\ 0 \Downarrow ps + [x \mapsto v]$
by (*simp only*: *2*)
} **note** *f2* = *this*

from *Assign'* **show** *?case*
apply (*auto simp*: *dollar_def sep_conj_def pointsto_def sep_impl_def hoare3o_valid_def*)
apply(*rule exI*[**where** *x=1*]) **apply** (*auto simp*: *sep_add_ac*)
subgoal unfolding *sep_disj_fun_def f* **by** *auto*
subgoal for *ps n*
apply(*rule exI*[**where** *x=ps+*[*x* ↦ *v*]])
apply(*rule exI*[**where** *x=0*])
apply(*rule exI*[**where** *x=Suc 0*])
apply *safe*
subgoal using *f2* **by** *auto*
subgoal **by** *auto*
subgoal by *force*
done
done
next
case (*Assign x a v*)
then show *?case* **unfolding** *hoare3o_valid_def*
apply(*rule exI*[**where** *x=1*])
apply *auto* **apply** (*auto simp*: *dollar_def*)
subgoal for *ps n*
apply (*subst* (*asm*) *separate_othogonal*) **apply** *auto*
apply(*rule exI*[**where** *x=ps*(*x*:=*Some v*)])
apply(*rule exI*[**where** *x=0*])
apply(*rule exI*[**where** *x=1*])
apply *auto*
apply (*auto simp*: *pointsto_def*) **unfolding** *sep_conj_def*
subgoal **apply**(*rule exI*[**where** *x*=((%*y*. *if y=x then None else ps y*) , *0*)])
apply(*rule exI*[**where** *x*=((%*y*. *if y = x then Some* (*paval a ps*) *else*

None),*0*)])
  apply (*auto simp*: *sep_disj_prod_def sep_disj_fun_def plus_fun_def*)
  apply (*smt domIff domain_conv*)
  apply (*metis domI insertE option.simps*(*3*))
  using *domIff* **by** *fastforce*
 done
 done
next
 case (*If P b* c_1 *Q* c_2)
 from *If*(*3*)[*unfolded hoare3o_valid_def*]
 obtain *k1* **where** *T*: $\bigwedge ps\ n.\ P\ (ps,\ n) \Longrightarrow vars\ b \subseteq dom\ ps \Longrightarrow pbval\ b\ ps$
 $\Longrightarrow (\exists ps'\ ps''\ m\ e\ e'.\ (c_1,\ ps) \Rightarrow_A m \Downarrow ps' + ps'' \wedge ps'\ \#\#\ ps'' \wedge k1 * n = k1 * e + e' + m \wedge Q\ (ps',\ e))$
 and *k1*: $k1 > 0$ **by** *force*
 from *If*(*4*)[*unfolded hoare3o_valid_def*]
 obtain *k2* **where** *F*: $\bigwedge ps\ n.\ P\ (ps,\ n) \Longrightarrow vars\ b \subseteq dom\ ps \Longrightarrow \neg\ pbval\ b\ ps$
 $\Longrightarrow (\exists ps'\ ps''\ m\ e\ e'.\ (c_2,\ ps) \Rightarrow_A m \Downarrow ps' + ps'' \wedge ps'\ \#\#\ ps'' \wedge k2 * n = k2 * e + e' + m \wedge Q\ (ps',\ e))$
 and *k2*: $k2 > 0$ **by** *force*

 show *?case* **unfolding** *hoare3o_valid_def* **apply** *auto* **apply** (*auto simp*: *dollar_def*)
 apply(*rule exI*[**where** $x{=}k1 * k2$]) **using** *k1 k2* **apply** *auto*
 proof (*goal_cases*)
 case (*1 ps n*)
 then obtain n' **where** *P*: $P\ (ps,\ n')$ **and** *dom*: $vars\ b \subseteq dom\ ps$ **and** *Suc*: $n = Suc\ n'$ **unfolding** *sep_conj_def*
 by *force*
 show *?case*
 proof(*cases pbval b ps*)
 case *True*
 with *T*[*OF P dom*] **obtain** $ps'\ ps''\ m\ e\ e'$ **where** *d*: $(c_1,\ ps) \Rightarrow_A m \Downarrow ps' + ps''$
 and *orth*: $ps'\ \#\#\ ps''$ **and** *m1*: $k1 * n' = k1 * e + e' + m$ **and** *Q*: $Q\ (ps',\ e)$
 by *blast*
 from *big_step_t3_post_dom_conv*[*OF d*] **have** *klong*: $dom\ (ps' + ps'') = dom\ ps$ **.**
 from *k1* **obtain** $k1'$ **where** $k1'$: $k1 = Suc\ k1'$
 using *gr0_implies_Suc* **by** *blast*
 from *k2* **obtain** $k2'$ **where** $k2'$: $k2 = Suc\ k2'$
 using *gr0_implies_Suc* **by** *blast*

let *?e1* = (*k2'* ∗ (*e'* + *m* + *k1*) + *e'* + *k1'*)
show *?thesis*
apply(*rule exI*[**where** *x*=*ps'*])
apply(*rule exI*[**where** *x*=*ps''*]) **apply**(*rule exI*[**where** *x*=*m*+*1*])
apply *safe*
apply(*rule big_step_t_part.IfTrue*)
apply (*rule dom*)
apply *fact*
apply (*rule True*)
apply (*rule d*)
apply *simp*
apply *fact*
subgoal apply(*rule exI*[**where** *x*=*e*])
apply *safe*
subgoal proof (*rule exI*[**where** *x*=*?e1*])
have *k1* ∗ *k2* ∗ *n* = *k2* ∗ (*k1*∗*n*) **by** *auto*
also have ... = *k2* ∗ (*k1*∗*n'* + *k1*) **unfolding** *Suc* **by** *auto*
also have ... = *k2* ∗ (*k1* ∗ *e* + *e'* + *m* + *k1*) **unfolding** *m1* **by** *auto*
also have ... = *k1* ∗ *k2* ∗ *e* + *k2*∗ (*e'* + *m* + *k1*) **by** *algebra*
also have ... = *k1* ∗ *k2* ∗ *e* + *k2'* ∗ (*e'* + *m* + *k1*) + (*e'* + *m* + *k1*) **unfolding** *k2'*
by *simp*
also have ... = *k1* ∗ *k2* ∗ *e* + *k2'* ∗ (*e'* + *m* + *k1*) + (*e'* + *k1'* + *m* + *1*) **unfolding** *k1'* **by** *simp*
also have ... = *k1* ∗*k2*∗*e* + (*k2'* ∗ (*e'* + *m* + *k1*) + *e'* + *k1'*) + (*m*+*1*) **by** *algebra*
finally show *k1* ∗ *k2* ∗ *n* = *k1* ∗ *k2* ∗ *e* + *?e1* + (*m* + *1*) **.**
qed **using** *Q* **by** *force*
done
next
case *False*
with *F*[*OF P dom*] **obtain** *ps' ps'' m e e'* **where** *d*: (c_2, *ps*) $\Rightarrow_A$ *m* ⇓ *ps'* + *ps''*
and *orth*: *ps'* ## *ps''* **and** *m2*: *k2* ∗ *n'* = *k2* ∗ *e* + *e'* + *m* **and** *Q*: *Q* (*ps'*, *e*)
by *blast*
from *big_step_t3_post_dom_conv*[*OF d*] **have** *klong*: *dom* (*ps'* + *ps''*) = *dom ps* **.**
from *k1* **obtain** *k1'* **where** *k1'*: *k1* = *Suc k1'*
using *gr0_implies_Suc* **by** *blast*
from *k2* **obtain** *k2'* **where** *k2'*: *k2* = *Suc k2'*
using *gr0_implies_Suc* **by** *blast*
let *?e2* = (*k1'* ∗ (*e'* + *m* + *k2*) + *e'* + *k2'*)

show *?thesis*
apply(*rule exI*[**where** x=*ps'*])
apply(*rule exI*[**where** x=*ps''*]) **apply**(*rule exI*[**where** x=*m+1*])
apply *safe*
apply(*rule big_step_t_part.IfFalse*)
apply (*rule dom*)
apply *fact*
apply (*rule False*)
apply (*rule d*)
apply *simp*
apply *fact*
subgoal apply(*rule exI*[**where** x=*e*])
apply *safe*
subgoal proof (*rule exI*[**where** x=*?e2*])
have $k1 * k2 * n = k1 * (k2{*}n)$ **by** *auto*
also have $\ldots = k1 * (k2{*}n' + k2)$ **unfolding** *Suc* **by** *auto*
also have $\ldots = k1 * (k2 * e + e' + m + k2)$ **unfolding** *m2* **by** *auto*
also have $\ldots = k1 * k2 * e + k1{*}\,(e' + m + k2)$ **by** *algebra*
also have $\ldots = k1 * k2 * e + k1' * (e' + m + k2) + (e' + m + k2)$ **unfolding** *k1'*
by *simp*
also have $\ldots = k1 * k2 * e + k1' * (e' + m + k2) + (e' + k2' + m + 1)$ **unfolding** *k2'* **by** *simp*
also have $\ldots = k1 * k2{*}e + (k1' * (e' + m + k2) + e' + k2') + (m{+}1)$ **by** *algebra*
finally show $k1 * k2 * n = k1 * k2 * e + ?e2 + (m + 1)$ **.**
qed **using** *Q* **by** *force*
done
qed
qed
next
case (*Seq* P C_1 Q C_2 R)

from *Seq*(*3*)[*unfolded hoare3o_valid_def*] **obtain** *k1* **where**
1: $(\forall ps\ n.\ P\ (ps, n) \longrightarrow (\exists ps'\ ps''\ m\ e\ e'.\ (C_1, ps) \Rightarrow_A m \Downarrow ps' + ps'' \wedge ps'\ \#\#\ ps'' \wedge k1 * n = k1 * e + e' + m \wedge Q\ (ps', e)))$
and *k10*: $k1 > 0$ **by** *blast*
from *Seq*(*4*)[*unfolded hoare3o_valid_def*] **obtain** *k2* **where**
2: $(\forall ps\ n.\ Q\ (ps, n) \longrightarrow (\exists ps'\ ps''\ m\ e\ e'.\ (C_2, ps) \Rightarrow_A m \Downarrow ps' + ps'' \wedge ps'\ \#\#\ ps'' \wedge k2 * n = k2 * e + e' + m \wedge R\ (ps', e)))$
and *k20*: $k2 > 0$ **by** *blast*

from *k10* **obtain** *k1'* **where** *k1'*: $k1 = Suc\ k1'$

using *gr0_implies_Suc* **by** *blast*
from *k20* **obtain** *k2'* **where** *k2'*: *k2 = Suc k2'*
using *gr0_implies_Suc* **by** *blast*

show *?case* **unfolding** *hoare3o_valid_def*
apply(*rule exI*[**where** *x=k2∗k1*])
proof *safe*
fix *ps n*
assume *P* (*ps*, *n*)

with *1* **obtain** *ps1' ps1'' m1 e1 e1'* **where** *C1*: (C_1, *ps*) $\Rightarrow_A$ *m1* ⇓ *ps1' + ps1''* **and** *orth*: *ps1' ## ps1''*
and *m1*: *k1 ∗ n = k1 ∗ e1 + e1' + m1* **and** *Q*: *Q* (*ps1'*, *e1*) **by** *blast*

from *Q* **and** *2* **obtain** *ps2' ps2'' m2 e2 e2'* **where** *C2*: (C_2, *ps1'*) $\Rightarrow_A$ *m2* ⇓ *ps2' + ps2''* **and** *orth2*: *ps2' ## ps2''*
and *m2*: *k2 ∗ e1 = k2 ∗ e2 + e2' + m2* **and** *R*: *R* (*ps2'*, *e2*) **by** *blast*

let *?ee* = (*k1 ∗e2' + k2∗e1' +k2'∗m1+ k1'∗m2*)

show ∃ *ps' ps'' m e e'*. (C_1;; C_2, *ps*) $\Rightarrow_A$ *m* ⇓ *ps' + ps''* ∧ *ps' ## ps''* ∧ *k2 ∗ k1 ∗ n = k2 ∗ k1 ∗ e + e' + m* ∧ *R* (*ps'*, *e*)
apply(*rule exI*[**where** *x=ps2'*])
apply(*rule exI*[**where** *x=ps2'' + ps1''*])
apply(*rule exI*[**where** *x=m1+m2*])
apply(*rule exI*[**where** *x=e2*])
apply(*rule exI*[**where** *x=?ee*])
proof *safe*
have *C2'*: (C_2, *ps1' + ps1''*) $\Rightarrow_A$ *m2* ⇓ *ps2' +* (*ps2'' + ps1''*)
apply(*rule Framer2*[*OF C2*, *of ps1''*]) **apply** *fact* **apply** *simp*
using *sep_add_assoc*
by (*metis C2 big_step_t3_post_dom_conv map_convs*(*1*) *orth orth2 sep_add_commute sep_disj_addD1 sep_disj_commuteI sep_disj_fun_def*)
show (C_1;; C_2, *ps*) $\Rightarrow_A$ *m1 + m2* ⇓ *ps2' +* (*ps2'' + ps1''*)
using *C1 C2'* **by** *auto*
next
show *ps2' ## ps2'' + ps1''*
by (*metis C2 big_step_t3_post_dom_conv map_convs*(*1*) *orth orth2 sep_disj_addI3 sep_disj_fun_def*)
next
have *k2 ∗ k1 ∗ n = k2 ∗* (*k1 ∗ n*) **by** *auto*
also have ... = *k2 ∗* (*k1 ∗ e1 + e1' + m1*) **using** *m1* **by** *auto*

also have ... = $k1 * k2 * e1 + k2 * (e1' + m1)$ **by** *algebra*
also have ... = $k1 * (k2 * e2 + e2' + m2) + k2 * (e1' + m1)$ **using** $m2$ **by** *auto*
also have ... = $k2 * k1 * e2 + (k1 *e2' + k2*e1' +k2*m1+ k1*m2)$ **by** *algebra*
also have ... = $k2 * k1 * e2 + (k1 *e2' + k2*e1' +k2'*m1+m1+ k1'*m2+m2)$ **unfolding** $k1'$ $k2'$ **by** *auto*
also have ... = $k2 * k1 * e2 + (k1 *e2' + k2*e1' +k2'*m1+ k1'*m2)+(m1+m2)$ **by** *auto*
finally show $k2 * k1 * n = k2 * k1 * e2 + ?ee + (m1 + m2)$ **.**
qed *fact*
qed (*simp add*: *k10 k20*)
next
case (*While P b C*)

{
assume $\exists k>0. \forall ps\ n.$ (*case* (ps, n) *of* $(s, n) \Rightarrow P\ (s, n) \wedge$ *lmaps_to_axpr* b *True* $s) \longrightarrow$
$(\exists ps'\ ps''\ m\ e\ e'.\ (C, ps) \Rightarrow_A m \Downarrow ps' + ps'' \wedge ps'\ \#\#\ ps'' \wedge k * n = k * e + e' + m \wedge ((\lambda(s, n).\ P\ (s, n) \wedge vars\ b \subseteq dom\ s) \wedge*\ \$\ 1)\ (ps', e))$
then obtain k **where** *While2*: $\forall ps\ n.$ (*case* (ps, n) *of* $(s, n) \Rightarrow P\ (s, n) \wedge$ *lmaps_to_axpr* b *True* $s) \longrightarrow (\exists ps'\ ps''\ m\ e\ e'.\ (C, ps) \Rightarrow_A m \Downarrow ps' + ps'' \wedge ps'\ \#\#\ ps'' \wedge k * n = k * e + e' + m \wedge ((\lambda(s, n).\ P\ (s, n) \wedge vars\ b \subseteq dom\ s) \wedge*\ \$\ 1)\ (ps', e))$ **and** k: $k>0$ **by** *blast*

from k **obtain** k' **where** k': $k = Suc\ k'$
using *gr0_implies_Suc* **by** *blast*

have $\exists k>0. \forall ps\ n.\ ((\lambda(s, n).\ P\ (s, n) \wedge vars\ b \subseteq dom\ s) \wedge*\ \$\ 1)\ (ps, n) \longrightarrow$
$(\exists ps'\ ps''\ m\ e\ e'.$
$(WHILE\ b\ DO\ C, ps) \Rightarrow_A m \Downarrow ps' + ps'' \wedge$
$ps'\ \#\#\ ps'' \wedge k * n = k * e + e' + m \wedge$ (*case* (ps', e) *of* $(s, n) \Rightarrow P\ (s, n) \wedge$ *lmaps_to_axpr* b *False* $s))$ **proof** (*rule exI*[**where** $x=k$], *safe*, *goal_cases*)
case (*2 ps n*)
from *2* **show** *?case*
proof(*induct n arbitrary*: *ps rule*: *less_induct*)
case (*less x ps3*)

show *?case*
proof(*cases pbval b ps3*)
case *True*

from *less(2)* **obtain** x' **where** *P*: P $(ps3,\ x')$ **and** *dom*: *vars b* $\subseteq$ *dom ps3* **and** *Suc*: $x = Suc\ x'$ **unfolding** *sep_conj_def dollar_def* **by** *auto*

from *P dom True* **have**
g: $((\lambda(s,\ n).\ P\ (s,\ n) \wedge lmaps\_to\_axpr\ b\ True\ s))\ (ps3,\ x')$
unfolding *dollar_def* **by** *auto*

from *While2 g* **obtain** $ps3'\ ps3''\ m\ e\ e'$ **where** *C*: $(C,\ ps3) \Rightarrow_A m \Downarrow ps3' + ps3''$ **and** *orth*: $ps3'$ ## $ps3''$
and *x*: $k * x' = k * e + e' + m$ **and** P': $((\lambda(s,\ n).\ P\ (s,\ n) \wedge vars\ b \subseteq dom\ s) \wedge\!* \$\ 1)\ (ps3',\ e)$ **by** *blast*
then obtain x''' **where** P'': $P\ (ps3',\ x''')$ **and** *domb*: *vars b* $\subseteq$ *dom* $ps3'$ **and** Suc'': $e = Suc\ x'''$
unfolding *sep_conj_def dollar_def* **by** *auto*

from *C big_step_t3_post_dom_conv* **have** *dom ps3* $=$ *dom* $(ps3' + ps3'')$ **by** *simp*
with *dom* **have** dom': *vars b* $\subseteq$ *dom* $(ps3' + ps3'')$ **by** *auto*

from *C big_step_t3_gt0* **have** *gt0*: $m > 0$ **by** *auto*

have $e < x$ **using** *x Suc gt0*
by (*metis k le_add1 le_less_trans less_SucI less_add_same_cancel1 nat_mult_less_cancel1*)

have $\exists\, ps'\ ps''\ m\ e2\ e2'.\ (WHILE\ b\ DO\ C,\ ps3') \Rightarrow_A m \Downarrow ps' + ps'' \wedge ps'$ ## $ps'' \wedge k * e = k * e2 + e2' + m \wedge P\ (ps',\ e2) \wedge lmaps\_to\_axpr\ b\ False\ ps'$
apply(*rule less(1)*)
apply *fact* **by** *fact*

then obtain $ps4'\ ps4''\ mt\ et\ et'$ **where** *w*: $((WHILE\ b\ DO\ C,\ ps3') \Rightarrow_A mt \Downarrow ps4' + ps4'')$
and *ortho*: $ps4'$ ## $ps4''$ **and** m'': $k * e = k * et + et' + mt$
and P'': $P\ (ps4',\ et)$ **and** dom'': *vars b* $\subseteq$ *dom* $ps4'$ **and** b'': $\neg$ *pbval b* $ps4'$ **by** *auto*

have $ps4''$ ## $ps3''$ **and** $ps4'$ ## $ps3''$ **by** (*metis big_step_t3_post_dom_conv domain_conv orth ortho sep_add_disjD sep_disj_fun_def w*)+

show *?thesis*
apply(*rule exI*[**where** $x=ps4'$])
apply(*rule exI*[**where** $x=(ps4'' + ps3'')$])
apply(*rule exI*[**where** $x=1 + m + mt$])

apply(*rule exI*[**where** x=*et*])
apply(*rule exI*[**where** x= et' + k' + e'])
proof (*safe*)
have (*WHILE b DO C*, $ps3'$ + $ps3''$) $\Rightarrow_A$ *mt* $\Downarrow$ $ps4'$ + ($ps4''$ + $ps3''$)
apply(*rule Framer2*[*OF w*, *of* $ps3''$]) **apply** *fact*
apply *simp*
apply(*rule sep_add_assoc*[*symmetric*])
by *fact*+
show (*WHILE b DO C*, *ps3*) $\Rightarrow_A$ *1* + *m* + *mt* $\Downarrow$ $ps4'$ + ($ps4''$ + $ps3''$)
apply(*rule WhileTrue*) **apply** *fact* **apply** *fact* **apply** (*fact C*) **apply** *fact* **by** *auto*
next
show $ps4'$ ## $ps4''$ + $ps3''$
by (*metis big_step_t3_post_dom_conv domain_conv orth ortho sep_disj_addI3 sep_disj_fun_def w*)
next
have $k * x = k * x' + k$ **unfolding** *Suc* **by** *auto*
also have $\ldots = k * e + e' + m + k$ **unfolding** x **by** *simp*
also have $\ldots = k * et + et' + mt + e' + m + k$ **using** m'' **by** *simp*
also have $\ldots = k * et + et' + mt + e' + m + 1 + k'$ **using** k' **by** *simp*
also have $\ldots = k * et + (et' + k' + e') + (1 + m + mt)$ **using** k' **by** *simp*
finally show $k * x = k * et + (et' + k' + e') + (1 + m + mt)$ **by** *simp*
next
show P ($ps4'$, *et*) **by** *fact*
next
show *lmaps_to_axpr b False* $ps4'$ **apply** *simp* **using** dom'' b'' **..**
qed
next
case *False*
from *less*(*2*) **obtain** x' **where** P: P (*ps3*, x') **and** *dom*: *vars b* $\subseteq$ *dom ps3* **and** *Suc*: x = *Suc* x' **unfolding** *dollar_def sep_conj_def*
by *force*
show *?thesis*
apply(*rule exI*[**where** x=*ps3*])
apply(*rule exI*[**where** x=*0*])
apply(*rule exI*[**where** x=*Suc 0*])
apply(*rule exI*[**where** x=x'])
apply(*rule exI*[**where** x=k']) **apply** *safe*

apply *simp* **apply** (*rule big_step_t_part.WhileFalse*)
subgoal using *dom* **by** *simp*
apply *fact* **apply** *simp*
using *Suc k k'* **apply** *simp*
using *P Suc* **apply** *simp*
using *dom* **apply** *auto*
using *False* **apply** *auto* **done**
qed

qed

qed (*fact*)

} **with** *While*(*2*)
show *?case* **unfolding** *hoare3o_valid_def* **by** *simp*
next
case (*Assign''' x ds v*)
then show *?case*
unfolding *hoare3o_valid_def* **apply** *auto*
apply(*rule exI*[**where** *x=1*]) **apply** *auto*
subgoal for *ps n* **apply**(*rule exI*[**where** *x=ps*($x \mapsto v$)]) **apply**(*rule exI*[**where** *x=0*])
apply(*rule exI*[**where** *x=Suc 0*])
apply *safe*
apply(*rule big_step_t_part.Assign*)
apply (*auto*)
subgoal apply(*subst* (*asm*) *separate_othogonal_commuted'*) **by**(*auto simp*: *dollar_def pointsto_def*)
subgoal apply(*subst* (*asm*) *separate_othogonal_commuted'*) **by**(*auto simp*: *dollar_def pointsto_def*)
done
done
next
case (*Assign'''' P a v x Q'*)
from *Assign''''*(*3*)[*unfolded hoare3o_valid_def*] **obtain** *k* **where** *k*: *k>0* **and**
A: $\forall ps\ n.\ P\ (ps, n) \longrightarrow (\exists ps'\ ps''\ m\ e\ e'.\ (x ::= N\ v, ps) \Rightarrow_A m \Downarrow ps' + ps'' \wedge ps' \#\#\ ps'' \wedge k * n = k * e + e' + m \wedge Q'\ (ps', e))$
by *auto*
show *?case*
unfolding *hoare3o_valid_def* **apply** *auto*
apply(*rule exI*[**where** *x=k*]) **using** *k* **apply** *auto*
proof (*goal_cases*)

case (*1 ps n*)
with *A* **obtain** *ps′ ps″ m e e′*
where (*x* ::= *N v*, *ps*) $\Rightarrow_A$ *m* ⇓ *ps′* + *ps″* **and** *orth*: *ps′* ## *ps″* **and** *m*: *k* ∗ *n* = *k* ∗ *e* + *e′* + *m* **and** *Q*: *Q′* (*ps′*, *e*) **by** *metis*
from *1*(*2*) *Assign''''*(*1*)[*unfolded symeval_def*] **have** *paval′ a ps* = *Some v* **by** *auto*
show *?case* **apply**(*rule exI*[**where** *x=ps'*]) **apply**(*rule exI*[**where** *x=ps''*]) **apply**(*rule exI*[**where** *x=m*])
apply *safe*
apply(*rule avalDirekt3_correct*) **apply** *fact+*
apply(*rule exI*[**where** *x=e*]) **apply** *safe*
apply(*rule exI*[**where** *x=e'*]) **apply** *fact*
apply *fact* **done**
qed
next
case (*pureI P Q c R*)
show *?case*
proof (*cases P*)
case *True*
with *pureI*(*2*)[*unfolded hoare3o_valid_def*] **obtain** *k* **where** *k*: *k>0* **and**
thing: ∀ *ps n*. *Q* (*ps*, *n*) ⟶ (∃ *ps′ ps″ m e e′*. (*c*, *ps*) $\Rightarrow_A$ *m* ⇓ *ps′* + *ps″* ∧ *ps′* ## *ps″* ∧ *k* ∗ *n* = *k* ∗ *e* + *e′* + *m* ∧ *R* (*ps′*, *e*)) **by** *auto*

show *?thesis* **unfolding** *hoare3o_valid_def* **apply**(*rule exI*[**where** *x=k*])
apply *safe* **apply** *fact*
using *thing* **by** *fastforce*
next
case *False*
show *?thesis* **unfolding** *hoare3o_valid_def* **apply**(*rule exI*[**where** *x=1*])
using *False* **by** *auto*
qed
next
case (*normalize P m C Q n*)
then show *?case* **unfolding** *hoare3o_valid_def*
apply(*safe*) **subgoal for** *k* **apply**(*rule exI*[**where** *x=k*]) **apply** *safe*
proof (*goal_cases*)
case (*1 ps N*)
have *Q2*: *P* (*ps*, *N* − (*m* − *n*)) **apply**(*rule stardiff*) **by** *fact*
have *mn*: *m* − *n* ≤ *N* **apply**(*rule stardiff*(*2*)) **by** *fact*
have *P*: (*P* ∧∗ $ *m*) (*ps*, *N* − (*m* − *n*) + *m*) **unfolding** *sep_conj_def dollar_def*

apply(*rule exI*[**where** $x=(ps,N - (m - n))$]) **apply**(*rule exI*[**where** $x=(0,m)$])
apply(*auto simp*: *sep_disj_prod_def sep_disj_nat_def*) **by** *fact*
have *N*: $N - (m - n) + m = N + n$ **using** *normalize*(*2*)
using *mn* **by** *auto*
from *P N* **have** *P'*: ($P \wedge * \$ m$) (*ps*, $N + n$) **by** *auto*

from *P' 1*(*4*) **obtain** *ps' ps'' m' e e'* **where** (C, *ps*) $\Rightarrow_A m' \Downarrow ps' + ps''$ **and** *orth*: *ps'* ## *ps''*
and *m'*: $k * (N + n) = k * e + e' + m'$ **and** *Q*: ($Q \wedge * \$ n$) (*ps'*, *e*) **by** *blast*

have *Q2*: *Q* (*ps'*, $e - n$) **apply**(*rule stardiff*) **by** *fact*

have *en*: $e \geq n$ **using** *Q*
using *stardiff*(*2*) **by** *blast*

show *?case*
apply(*rule exI*[**where** *x*=*ps'*])
apply(*rule exI*[**where** *x*=*ps''*]) **apply**(*rule exI*[**where** *x*=*m'*])
apply(*rule exI*[**where** $x=e - n$])
apply(*rule exI*[**where** *x*=*e'*])
proof (*safe*)
show (C, *ps*) $\Rightarrow_A m' \Downarrow ps' + ps''$ **by** *fact*
next
show *ps'* ## *ps''* **by** *fact*
next
have $k * N = k * ((N + n) - n)$ **by** *auto*
also have $\ldots = k*(N + n) - k*n$ **using** *right_diff_distrib'* **by** *blast*
also have $\ldots = (k * e + e' + m') - k*n$ **using** *m'* **by** *auto*
also have $\ldots = k * e - k*n + e' + m'$ **using** *en*
by (*metis Nat.add_diff_assoc2 ab_semigroup_add_class.add_ac*(*1*) *distrib_left le_add1 le_add_diff_inverse*)
also have $\ldots = k * (e-n) + e' + m'$ **by** (*simp add*: *diff_mult_distrib2*)

finally show $k * N = k * (e - n) + e' + m'$ **.**
next
show *Q* (*ps'*, $e - n$) **by** *fact*
qed
qed
done
qed

thm *hoareT_sound2_part E_extendsR*

lemma *hoareT_sound2_partR*: $\vdash_{3a}$ $\{P\}$ *c* $\{$ $Q\}$ $\Longrightarrow$ $\models_{3'}$ $\{P\}$ *c* $\{Q\}$
using *hoareT_sound2_part E_extendsR* **by** *blast*

9.3.1 nice example

lemma *Frame_R*: **assumes** $\vdash_{3'}$ $\{$ P $\}$ C $\{$ Q $\}$ *Frame* P' P F
shows $\vdash_{3'}$ $\{$ P' $\}$ C $\{$ Q $**$ F $\}$
apply(*rule conseq*[**where** *k=1*]) **apply**(*rule Frame*) **apply**(*rule assms*(*1*))
using *assms*(*2*) **unfolding** *Frame_def* **by** *auto*

lemma *strengthen_post*: **assumes** $\vdash_{3'}$ $\{P\}c\{Q\}$ $\bigwedge s.\ Q\ s \Longrightarrow Q'\ s$
shows $\vdash_{3'}$ $\{P\}c\{$ $Q'\}$
apply(*rule conseq*[**where** *k=1*])
apply (*rule assms*(*1*))
apply *simp* **apply** *simp* **apply** *fact* **apply** *simp* **done**

lemmas *strongAssign* = *Assign''''*[*OF _ strengthen_post*, *OF _ Frame_R*, *OF _ Assign'''*]

lemma *weakenpre*: $[\![$ $\vdash_{3'}$ $\{P\}c\{Q\}$; $\bigwedge s.\ P'\ s \Longrightarrow P\ s$ $]\!]$ $\Longrightarrow$
$\vdash_{3'}$ $\{P'\}c\{$ $Q\}$
using *conseq*[**where** *k=1*] **by** *auto*

lemma *weakenpreR*: $[\![$ $\vdash_{3a}$ $\{P\}c\{Q\}$; $\bigwedge s.\ P'\ s \Longrightarrow P\ s$ $]\!]$ $\Longrightarrow$
$\vdash_{3a}$ $\{P'\}c\{$ $Q\}$
using *hoare3a.conseqS* **by** *auto*

9.4 Completeness

definition *wp3'* :: *com* $\Rightarrow$ *assn2* $\Rightarrow$ *assn2* ($\langle wp_{3'}{}' \rangle$) **where**
$wp_{3'}\ c\ Q\ =\ (\lambda(s,n).\ \exists\, t\ m.\ n \geq m \wedge (c,s) \Rightarrow_A m \Downarrow t \wedge Q\ (t,n-m))$

lemma *wp3Skip*[*simp*]: $wp_{3'}$ *SKIP* $Q = (Q\ **\ \$1)$
apply (*auto intro*!: *ext simp*: *wp3'_def*)
unfolding *sep_conj_def dollar_def sep_disj_prod_def sep_disj_nat_def*
apply *auto* **apply** *force*
subgoal for *t n* **apply**(*rule exI*[**where** *x=t*]) **apply**(*rule exI*[**where** *x=Suc 0*])

using *big_step_t_part.Skip* **by** *auto*
done

lemma *wp3Assign*[*simp*]: $wp_{3'}$ (x ::= e) Q = ((λ(*ps*,*t*). *vars e* $\cup$ {x} $\subseteq$ *dom ps* $\wedge$ Q (*ps*($x \mapsto$ *paval e ps*),*t*)) ** \$*1*)
apply (*auto intro*!: *ext simp*: *wp3'_def*)
unfolding *sep_conj_def* **apply** (*auto simp*: *sep_disj_prod_def sep_disj_nat_def dollar_def*) **apply** *force*
by *fastforce*

lemma *wpt_Seq*[*simp*]: $wp_{3'}$ (c_1;;c_2) Q = $wp_{3'}$ c_1 ($wp_{3'}$ c_2 Q)
apply (*auto simp*: *wp3'_def fun_eq_iff*)
subgoal for *a b t m1 s2 m2*
apply(*rule exI*[**where** *x*=*s2*])
apply(*rule exI*[**where** *x*=*m1*])
apply *simp*
apply(*rule exI*[**where** *x*=*t*])
apply(*rule exI*[**where** *x*=*m2*])
apply *simp* **done**
subgoal for *s m t' m1 t m2*
apply(*rule exI*[**where** *x*=*t*])
apply(*rule exI*[**where** *x*=*m1*+*m2*])
apply (*auto simp*: *big_step_t_part.Seq*) **done**
done

lemma *wp3If*[*simp*]:
$wp_{3'}$ (*IF b THEN* c_1 *ELSE* c_2) Q = ((λ(*ps*,*t*). *vars b* $\subseteq$ *dom ps* $\wedge$ $wp_{3'}$ (*if pbval b ps then* c_1 *else* c_2) Q (*ps*,*t*)) ** \$*1*)
apply (*auto simp*: *wp3'_def fun_eq_iff*)
unfolding *sep_conj_def* **apply** (*auto simp*: *sep_disj_prod_def sep_disj_nat_def dollar_def*)
subgoal for *a ba t x* **apply**(*rule exI*[**where** *x*=*ba* − *1*]) **apply** *auto*
apply(*rule exI*[**where** *x*=*t*]) **apply**(*rule exI*[**where** *x*=*x*]) **apply** *auto*
done
subgoal for *a ba t x* **apply**(*rule exI*[**where** *x*=*ba* − *1*]) **apply** *auto*
apply(*rule exI*[**where** *x*=*t*]) **apply**(*rule exI*[**where** *x*=*x*]) **apply** *auto*
done
subgoal for *a ba t m*
apply(*rule exI*[**where** *x*=*t*]) **apply**(*rule exI*[**where** *x*=*Suc m*]) **apply** *auto*
apply(*cases pbval b a*)
subgoal apply *simp* **apply**(*subst big_step_t_part.IfTrue*) **using** *big_step_t3_post_dom_conv* **by** *auto*
subgoal apply *simp* **apply**(*subst big_step_t_part.IfFalse*) **using** *big_step_t3_post_dom_conv*

by *auto*
  done
  done

lemma *sFTrue*: **assumes** *pbval b ps vars b* $\subseteq$ *dom ps*
  shows $wp_{3'}$ (*WHILE b DO c*) *Q* (*ps*, *n*) = ((λ(*ps*, *n*). *vars b* $\subseteq$ *dom ps* $\wedge$ (*if pbval b ps then* $wp_{3'}$ *c* ($wp_{3'}$ (*WHILE b DO c*) *Q*) (*ps*, *n*) *else Q* (*ps*, *n*))) $\wedge*$ \$ *1*) (*ps*, *n*)
    (**is** *?wp* = (*?I* $\wedge*$ \$ *1*) _)
proof
  assume $wp_{3'}$ (*WHILE b DO c*) *Q* (*ps*, *n*)
  from *this*[*unfolded wp3'_def*] **obtain** *ps''* *tt* **where** *tn*: *tt* $\leq$ *n* **and** *w1*: (*WHILE b DO c*, *ps*) $\Rightarrow_A$ *tt* $\Downarrow$ *ps''* **and** *Q*: *Q* (*ps''*, *n* $-$ *tt*) **by** *blast*
  with *assms* **obtain** *t t' ps'* **where** *w2*: (*WHILE b DO c*, *ps'*) $\Rightarrow_A$ *t'* $\Downarrow$ *ps''* **and** *c*: (*c*, *ps*) $\Rightarrow_A$ *t* $\Downarrow$ *ps'* **and** *tt*: *tt=1+t+t'* **by** *auto*

  from *tn* **obtain** *k* **where** *n*: *n=tt+k*
    using *le_Suc_ex* **by** *blast*

  from *assms* **show** (*?I* $\wedge*$ \$ *1*) (*ps*,*n*)
    unfolding *sep_conj_def dollar_def wp3'_def* **apply** *auto*
    apply(*rule exI*[**where** *x=t+t'+k*])
      apply *safe* **subgoal using** *n tt* **by** *auto*
      apply(*rule exI*[**where** *x=ps'*])
    apply(*rule exI*[**where** *x=t*])
    using *c* **apply** *auto*
      apply(*rule exI*[**where** *x=ps''*])
    apply(*rule exI*[**where** *x=t'*])
    using *w2 Q n* **by** *auto*
next
  assume (*?I* $\wedge*$ \$ *1*) (*ps*,*n*)
  with *assms* **have** *Q*: $wp_{3'}$ *c* ($wp_{3'}$ (*WHILE b DO c*) *Q*) (*ps*, *n*$-$*1*) **and** *n*: *n*$\geq$*1* **unfolding** *dollar_def sep_conj_def* **by** *auto*
  then obtain *t ps' t' ps''* **where** *t*: *t* $\leq$ *n* $-$ *1*
      and *c*: (*c*, *ps*) $\Rightarrow_A$ *t* $\Downarrow$ *ps'* **and** *t'*: *t'* $\leq$ (*n*$-$*1*) $-$ *t* **and** *w*: (*WHILE b DO c*, *ps'*) $\Rightarrow_A$ *t'* $\Downarrow$ *ps''*
      and *Q*: *Q* (*ps''*, ((*n*$-$*1*) $-$ *t*) $-$ *t'*)
    unfolding *wp3'_def* **by** *auto*
  show *?wp* **unfolding** *wp3'_def*
      apply *simp* **apply**(*rule exI*[**where** *x=ps''*]) **apply**(*rule exI*[**where** *x=1+t+t'*])
    apply *safe*
    subgoal using *t t' n* **by** *simp*

subgoal using *c w assms* **by** *auto*
subgoal using *Q t t′ n* **by** *simp*
done
qed

lemma *sFFalse*: **assumes** ~ *pbval b ps vars b* ⊆ *dom ps*
shows $wp_{3'}$ (*WHILE b DO c*) *Q* (*ps*, *n*) = ((λ(*ps*, *n*). *vars b* ⊆ *dom ps* ∧ (*if pbval b ps then* $wp_{3'}$ *c* ($wp_{3'}$ (*WHILE b DO c*) *Q*) (*ps*, *n*) *else Q* (*ps*, *n*))) ∧∗ $ *1*) (*ps*, *n*)
(**is** *?wp* = (*?I* ∧∗ $ *1*) _)
proof
assume $wp_{3'}$ (*WHILE b DO c*) *Q* (*ps*, *n*)
from *this*[*unfolded wp3′_def*] **obtain** *ps′ t* **where** *tn*: *t* ≤ *n* **and** *w1*: (*WHILE b DO c*, *ps*) $\Rightarrow_A$ *t* ⇓ *ps′* **and** *Q*: *Q* (*ps′*, *n* − *t*) **by** *blast*
from *assms* **have** *w2*: (*WHILE b DO c*, *ps*) $\Rightarrow_A$ *1* ⇓ *ps* **by** *auto*
from *w1 w2 big_step_t_determ2* **have** *t1*: *t=1* **and** *pps*: *ps=ps′* **by** *auto*
from *assms* **show** (*?I* ∧∗ $ *1*) (*ps*,*n*)
unfolding *sep_conj_def dollar_def* **using** *t1 tn Q pps* **apply** *auto* **apply**(*rule exI*[**where** *x=n−1*]) **by** *auto*
next
assume (*?I* ∧∗ $ *1*) (*ps*,*n*)
with *assms* **have** *Q*: *Q*(*ps*,*n−1*) *n≥1* **unfolding** *dollar_def sep_conj_def* **by** *auto*
from *assms* **have** *w2*: (*WHILE b DO c*, *ps*) $\Rightarrow_A$ *1* ⇓ *ps* **by** *auto*
show *?wp* **unfolding** *wp3′_def*
apply *auto* **apply**(*rule exI*[**where** *x=ps*]) **apply**(*rule exI*[**where** *x=1*])
using *Q w2* **by** *auto*
qed

lemma *sF′*: $wp_{3'}$ (*WHILE b DO c*) *Q* (*ps*,*n*) = ((λ(*ps*, *n*). *vars b* ⊆ *dom ps* ∧ (*if pbval b ps then* $wp_{3'}$ *c* ($wp_{3'}$ (*WHILE b DO c*) *Q*) (*ps*, *n*) *else Q* (*ps*, *n*))) ∧∗ $ *1*) (*ps*,*n*)
apply(*cases vars b* ⊆ *dom ps*)
subgoal apply(*cases pbval b ps*) **using** *sFTrue sFFalse* **by** *auto*
subgoal by (*auto simp add*: *dollar_def wp3′_def sep_conj_def*)
done

lemma *sF*: $wp_{3'}$ (*WHILE b DO c*) *Q s* = ((λ(*ps*, *n*). *vars b* ⊆ *dom ps* ∧ (*if pbval b ps then* $wp_{3'}$ *c* ($wp_{3'}$ (*WHILE b DO c*) *Q*) (*ps*, *n*) *else Q* (*ps*, *n*))) ∧∗ $ *1*) *s*
using *sF′*
by (*metis* (*mono_tags*, *lifting*) *prod.case_eq_if prod.collapse sep_conj_impl1*)

lemma *strengthen_postR*: **assumes** $\vdash_{3a} \{P\}c\{Q\}$ $\bigwedge s.\ Q\ s \Longrightarrow Q'\ s$
shows $\vdash_{3a} \{P\}c\{\ Q'\}$
apply(*rule hoare3a.conseqS*)
apply (*rule assms*(*1*))
apply *simp* **by** (*fact assms*(*2*))

lemma assumes $\bigwedge Q.\ \vdash_{3a} \{wp_{3'}\ c\ Q\}\ c\ \{Q\}$
shows *WhileWpisPre*: $\vdash_{3a} \{wp_{3'}\ (WHILE\ b\ DO\ c)\ Q\}\ WHILE\ b\ DO\ c\ \{\ Q\}$
proof –
define *I* **where** $I \equiv (\lambda(ps,\ n).\ vars\ b \subseteq dom\ ps \wedge (if\ pbval\ b\ ps\ then\ wp_{3'}\ c\ (wp_{3'}\ (WHILE\ b\ DO\ c)\ Q)\ (ps,\ n)\ else\ Q\ (ps,\ n)))$
define I' **where** $I' \equiv (\lambda(ps,\ n).\ (if\ pbval\ b\ ps\ then\ wp_{3'}\ c\ (wp_{3'}\ (WHILE\ b\ DO\ c)\ Q)\ (ps,\ n)\ else\ Q\ (ps,\ n)))$
have I': $I = (\lambda(ps,\ n).\ vars\ b \subseteq dom\ ps \wedge I'\ (ps,\ n))$ **unfolding** *I_def* *I'_def* **by** *auto*

from *assms*[**where** Q=($wp_{3'}$ (*WHILE b DO c*) *Q*)] **have**
c: $\vdash_{3a} \{wp_{3'}\ c\ (wp_{3'}\ (WHILE\ b\ DO\ c)\ Q)\}\ c\ \{(wp_{3'}\ (WHILE\ b\ DO\ c)\ Q)\}$ **.**
have c': $\vdash_{3a} \{\ (\lambda(s,n).\ I\ (s,n) \wedge lmaps\_to\_axpr\ b\ True\ s)\ \}\ c\ \{\ I\ **\ \$1\ \}$
apply(*rule hoare3a.conseqS*)
apply(*rule c*)
subgoal apply *auto* **unfolding** *I_def* **by** *auto*
subgoal unfolding *I_def* **using** *sF* **by** *auto*
done

have c'': $\vdash_{3a} \{\ (\lambda(s,n).\ I\ (s,n) \wedge lmaps\_to\_axpr\ b\ True\ s)\ \}\ c\ \{\ (\lambda(s,\ n).\ I\ (s,\ n) \wedge vars\ b \subseteq dom\ s)\ \ **\ \$1\ \}$
apply(*rule strengthen_postR*[*OF* c'])
unfolding I'
by (*smt R case_prod_beta prod.sel*(*1*) *prod.sel*(*2*))

have *ka*: $(\lambda(s,\ n).\ I\ (s,\ n) \wedge vars\ b \subseteq dom\ s) = I$
apply *rule* **unfolding** I' **by** *auto*

from *hoare3a.While*[**where** P=I] c'' **have**
w: $\vdash_{3a} \{\ (\lambda(s,n).\ I\ (s,n) \wedge\ vars\ b \subseteq dom\ s)\ **\ \$1\ \}\ WHILE\ b\ DO\ c\ \{\ \lambda(s,n).\ I\ (s,n) \wedge lmaps\_to\_axpr\ b\ False\ s\ \}$ **.**

show $\vdash_{3a} \{wp_{3'}\ (WHILE\ b\ DO\ c)\ Q\}\ WHILE\ b\ DO\ c\ \{\ Q\}$

apply(*rule hoare3a.conseqS*)
apply(*rule w*)
subgoal unfolding *ka* **using** *sF I_def* **by** *simp*
subgoal unfolding *I_def* **by** *auto*
done
qed

lemma *wpT_is_pre*: $\vdash_{3a}$ $\{wp_{3'}\ c\ Q\}\ c\ \{\ Q\}$
proof (*induction c arbitrary*: *Q*)
case *SKIP*
then show *?case* **apply** *auto*
using *hoare3a.Frame*[**where** *F*=*Q* **and** *Q*=\$*0* **and** *P*=\$*1*, *OF hoare3a.Skip*]
by (*auto simp*: *sep.add_ac*)
next
case (*Assign x1 x2*)
then show *?case* **using** *hoare3a.Assign4* **by** *simp*
next
case (*Seq c1 c2*)
then show *?case* **apply** *auto*
apply(*subst hoare3a.Seq*[*rotated*]) **by** *auto*
next
case (*If x1 c1 c2*)
then show *?case* **apply** *auto*
apply(*rule weakenpreR*[*OF hoare3a.If*, **where** *P1*=%(*ps*,*n*). $wp_{3'}$ (*if pbval x1 ps then c1 else c2*) *Q* (*ps*,*n*)])
apply *auto*
subgoal apply(*rule hoare3a.conseqS*[**where** *P*=$wp_{3'}$ *c1 Q* **and** *Q*=*Q*]) **by** *auto*
subgoal apply(*rule hoare3a.conseqS*[**where** *P*=$wp_{3'}$ *c2 Q* **and** *Q*=*Q*]) **by** *auto*
proof −
fix *a b*
assume ((λ(*ps*, *t*). *vars x1* $\subseteq$ *dom ps* $\wedge$ $wp_{3'}$ (*if pbval x1 ps then c1 else c2*) *Q* (*ps*, *t*)) $\wedge$∗ \$ (*Suc 0*)) (*a*, *b*)
then show ((λ(*ps*, *t*). $wp_{3'}$ (*if pbval x1 ps then c1 else c2*) *Q* (*ps*, *t*) $\wedge$ *vars x1* $\subseteq$ *dom ps*) $\wedge$∗ \$ (*Suc 0*)) (*a*, *b*)
unfolding *sep_conj_def* **apply** *auto* **apply**(*case_tac pbval x1 aa*) **apply** *auto* **done**
qed
next
case (*While b c*)
with *WhileWpisPre* **show** *?case* **.**
qed

lemma *hoare3o_valid_alt*: $\models_{3'}$ { P } c { Q } $\Longrightarrow$
 ($\exists$ $k>0$. ($\forall ps$ n. P (ps,n div k)
 $\longrightarrow$ ($\exists ps'$ ps'' m e e'. ((c,ps) $\Rightarrow_A$ m $\Downarrow$ ps' + ps'')
 $\wedge$ ps' ## ps'' $\wedge$ n = e + e' + m
 $\wedge$ Q (ps',e div k))))
proof –
 assume $\models_{3'}$ {P} c { Q}
 from *this*[*unfolded hoare3o_valid_def*] **obtain** k **where** *k0*: $k>0$ **and**
 P: $\bigwedge ps$ n. P (ps, n) $\Longrightarrow$ ($\exists ps'$ ps'' m e e'. (c, ps) $\Rightarrow_A$ m $\Downarrow$ ps' + ps'' $\wedge$ ps' ## ps'' $\wedge$ $k * n$ = $k * e$ + e' + m $\wedge$ Q (ps', e))
 by *blast*
 show *?thesis* **apply**(*rule exI*[**where** x=k])
 apply *safe* **apply** *fact*
 proof –
 fix *ps n*
 assume P (ps, n div k)
 with P **obtain** ps' ps'' m e e' **where** *1*: (c, ps) $\Rightarrow_A$ m $\Downarrow$ ps' + ps'' ps' ## ps'' **and** e: k * (n div k) = k * e + e' + m **and** Q: Q (ps', e)
 by *blast*
 have k * (n div k) $\leq$ n **using** *k0*
 by *simp*
 then obtain e'' **where** n = k * (n div k) + e'' **using** *le_Suc_ex* **by** *blast*
 also have ... = k * e + e' + e'' + m **using** e **by** *auto*
 finally have n = k * e + (e'+e'') + m **and** Q (ps', (k*e) div k) **using** *Q k0* **by** *auto*
 with *1*
 show $\exists ps'$ ps'' m e e'. (c, ps) $\Rightarrow_A$ m $\Downarrow$ ps' + ps'' $\wedge$ ps' ## ps'' $\wedge$ n = e + e' + m $\wedge$ Q (ps', e div k) **by** *blast*
 qed
qed

lemma *valid_alternative_with_GC*: **assumes** ($\forall ps$ n. P (ps,n)
 $\longrightarrow$ ($\exists ps'$ ps'' m e e'. ((c,ps) $\Rightarrow_A$ m $\Downarrow$ ps' + ps'')
 $\wedge$ ps' ## ps'' $\wedge$ n = e + e' + m
 $\wedge$ Q (ps',e))) **shows** ($\forall ps$ n. P (ps,n)
 $\longrightarrow$ ($\exists ps'$ m e . ((c,ps) $\Rightarrow_A$ m $\Downarrow$ ps')
 $\wedge$ n = e + m $\wedge$ (Q ** *sep_true*) (ps',e)))
proof *safe*
 fix *ps n*
 assume P: P (ps,n)
 with *assms* **obtain** ps' ps'' m e e' **where** c: (c,ps) $\Rightarrow_A$ m $\Downarrow$ ps' + ps''

and
ps: ps' ## ps'' **and** n: $n = e + e' + m$ **and** Q: Q (ps',e) **by** *blast*
show $\exists ps'\ m\ e.\ (c,\ ps) \Rightarrow_A m \Downarrow ps' \wedge n = e + m \wedge (Q \wedge\!* (\lambda s.\ True))$ $(ps',\ e)$
apply(*rule exI*[**where** $x{=}ps' + ps''$])
apply(*rule exI*[**where** $x{=}m$])
apply(*rule exI*[**where** $x{=}e{+}e'$])
apply *safe* **apply** *fact* **apply** *fact*
unfolding *sep_conj_def* **apply** *simp* **apply**(*rule exI*[**where** $x{=}ps'$]) **apply**(*rule exI*[**where** $x{=}e$])
apply(*rule exI*[**where** $x{=}ps''$]) **apply** *safe* **apply** *fact* **apply**(*rule exI*[**where** $x{=}e'$]) **apply** *simp*
apply *fact* **done**
qed

lemma *hoare3o_valid_GC*: $\models_{3'} \{P\}\ c\ \{\ Q\ \} \Longrightarrow\ \models_{3'} \{P\}\ c\ \{\ Q\ **\ sep\_true\}$
proof –
assume $\models_{3'} \{P\}\ c\ \{\ Q\ \}$
then obtain k **where** $k{>}0$ **and** P: $\bigwedge ps\ n.\ P\ (ps,\ n) \Longrightarrow (\exists ps'\ ps''\ m\ e\ e'.\ (c,\ ps) \Rightarrow_A m \Downarrow ps' + ps'' \wedge ps'\ \#\#\ ps'' \wedge k * n = k * e + e' + m \wedge Q\ (ps',\ e))$
unfolding *hoare3o_valid_def* **by** *blast*
show $\models_{3'} \{P\}\ c\ \{\ Q\ **\ sep\_true\}$ **unfolding** *hoare3o_valid_def*
apply(*rule exI*[**where** $x{=}k$])
apply *safe* **apply** *fact*
proof –
fix $ps\ n$
assume $P\ (ps,\ n)$
with P **obtain** $ps'\ ps''\ m\ e\ e'$ **where** $(c,\ ps) \Rightarrow_A m \Downarrow ps' + ps''$ ps' ## ps'' $k * n = k * e + e' + m$ **and** Q: $Q\ (ps',\ e)$
by *blast*

show $\exists ps'\ ps''\ m\ e\ e'.\ (c,\ ps) \Rightarrow_A m \Downarrow ps' + ps'' \wedge ps'\ \#\#\ ps'' \wedge k * n = k * e + e' + m \wedge (Q \wedge\!* (\lambda s.\ True))\ (ps',\ e)$
apply(*rule exI*[**where** $x{=}ps'$])
apply(*rule exI*[**where** $x{=}ps''$])
apply(*rule exI*[**where** $x{=}m$])
apply(*rule exI*[**where** $x{=}e$])
apply(*rule exI*[**where** $x{=}e'$])
apply *safe* **apply** *fact+* **unfolding** *sep_conj_def* **apply**(*rule exI*[**where** $x{=}(ps',\ e)$]) **apply**(*rule exI*[**where** $x{=}0$]) **using** Q **by** *simp*
qed

qed

lemma *hoare3a_sound_GC*: $\vdash_{3a}$ $\{P\}$ c $\{$ $Q\}$ $\Longrightarrow$ $\models_{3'}$ $\{P\}$ c $\{$ Q $**$ *sep_true*$\}$ **using** *hoare3o_valid_GC hoareT_sound2_partR* **by** *auto*

lemma *valid_wp*: $\models_{3'}$ $\{P\}$ c $\{$ $Q\}$ $\Longrightarrow$ $(\exists\, k{>}0.\ \forall\, s\ n.\ P\ (s,\ n) \longrightarrow wp_{3'}\ c$ $(\lambda(ps,\ n).\ (Q\ **\ sep\_true)\ (ps,\ n\ div\ k))\ (s,\ k * n))$
proof −
  let $?P = \lambda k\ (ps,n).\ P\ (ps,n\ div\ k)$
  let $?Q = \lambda k\ (ps,n).\ Q\ (ps,n\ div\ k)$
  let $?QG = \lambda k\ (ps,n).\ (Q\ **\ sep\_true)\ (ps,n\ div\ k)$
  assume $\models_{3'}$ $\{P\}$ c $\{$ $Q\}$
  then obtain k **where** $k[simp]$: $k{>}0$ **and** $(\forall\, ps\ n.\ P\ (ps,n\ div\ k)$
 $\longrightarrow (\exists\, ps'\ ps''\ m\ e\ e'.\ ((c,ps) \Rightarrow_A m \Downarrow ps' + ps'')$
    $\wedge\ ps'\ \#\#\ ps'' \wedge n = e + e' + m$
    $\wedge\ Q\ (ps',e\ div\ k)))$ **using** *hoare3o_valid_alt* **by** *blast*
  then have $(\forall\, ps\ n.\ ?P\ k\ (ps,n)$
 $\longrightarrow (\exists\, ps'\ ps''\ m\ e\ e'.\ ((c,ps) \Rightarrow_A m \Downarrow ps' + ps'')$
    $\wedge\ ps'\ \#\#\ ps'' \wedge n = e + e' + m$
    $\wedge\ ?Q\ k\ (ps',e)))$ **by** *auto*
  then have f: $(\forall\, ps\ n.\ ?P\ k\ (ps,n) \longrightarrow (\exists\, ps'\ \ m\ e\ .\ ((c,ps) \Rightarrow_A m \Downarrow ps'$
$)$
     $\wedge\ n = e + m \wedge (?Q\ k\ **\ sep\_true)\ (ps',e)))$
    apply(*rule valid_alternative_with_GC*) **done**

  have $\bigwedge s\ n.\ P\ (s,\ n) \Longrightarrow wp_{3'}\ c\ (\lambda(ps,\ n).\ (Q\ **\ sep\_true)\ (ps,\ n\ div\ k))$
$(s,\ k * n)$
    unfolding *wp3'_def* **apply** *auto*
  proof −
    fix *ps n*
    assume $P\ (ps,n)$
    then have $P\ (ps,(k{*}n)\ div\ k)$ **apply** *simp* **done**
    with f **obtain** $ps'\ m\ e$ **where** $((c,ps) \Rightarrow_A m \Downarrow ps'\ \ )$ **and** z: $k * n = e$
$+ m$
      and Q: $(?Q\ k\ **\ sep\_true)\ (ps',e)$ **by** *blast*
    from z **have** e: $e = k * n - m$ **by** *auto*
    from $Q[unfolded\ e\ sep\_conj\_def]$ **obtain** *ps1 ps2 e1 e2* **where**
     $ps1\ \#\#\ ps2\ (ps' = ps1\ {+}ps2)$ **and** eq: $k * n - m = e1 + \ e2$ **and**
Q: $Q\ (ps1,\ e1\ div\ k)$ **by** *force*

    let $?f = (e1 + e2)\ div\ k - (e1\ div\ k + (e2\ div\ k))$
    have kl: $(e1 + e2)\ div\ k \geq (e1\ div\ k + (e2\ div\ k))$ **using** k
     using *div_add1_eq le_iff_add* **by** *blast*

show $\exists\, t\ m.\ m \leq k * n \wedge (c,\ ps) \Rightarrow_A m \Downarrow t \wedge (Q \wedge\!* (\lambda s.\ True))\ (t,\ (k * n - m)\ div\ k)$
  apply(*rule exI*[**where** *x*=*ps'*])
  apply(*rule exI*[**where** *x*=*m*]) **apply** *safe* **using** *z* **apply** *simp*
  apply *fact* **unfolding** *e* **unfolding** *sep_conj_def*
  apply(*rule exI*[**where** *x*=(*ps1*,*e1 div k*)])
  apply(*rule exI*[**where** *x*=(*ps2*,*e2 div k*+*?f*)]) **apply** *auto* **apply** *fact*+
  unfolding *eq* **using** *kl*
  apply *force* **using** *Q* **by** *auto*
qed
then show $(\exists\, k{>}0.\ \forall\, s\ n.\ P\ (s,\ n) \longrightarrow wp_{3'}\ c\ (\lambda(ps,\ n).\ (Q ** sep\_true)\ (ps,\ n\ div\ k))\ (s,\ k * n))$
  using *k* **by** *metis*
qed

theorem *completeness*: $\models_{3'} \{P\}\ c\ \{\ Q\} \Longrightarrow \vdash_{3b} \{P\}\ c\ \{\ Q ** sep\_true\}$
proof –
let $?P = \lambda k\ (ps,n).\ P\ (ps,n\ div\ k)$
let $?Q = \lambda k\ (ps,n).\ Q\ (ps,n\ div\ k)$
let $?QG = \lambda k\ (ps,n).\ (Q ** sep\_true)\ (ps,n\ div\ k)$
assume $\models_{3'} \{P\}\ c\ \{\ Q\}$
then obtain *k* **where** *k*[*simp*]: $k{>}0$ **and** *P*: $\bigwedge s\ n.\ P\ (s,\ n) \Longrightarrow wp_{3'}\ c\ (\lambda(ps,\ n).\ (Q ** sep\_true)\ (ps,\ n\ div\ k))\ (s,\ k * n)$
  using *valid_wp* **by** *blast*

from *wpT_is_pre* **have** *R*: $\vdash_{3a} \{wp_{3'}\ c\ (?QG\ k)\}\ c\ \{?QG\ k\}$ **by** *auto*

show $\vdash_{3b} \{P\}\ c\ \{\ Q ** sep\_true\}$
  apply(*rule hoare3b.conseq*[*OF hoare3b.import*[*OF R*], **where** *k*=*k*])
  subgoal for *s n* **by** (*fact P*)
  apply *simp* **by** (*fact k*)
qed

thm *E_extendsR completeness*

lemma *completenessR*: $\models_{3'} \{P\}\ c\ \{\ Q\} \Longrightarrow \vdash_{3'} \{P\}\ c\ \{\ Q ** sep\_true\}$
using *E_extendsS completeness* **by** *metis*

end
theory *SepLogK_VCG*
imports *SepLogK_Hoare*

begin

lemmas *conseqS* = *conseq*[**where** *k=1*, *simplified*]

datatype *acom* =
  Askip (‹*SKIP*›) |
  Aassign vname aexp (‹(_ ::= _)› [*1000*, *61*] *61*) |
  Aseq acom acom (‹_;;/ _› [*60*, *61*] *60*) |
  Aif bexp acom acom (‹(*IF* _/ *THEN* _/ *ELSE* _)› [*0*, *0*, *61*] *61*) |
  Awhile assn2 bexp acom (‹({_}/ *WHILE* _/ *DO* _)› [*0*, *0*, *61*] *61*)

notation *com.SKIP* (‹*SKIP*›)

fun *strip* :: *acom* ⇒ *com* **where**
strip SKIP = *SKIP* |
strip (*x* ::= *a*) = (*x* ::= *a*) |
strip (C_1;; C_2) = (*strip* C_1;; *strip* C_2) |
strip (*IF b THEN* C_1 *ELSE* C_2) = (*IF b THEN strip* C_1 *ELSE strip* C_2) |
strip ({_} *WHILE b DO C*) = (*WHILE b DO strip C*)

fun *pre* :: *acom* ⇒ *assn2* ⇒ *assn2* **where**
pre SKIP Q = ($*1* ∗∗ *Q*) |
pre (*x* ::= *a*) *Q* = ((λ(*ps*,*t*). *x*∈*dom ps* ∧ *vars a* ⊆ *dom ps* ∧ *Q* (*ps*(*x*↦(*paval a ps*)),*t*)) ∗∗ $*1*) |
pre (C_1;; C_2) *Q* = *pre* C_1 (*pre* C_2 *Q*) |
pre (*IF b THEN* C_1 *ELSE* C_2) *Q* = (
  $*1* ∗∗ (λ(*ps*,*n*). *vars b* ⊆ *dom ps* ∧ (*if pbval b ps then pre* C_1 *Q* (*ps*,*n*) *else pre* C_2 *Q* (*ps*,*n*)))) |
pre ({*I*} *WHILE b DO C*) *Q* = (*I* ∗∗ $*1*)

fun *vc* :: *acom* ⇒ *assn2* ⇒ *bool* **where**
vc SKIP Q = *True* |
vc (*x* ::= *a*) *Q* = *True* |
vc (C_1 ;; C_2) *Q* = ((*vc* C_1 (*pre* C_2 *Q*)) ∧ (*vc* C_2 *Q*)) |
vc (*IF b THEN* C_1 *ELSE* C_2) *Q* = (*vc* C_1 *Q* ∧ *vc* C_2 *Q*) |
vc ({*I*} *WHILE b DO C*) *Q* = ((∀*s*. (*I s* ⟶ *vars b* ⊆ *dom* (*fst s*)) ∧ ((λ(*s*,*n*). *I* (*s*,*n*) ∧ *lmaps_to_axpr b True s*) *s* ⟶ *pre C* (*I* ∗∗ $ *1*) *s*)
  ∧ ((λ(*s*,*n*). *I* (*s*,*n*) ∧ *lmaps_to_axpr b False s*) *s* ⟶ *Q s*)) ∧ *vc C* (*I* ∗∗ $ *1*))

lemma *dollar0_left*: ($ *0* ∧∗ *Q*) = *Q*

apply *rule* **unfolding** *dollar_def sep_conj_def*
by *force*

lemma *vc_sound*: $vc\ C\ Q \Longrightarrow\ \vdash_{3'} \{pre\ C\ Q\}\ strip\ C\ \{\ Q\ \}$
proof (*induct C arbitrary*: *Q*)
case *Askip*
then show *?case*
apply *simp*
apply(*rule conseqS*[*OF Frame*[*OF Skip*]])
by (*auto simp*: *dollar0_left*)
next
case (*Aassign x1 x2*)
then show *?case*
apply *simp*
apply(*rule conseqS*)
apply(*rule Assign4*)
apply *auto* **done**
next
case (*Aseq C1 C2*)
then show *?case* **apply** (*auto intro*: *Seq*) **done**
next
case (*Aif b C1 C2*)
then have *Aif1*: $\vdash_{3'} \{pre\ C1\ Q\}\ strip\ C1\ \{Q\}$
and *Aif2*: $\vdash_{3'} \{pre\ C2\ Q\}\ strip\ C2\ \{Q\}$ **by** *auto*
show *?case* **apply** *simp*
apply (*rule conseqS*)
apply(*rule If*[**where** $P=\%(ps,n).\ (if\ pbval\ b\ ps\ then\ pre\ C1\ Q\ (ps,n)$ $else\ pre\ C2\ Q\ (ps,n))$ **and** $Q=Q$])
subgoal apply *simp*
apply (*rule conseqS*) **apply**(*fact Aif1*) **by** *auto*
subgoal apply *simp*
apply (*rule conseqS*) **apply**(*fact Aif2*) **by** *auto*
apply (*auto simp*: *sep_conj_ac*)
unfolding *sep_conj_def* **by** *blast*
next
case (*Awhile I b C*)
then have
dom : $\bigwedge s.\ (I\ s \longrightarrow vars\ b \subseteq dom\ (fst\ s)\ )$
and *i*: $\bigwedge s.\ \ (\lambda(s,n).\ I\ (s,n) \wedge lmaps\_to\_axpr\ b\ True\ s)\ s \longrightarrow pre\ C$ $(I\ {*}{*}\ \$\ 1)\ s$
and *ii*: $\bigwedge s.\ \ (\lambda(s,n).\ I\ (s,n) \wedge lmaps\_to\_axpr\ b\ False\ s)\ s \longrightarrow Q\ s$
and *C*: $\vdash_{3'} \{pre\ C\ (I\ {*}{*}\ \$\ 1)\}\ strip\ C\ \{I\ {*}{*}\ \$\ 1\}$
by *fastforce+*

```
  show ?case
    apply simp
    apply(rule conseqS)
       apply(rule While[where P=I])
       apply(rule conseqS)
        apply(rule C)
    subgoal using i by auto
    subgoal apply simp using dom unfolding sep_conj_def by force
    subgoal apply simp using dom unfolding sep_conj_def by force
    subgoal using ii apply auto done
    done
qed
```

lemma *vc2valid*: $vc\ C\ Q \Longrightarrow \forall s.\ P\ s \longrightarrow pre\ C\ Q\ s \Longrightarrow \models_{3'} \{P\}\ strip\ C\ \{\ Q\}$
  using *hoareT_sound2_part weakenpre vc_sound* **by** *metis*

lemma *pre_mono*: **assumes** $\forall s.\ P\ s \longrightarrow Q\ s$ **shows** $\bigwedge s.\ pre\ C\ P\ s \Longrightarrow pre\ C\ Q\ s$

```
using assms proof(induct C arbitrary: P Q)
  case Askip
  then show ?case  apply (auto simp: sep_conj_def dollar_def)
    by force
next
  case (Aassign x1 x2)
  then show ?case by (auto simp: sep_conj_def dollar_def)
next
  case (Aseq C1 C2)
  then show ?case by auto
next
  case (Aif b C1 C2)
  then show ?case apply (auto simp: sep_conj_def dollar_def)
    subgoal for ps n
      apply(rule exI[where x=0])
      apply(rule exI[where x=1])
      apply(rule exI[where x=ps]) by auto
    done
next
  case (Awhile x1 x2 C)
  then show ?case by auto
qed
```

lemma *vc_mono*: **assumes** $\forall s.\ P\ s \longrightarrow Q\ s$ **shows** $vc\ C\ P \Longrightarrow vc\ C\ Q$

using *assms* **proof**(*induct C arbitrary*: *P Q*)
  case *Askip*
  then show *?case* **by** *auto*
next
  case (*Aassign x1 x2*)
  then show *?case* **by** *auto*
next
  case (*Aseq C1 C2 P Q*)
  then have *i*: *vc C1* (*pre C2 P*) **and** *ii*: *vc C2 P* **by** *auto*
  from *pre_mono*[*OF*] *Aseq*(*4*) **have** *iii*: $\forall$ *s*. *pre C2 P s* $\longrightarrow$ *pre C2 Q s* **by** *blast*
  show *?case* **apply** *auto*
    using *Aseq*(*1*)[*OF i iii*] *Aseq*(*2*)[*OF ii Aseq*(*4*)] **by** *auto*
next
  case (*Aif x1 C1 C2*)
  then show *?case* **by** *auto*
next
  case (*Awhile I b C P Q*)
  then show *?case* **by** *auto*
qed

lemma *vc_sound'*: $vc\ C\ Q \Longrightarrow (\bigwedge s\ n.\ P'\ (s,\ n) \Longrightarrow pre\ C\ Q\ (s,\ k * n)) \Longrightarrow (\bigwedge s\ n.\ Q\ (s,\ n) \Longrightarrow Q'\ (s,\ n\ div\ k)) \Longrightarrow 0 < k \Longrightarrow \vdash_{3'} \{P'\}\ strip\ C\ \{Q'\}$
  using *conseq vc_mono vc_sound* **by** *metis*

lemma *pre_Frame*: $(\forall s.\ P\ s \longrightarrow pre\ C\ Q\ s) \Longrightarrow vc\ C\ Q$
  $\Longrightarrow (\exists C'.\ strip\ C = strip\ C' \wedge\ vc\ C'\ (Q ** F) \wedge (\forall s.\ (P ** F)\ s \longrightarrow pre\ C'\ (Q ** F)\ s)\ )$
proof (*induct C arbitrary*: *P Q*)
  case *Askip*
  show *?case*
  proof (*rule exI*[**where** *x*=*Askip*], *safe*)
    fix *a b*
    assume $(P \wedge* F)\ (a,\ b)$
    then obtain *ps1 ps2 n1 n2* **where** *A*: *ps1##ps2 a=ps1+ps2 b=n1+n2*
      and *P*: *P* (*ps1*,*n1*) **and** *F*: *F* (*ps2*,*n2*) **unfolding** *sep_conj_def* **by** *auto*
    from *P Askip* **have** *p*: ($ (*Suc 0*) $\wedge*$ *Q*) (*ps1*, *n1*) **by** *auto*

```
    from p A F
    have (($ (Suc 0) ∧* Q) ∧* F) (a, b)
      apply(subst (2) sep_conj_def) by auto
    then show pre SKIP (Q ∧* F) (a, b) by (simp add: sep_conj_ac)
  qed simp
next
  case (Aassign x a)
  show ?case
  proof (rule exI[where x=Aassign x a], safe)
    fix ps n
    assume (P ∧* F) (ps,n)
   then obtain ps1 ps2 n1 n2 where o: ps1##ps2 ps=ps1+ps2 n=n1+n2
      and P: P (ps1,n1) and F: F (ps2,n2) unfolding sep_conj_def by
auto
    from P Aassign(1) have z: ((λ(ps, t). x ∈ dom ps ∧ vars a ⊆ dom ps
∧ Q (ps(x ↦ paval a ps), t))
                                  ∧* $ (Suc 0)) (ps1, n1)
        by auto
    with o F show  pre (x ::= a) (Q ∧* F) (ps,n) apply auto
      unfolding sep_conj_def dollar_def apply (auto)
        subgoal by(simp add: plus_fun_def)
        subgoal by(auto simp add: plus_fun_def)
        subgoal
       by (smt add_update_distrib dom_fun_upd domain_conv insert_dom
option.simps(3) paval_extend sep_disj_fun_def)
        done
  qed auto
next
  case (Aseq C1 C2)
  from Aseq(3) have pre: ∀ s. P s ⟶ pre C1 (pre C2 Q) s by auto
  from Aseq(4) have vc1: vc C1 (pre C2 Q) and vc2: vc C2 Q by auto
  from Aseq(1)[OF pre vc1] obtain C1' where S1: strip C1 = strip C1'
      and vc1': vc C1' (pre C2 Q ∧* F)
      and I1: (∀ s. (P ∧* F) s ⟶ pre C1' (pre C2 Q ∧* F) s) by blast
   from Aseq(2)[of pre C2 Q Q, OF _ vc2] obtain C2' where S2: strip
C2 = strip C2'
      and vc2': vc C2' (Q ∧* F)
      and I2: (∀ s. (pre C2 Q ∧* F) s ⟶ pre C2' (Q ∧* F) s)  by blast

  show ?case apply(rule exI[where x=Aseq C1' C2'])
    apply safe
    subgoal using S1 S2 by auto
    subgoal apply simp apply safe
```

```
      subgoal using vc_mono[OF I2 vc1'] .
      subgoal by (fact vc2')
    done
    subgoal using I1 I2 pre_mono
      by force
    done
next
  case (Aif b C1 C2)
  from Aif(3) have i: ∀s. P s ⟶
        ($ (Suc 0) ∧*
         (λ(ps, n). vars b ⊆ dom ps ∧ (if pbval b ps then pre C1 Q (ps, n)
else pre C2 Q (ps, n))))
         s by simp
  from Aif(4) have vc1: vc C1 Q and vc2: vc C2 Q by auto
   from Aif(1)[where P=pre C1 Q and Q=Q, OF _ vc1] obtain C1'
where
    s1: strip C1 = strip C1' and v1: vc C1' (Q ∧* F)
    and p1: (∀s. (pre C1 Q ∧* F) s ⟶ pre C1' (Q ∧* F) s)
    by auto
   from Aif(2)[where P=pre C2 Q and Q=Q, OF _ vc2] obtain C2'
where
    s2: strip C2 = strip C2' and v2: vc C2' (Q ∧* F)
    and p2: (∀s. (pre C2 Q ∧* F) s ⟶ pre C2' (Q ∧* F) s)
    by auto

  show ?case apply(rule exI[where x=Aif b C1' C2'])
  proof safe
    fix ps n
    assume (P ∧* F) (ps, n)
   then obtain ps1 ps2 n1 n2 where o: ps1##ps2 ps=ps1+ps2 n=n1+n2
      and P: P (ps1,n1) and F: F (ps2,n2) unfolding sep_conj_def by
auto
    from P i have P': ($ (Suc 0) ∧*
        (λ(ps, n). vars b ⊆ dom ps ∧ (if pbval b ps then pre C1 Q (ps, n)
else pre C2 Q (ps, n))))
        (ps1,n1) by auto
    have PF: (($ (Suc 0) ∧*
        (λ(ps, n). vars b ⊆ dom ps ∧ (if pbval b ps then pre C1 Q (ps, n)
else pre C2 Q (ps, n)))) ** F)
        (ps,n) apply(subst (2) sep_conj_def)
       apply(rule exI[where x=(ps1,n1)])
       apply(rule exI[where x=(ps2,n2)])
      using F P' o by auto
    from this[simplified sep_conj_assoc] obtain ps1 ps2 n1 n2 where o:
```

ps1##ps2 ps=ps1+ps2 n=n1+n2
and *P*: ($ (*Suc 0*)) (*ps1*,*n1*) **and** *F*: ((λ(*ps*, *n*). *vars b* ⊆ *dom ps* ∧ (*if pbval b ps then pre C1 Q* (*ps*, *n*) *else pre C2 Q* (*ps*, *n*))) ∧∗ *F*) (*ps2*,*n2*)
unfolding *sep_conj_def* **apply** *auto* **by** *fast*
then have ((λ(*ps*, *n*). *vars b* ⊆ *dom ps* ∧ (*if pbval b ps then* (*pre C1 Q* ∧∗ *F*) (*ps*, *n*) *else* (*pre C2 Q* ∧∗ *F*) (*ps*, *n*)))) (*ps2*,*n2*)
unfolding *sep_conj_def* **apply** *auto*
apply (*metis contra_subsetD domD map_add_dom_app_simps*(*1*) *plus_fun_conv sep_add_commute*)
using *pbval_extend* **apply** *auto*[*1*]
apply (*metis contra_subsetD domD map_add_dom_app_simps*(*1*) *plus_fun_conv sep_add_commute*)
using *pbval_extend* **apply** *auto*[*1*] **done**
then have ((λ(*ps*, *n*). *vars b* ⊆ *dom ps* ∧ (*if pbval b ps then* (*pre C1′* (*Q* ∧∗ *F*)) (*ps*, *n*) *else* (*pre C2′* (*Q* ∧∗ *F*)) (*ps*, *n*)))) (*ps2*,*n2*)
using *p1 p2* **by** *auto*
with *o P*
show *pre* (*IF b THEN C1′ ELSE C2′*) (*Q* ∧∗ *F*) (*ps*, *n*)
apply *auto* **apply**(*subst sep_conj_def*) **by** *force*
qed (*auto simp*: *s1 s2 v1 v2*)
next
case (*Awhile I b C*)
from *Awhile*(*2*) **have** *pre*: ∀ *s*. *P s* ⟶ (*I* ∗∗ $*1*) *s* **by** *auto*
from *Awhile*(*3*) **have**
dom: ∀ *ps n*. *I* (*ps*, *n*) ⟶ *vars b* ⊆ *dom ps*
and *tB*: ∀ *s*. *I s* ∧ *vars b* ⊆ *dom* (*fst s*) ∧ *pbval b* (*fst s*) ⟶ *pre C* (*I* ∧∗ $ (*Suc 0*)) *s*
and *fB*: ∀ *ps n*. *I* (*ps*, *n*) ∧ *vars b* ⊆ *dom ps* ∧ ¬ *pbval b ps* ⟶ *Q* (*ps*, *n*)
and *vcB*: *vc C* (*I* ∧∗ $(*Suc 0*)) **by** *auto*
from *Awhile*(*1*)[*OF tB vcB*] **obtain** *C′* **where** *st*: *strip C* = *strip C′*
and *vc′*: *vc C′* ((*I* ∧∗ $ (*Suc 0*)) ∧∗ *F*)
and *pre′*: (∀ *s*. ((λ*a*. *I a* ∧ *vars b* ⊆ *dom* (*fst a*) ∧ *pbval b* (*fst a*)) ∧∗ *F*) *s* ⟶
pre C′ ((*I* ∧∗ $ (*Suc 0*)) ∧∗ *F*) *s*)
by *auto*
show *?case* **apply**(*rule exI*[**where** *x=Awhile* (*I*∗∗*F*) *b C′*])
apply *safe*
subgoal using *st* **by** *simp*
subgoal apply *simp* **apply** *safe*
subgoal using *dom* **unfolding** *sep_conj_def* **apply** *auto*
by (*metis domD sep_substate_disj_add subState subsetCE*)
subgoal using *pre′* **apply**(*auto simp*: *sep_conj_ac*)
apply(*subst* (*asm*) *sep_conj_def*)

apply(*subst* (*asm*) *sep_conj_def*) **apply** *auto*
by (*metis dom pbval_extend sep_add_commute sep_disj_commuteI*)
subgoal using *fB* **unfolding** *sep_conj_def* **apply** *auto*
using *dom pbval_extend* **by** *fastforce*
subgoal using vc' **apply**(*auto simp*: *sep_conj_ac*) **done**
done
subgoal apply *simp* **using** *pre* **unfolding** *sep_conj_def* **apply** *auto*
by (*smt semiring_normalization_rules*(*23*) *sep_add_assoc sep_add_commute sep_add_disjD sep_add_disjI1*)
done
qed

lemma *vc_complete*: $\vdash_{3a}$ $\{P\}$ c $\{$ Q $\}$ $\Longrightarrow$ $(\exists\, C.\ vc\ C\ Q \wedge (\forall\, s.\ P\ s \longrightarrow pre\ C\ Q\ s) \wedge strip\ C = c)$
proof(*induct rule*: *hoare3a.induct*)
case *Skip*
then show *?case* **apply**(*rule exI*[**where** *x*=*Askip*]) **by** *auto*
next
case (*Assign4 x a Q*)
then show *?case* **apply**(*rule exI*[**where** *x*=*Aassign x a*]) **by** *auto*
next
case (*If P b* c_1 *Q* c_2)
from *If*(*2*) **obtain** *C1* **where** *A1*: *vc C1 Q strip C1* = c_1 **and**
A2: $\bigwedge$*ps n*. (*P* (*ps*, *n*) $\wedge$ *lmaps_to_axpr b True ps*) $\longrightarrow$ *pre C1 Q* (*ps*,*n*)
by *blast*
from *If*(*4*) **obtain** *C2* **where** *B1*: *vc C2 Q strip C2* = c_2 **and** *B2*:
$\bigwedge$*ps n*. (*P* (*ps*, *n*) $\wedge$ *lmaps_to_axpr b False ps*) $\longrightarrow$ *pre C2 Q* (*ps*,*n*)
by *blast*

show *?case* **apply**(*rule exI*[**where** *x*=*Aif b C1 C2*]) **using** *A1 B1* **apply** *auto*
subgoal for *ps n*
unfolding *sep_conj_def dollar_def* **apply** *auto*
apply(*rule exI*[**where** *x*=*0*])
apply(*rule exI*[**where** *x*=*1*])
apply(*rule exI*[**where** *x*=*ps*])
using *A2 B2* **by** *auto*
done
next

case (*Frame P C Q F*)
then obtain C' **where** *vc*: *vc* C' *Q* **and** *pre*: ($\forall$ *s*. *P s* $\longrightarrow$ *pre* C' *Q s*)
and *strip*: *strip* $C' = C$ **by** *auto*
show *?case* **using** *pre_Frame*[*OF pre vc*] *strip* **by** *metis*
next
case (*Seq P* c_1 *Q* c_2 *R*)
from *Seq*(*2*) **obtain** *C1* **where** *A1*: *vc C1 Q strip C1* = c_1 **and**
A2: $\bigwedge$*s*. *P s* $\longrightarrow$ *pre C1 Q s*
by *blast*
from *Seq*(*4*) **obtain** *C2* **where** *B1*: *vc C2 R strip C2* = c_2 **and**
B2: $\bigwedge$*s*. *Q s* $\longrightarrow$ *pre C2 R s*
by *blast*
show *?case* **apply**(*rule exI*[**where** *x*=*Aseq C1 C2*])
using *B1 A1* **apply** *auto*
subgoal using *vc_mono B2* **by** *auto*
subgoal apply(*rule pre_mono*[**where** *P*=*Q*]) **using** *B2* **apply** *auto*
using *A2* **by** *auto*
done
next
case (*While I b c*)
then obtain *C* **where** *1*: *vc C* ((λ(*s*, *n*). *I* (*s*, *n*) $\wedge$ *vars b* $\subseteq$ *dom s*) $\wedge *$
\$ *1*)
strip C = *c* **and** *2*:
$\bigwedge$*ps n*. (*I* (*ps*, *n*) $\wedge$ *lmaps_to_axpr b True ps*) $\longrightarrow$
pre C ((λ(*s*, *n*). *I* (*s*, *n*) $\wedge$ *vars b* $\subseteq$ *dom s*) $\wedge *$ \$ *1*) (*ps*,*n*) **by**
blast

show *?case* **apply**(*rule exI*[**where** *x*=*Awhile* (λ(*s*, *n*). *I* (*s*, *n*) $\wedge$ *vars b*
$\subseteq$ *dom s*) *b C*])
using *1 2* **by** *auto*
next
case (*conseqS P c Q* P' Q')
then obtain C' **where** C': *vc* C' *Q* ($\forall$ *s*. *P s* $\longrightarrow$ *pre* C' *Q s*) *strip* C' =
c
by *blast*
show *?case* **apply**(*rule exI*[**where** *x*=C'])
using C' *conseqS*(*3*,*4*) *pre_mono vc_mono* **by** *force*
qed

theorem *vc_completeness*:
assumes $\models_{3'}$ $\{P\}$ *c* $\{Q\}$

shows $\exists\, C\ k.\ vc\ C\ (Q\ {*}{*}\ sep\_true)$
$\wedge\ (\forall\, ps\ n.\ P\ (ps,\ n) \longrightarrow pre\ C\ (\lambda(ps,\ n).\ (Q\ {*}{*}\ sep\_true)\ (ps,\ n\ div\ k))\ (ps,\ k * n))$
$\wedge\ strip\ C = c$
proof −
let $?QG = \lambda k\ (ps,n).\ (Q\ {*}{*}\ sep\_true)\ (ps,n\ div\ k)$
from *assms* **obtain** k **where** $k[simp]$: $k{>}0$ **and** p: $\bigwedge ps\ n.\ P\ (ps,\ n) \Longrightarrow wp_{3'}\ c\ (\lambda(ps,\ n).\ (Q\ {*}{*}\ sep\_true)\ (ps,\ n\ div\ k))\ (ps,\ k * n)$
using *valid_wp* **by** *blast*

from *wpT_is_pre* **have** R: $\vdash_{3a}\ \{wp_{3'}\ c\ (?QG\ k)\}\ c\ \{?QG\ k\}$ **by** *auto*

have z: $(\forall\, s.\ (\lambda(ps,\ n).\ (Q \wedge{*}\ (\lambda s.\ True))\ (ps,\ n\ div\ k))\ s) \Longrightarrow (\forall\, s.\ (\lambda(ps,\ n).\ (Q \wedge{*}\ (\lambda s.\ True))\ (ps,\ n))\ s)$
by (*metis* (*no_types*) *case_prod_conv k neq0_conv nonzero_mult_div_cancel_left old.prod.exhaust*)

have z: $\bigwedge ps\ n.\ ((Q \wedge{*}\ (\lambda s.\ True))\ (ps,\ n\ div\ k) \Longrightarrow (Q \wedge{*}\ (\lambda s.\ True))\ (ps,\ n))$
proof −
fix *ps n*
assume $(Q \wedge{*}\ (\lambda s.\ True))\ (ps,\ n\ div\ k)$
then obtain *ps1 n1 ps2 n2*
where o: $ps1\ \#\#\ ps2\ ps = ps1 + ps2\ Q\ (ps1,\ n1)\ n\ div\ k = n1 + n2$
unfolding *sep_conj_def* **by** *auto*
from $o(4)$ **have** $nn1$: $n{\geq}n1$ **using** k
by (*metis* (*full_types*) *add_leE div_le_dividend*)
show $(Q \wedge{*}\ (\lambda s.\ True))\ (ps,\ n)$ **unfolding** *sep_conj_def*
apply(*rule exI*[**where** $x{=}(ps1,\ n1)$])
apply(*rule exI*[**where** $x{=}(ps2,\ n - n1)$])
using *o nn1* **by** *auto*
qed
then have z': $\forall\, s.\ ((Q \wedge{*}\ (\lambda s.\ True))\ (fst\ s,\ (snd\ s)\ div\ k) \longrightarrow (Q \wedge{*}\ (\lambda s.\ True))\ s)$
by (*metis prod.collapse*)

from *vc_complete*[*OF R*] **obtain** C
where o: $vc\ C\ (\lambda(ps,\ n).\ (Q \wedge{*}\ (\lambda s.\ True))\ (ps,\ n\ div\ k))$
$\forall\, a\ b.\ wp_{3'}\ (strip\ C)\ (\lambda(ps,\ n).\ (Q \wedge{*}\ (\lambda s.\ True))\ (ps,\ n\ div\ k))\ (a,\ b) \longrightarrow$
$pre\ C\ (\lambda(ps,\ n).\ (Q \wedge{*}\ (\lambda s.\ True))\ (ps,\ n\ div\ k))\ (a,\ b)$
$c = strip\ C$ **by** *auto*

have y: $\bigwedge ps\ n.\ P\ (ps,\ n) \Longrightarrow\ pre\ C\ (\lambda(ps,\ n).\ (Q \wedge\!* (\lambda s.\ True))\ (ps,\ n\ div\ k))\ (ps,\ k * n)$
using *o p* **by** *metis*

show *?thesis* **apply**(*rule exI*[**where** x=C]) **apply**(*rule exI*[**where** x=k])
apply *safe*
subgoal apply(*rule vc_mono*[*OF _ o*(*1*)]) **using** *z* **by** *blast*
subgoal using *y* **by** *blast*
subgoal using *o* **by** *simp*
done
qed

end

10 Discussion

10.1 Relation between the explicit Hoare logics

theory *Discussion*
imports *Quant_Hoare SepLog_Hoare*
begin

10.1.1 Relation SepLogic to quantHoare

definition *em* **where** $em\ P' = (\%(ps,n).\ P'\ (emb\ ps\ (\%\_.\ 0)) \le enat\ n\ )$

lemma assumes s: $\models_3 \{\ em\ P'\}\ c\ \{\ em\ Q'\ \}$
shows $\models_2 \{\ P'\ \}\ c\ \{\ Q'\ \}$
proof −
from s **have** s': $\bigwedge ps\ n.\ em\ P'\ (ps,\ n) \Longrightarrow (\exists\, ps'\ m.\ (c,\ ps) \Rightarrow_A m \Downarrow ps' \wedge m \le n \wedge em\ Q'\ (ps',\ n - m))$ **unfolding** *hoare3_valid_def* **by** *auto*
{
fix s
assume P': $P'\ s < \infty$
then obtain n **where** n: $P'\ s = enat\ n$
by *fastforce*
with P' **have** $em\ P'\ (part\ s,\ n)$ **unfolding** *em_def* **by** *auto*
with s' **obtain** $ps'\ m$ **where** c: $(c,\ part\ s) \Rightarrow_A m \Downarrow ps'$ **and** m: $m \le n$ **and** Q': $em\ Q'\ (ps',\ n - m)$ **by** *blast*

from Q' **have** q: $Q'\ (emb\ ps'\ (\lambda\_.\ 0)) \le enat\ (\ n - m)$ **unfolding** *em_def* **by** *auto*

thm *full_to_part part_to_full*
have *i*: (*c*, *s*) ⇒ *m* ⇓ *emb ps′* (λ_. *0*) **using** *part_to_full′*[*OF c*] **apply** *simp* **done**

have *ii*: *enat m* + *Q′* (*emb ps′* (λ_. *0*)) ≤ *P′ s* **unfolding** *n* **using** *q m*
using *enat_ile* **by** *fastforce*

from *i ii* **have** (∃ *t p*. (*c*, *s*) ⇒ *p* ⇓ *t* ∧ *enat p* + *Q′ t* ≤ *P′ s*) **by** *auto*
} then
show *?thesis* **unfolding** *hoare2_valid_def* **by** *blast*
qed

end

10.2 Relation between the Hoare logics in big-O style

theory *DiscussionO*
imports *SepLogK_Hoare QuantK_Hoare Nielson_Hoare*
begin

10.2.1 Relation Nielson to quantHoare

definition *emN* :: *qassn* ⇒ *Nielson_Hoare.assn2* **where** *emN P* = (λ*l s*. *P s* < ∞)

lemma assumes *s*: $\models_1$ { *emN P′*} *c* { %*s*. (*THE e*. *enat e* = *P′ s* − *Q′* (*THE t*. (∃ *n*. (*c*, *s*) ⇒ *n* ⇓ *t*))) ⇓ *emN Q′* } (**is** $\models_1$ { *?P* } *c* { *?e* ⇓ *?Q* })
shows *quantNielson*: $\models_{2'}$ { *P′* } *c* { *Q′* }
proof −
from *s* **obtain** *k* **where** *k*: *k*>*0* **and** *qd*: ⋀*l s*. *emN P′ l s* ⟹ (∃ *t p*. (*c*, *s*) ⇒ *p* ⇓ *t* ∧ *p* ≤ *k* ∗ *?e s* ∧ *emN Q′ l t*)
unfolding *hoare1_valid_def* **by** *blast*

show *?thesis* **unfolding** *QuantK_Hoare.hoare2o_valid_def*
apply(*rule exI*[**where** *x*=*k*])
apply *safe* **apply** *fact*
proof −
fix *s*
assume *P′*: *P′ s* < ∞

then have $(emN\ P')\ (\lambda\_.\ 0)\ s$ **unfolding** *emN_def* **by** *auto*
with *qd* **obtain** $p\ t$ **where** i: $(c,\ s) \Rightarrow p \Downarrow t$ **and** p: $p \leq k * ?e\ s$ **and** e: $emN\ Q'\ (\lambda\_.\ 0)\ t$
by *blast*
have t: $\downarrow_s (c,\ s) = t$ **using** *bigstepT_the_state*[*OF i*] **by** *auto*

from P' **obtain** *pre* **where** *pre*: $P'\ s = enat\ pre$ **by** *fastforce*
from e **have** $Q'\ t < \infty$ **unfolding** *emN_def* **by** *auto*
then obtain *post* **where** *post*: $Q'\ t = enat\ post$ **by** *fastforce*

have $p > 0$ **using** *i bigstep_progress* **by** *auto*

thm *enat.inject idiff_enat_enat the_equality*
have k: $(THE\ e.\ enat\ e = P'\ s - Q'\ (THE\ t.\ \exists n.\ (c,\ s) \Rightarrow n \Downarrow t)) = pre - post$
unfolding *t pre post* **apply**(*rule the_equality*)
using *idiff_enat_enat* **by** *auto*
with p **have** *ieq*: $p \leq k * (pre - post)$ **by** *auto*
then have $p + k * post \leq k * pre$ **using** ‹*p>0*›
using *diff_mult_distrib2* **by** *auto*
then
have *ii*: $enat\ p + k * Q'\ t \leq k * P'\ s$ **unfolding** *post pre* **by** *simp*

from *i ii* **show** $(\exists t\ p.\ (c,\ s) \Rightarrow p \Downarrow t \wedge enat\ p + k * Q'\ t \leq k * P'\ s)$ **by** *auto*
qed
qed

lemma assumes s: $\models_{2'} \{ \%s\ .\ emb\ (\forall l.\ P\ l\ s) + enat\ (e\ s) \}\ c\ \{ \%s.\ emb\ (\forall l.\ Q\ l\ s) \}$ (**is** $\models_{2'} \{ ?P \}\ c\ \{ ?Q \}$)
and *sP*: $\bigwedge l\ t.\ P\ l\ t \Longrightarrow \forall l.\ P\ l\ t$
and *sQ*: $\bigwedge l\ t.\ Q\ l\ t \Longrightarrow \forall l.\ Q\ l\ t$
shows *NielsonQuant*: $\models_1 \{ P \}\ c\ \{ e \Downarrow Q \}$
proof $-$
from s **obtain** k **where** k: $k{>}0$ **and** *qd*: $\bigwedge s.\ ?P\ s < \infty \longrightarrow (\exists t\ p.\ (c,\ s) \Rightarrow p \Downarrow t \wedge enat\ p + enat\ k * ?Q\ t \leq enat\ k * ?P\ s)$
unfolding *QuantK_Hoare.hoare2o_valid_def* **by** *blast*

show *?thesis* **unfolding** *hoare1_valid_def*
apply(*rule exI*[**where** $x{=}k$])
apply *safe* **apply** *fact*

proof −
fix l s
assume P': $P\ l\ s$
then have aP: $\forall\, l.\ P\ l\ s$ **using** sP **by** *auto*
then have P: $?P\ s < \infty$ **by** *auto*
with qd **obtain** p t **where** i: $(c,\ s) \Rightarrow p \Downarrow t$ **and** p: $enat\ p + enat\ k * ?Q\ t \leq enat\ k * ?P\ s$
by *blast*
have t: $\downarrow_s (c,\ s) = t$ **using** *bigstepT_the_state[OF i]* **by** *auto*

from P **have** Q: $Q\ l\ t$ **using** p k
apply *auto*
by (*metis* (*full_types*) *emb.simps*(*1*) *enat_ord_simps*(*2*) *imult_is_infinity infinity_ileE not_less_zero plus_enat_simps*(*3*))
with sQ **have** $\forall\, l.\ Q\ l\ t$ **by** *auto*
then have $?Q\ t = 0$ **by** *auto*
with p **have** $enat\ p \leq enat\ k * ?P\ s$ **by** *auto*
with aP **have** p': $p \leq k * e\ s$ **by** *auto*

from i Q p' **show** $\exists\, t\ p.\ (c,\ s) \Rightarrow p \Downarrow t \wedge p \leq k * e\ s \wedge Q\ l\ t$ **by** *blast*

qed
qed

10.2.2 Relation SepLogic to quantHoare

definition em :: $qassn \Rightarrow (pstate\_t \Rightarrow bool)$ **where**
$em\ P = (\%(ps,n).\ (\forall\, ex.\ P\ (Partial\_Evaluation.emb\ ps\ ex) \leq enat\ n)\ )$

lemma assumes s: $\models_{3'} \{\ em\ P\}\ c\ \{\ em\ Q\ \}$
shows $\models_{2'} \{\ P\ \}\ c\ \{\ Q\ \}$
proof −
from s **obtain** k **where** k: $0<k$ **and** s': $\bigwedge ps\ n.\ em\ P\ (ps,\ n) \Longrightarrow (\exists\, ps'\ ps''\ m\ e\ e'.\ (c,\ ps) \Rightarrow_A m \Downarrow ps' + ps'' \wedge ps'\ \#\#\ ps'' \wedge k * n = k * e + e' + m \wedge em\ Q\ (ps',\ e))$ **unfolding** *hoare3o_valid_def* **by** *auto*
{
fix s
assume P: $P\ s < \infty$
then obtain n **where** n: $P\ s = enat\ n$
by *fastforce*
with P **have** $em\ P\ (part\ s,\ n)$ **unfolding** *em_def* **by** *auto*
with s' **obtain** $ps'\ ps''\ m\ e\ e'$ **where** c: $(c,\ part\ s) \Rightarrow_A m \Downarrow ps' + ps''$ **and** $orth$: $ps'\ \#\#\ ps''$
and m: $k * n = k * e + e' + m$ **and** Q: $em\ Q\ (ps',\ e)$ **by** *blast*

from *Q* **have** *q*: *Q* (*Partial_Evaluation.emb ps′* (*Partial_Evaluation.emb ps″* (λ_. *0*))) ≤ *enat* (*e*) **unfolding** *em_def* **by** *auto*

have *z*: (*Partial_Evaluation.emb ps′* (*Partial_Evaluation.emb ps″* (λ_. *0*))) = (*Partial_Evaluation.emb* (*ps′*+*ps″*) (λ_. *0*))
unfolding *Partial_Evaluation.emb_def* **apply** (*auto simp*: *plus_fun_def*)
apply (*rule ext*) **subgoal for** *v* **apply** (*cases ps′ v*) **apply** *auto* **using** *orth* **by** (*auto simp*: *sep_disj_fun_def domain_conv*) **done**

from *q z* **have** *q*: *enat k* ∗ *Q* (*Partial_Evaluation.emb* (*ps′*+*ps″*) (λ_. *0*)) ≤ *enat k* ∗ *enat e* **using** *k*
by (*metis i0_lb mult_left_mono*)

have *i*: (*c*, *s*) ⇒ *m* ⇓ (*Partial_Evaluation.emb* (*ps′*+*ps″*) (λ_. *0*)) **using** *part_to_full′*[*OF c*] **by** *simp*

have *ii*: *enat m* + *enat k* ∗ *Q* (*Partial_Evaluation.emb* (*ps′*+*ps″*) (λ_. *0*)) ≤ *enat k* ∗ *P s* **unfolding** *n* **using** *q m*
using *enat_ile* **by** *fastforce*

from *i ii* **have** (∃ *t p*. (*c*, *s*) ⇒ *p* ⇓ *t* ∧ *enat p* + *enat k* ∗ *Q t* ≤ *enat k* ∗ *P s*) **by** *auto*
} **note** *B*=*this*
show *?thesis* **unfolding** *QuantK_Hoare.hoare2o_valid_def*
apply(*rule exI*[**where** *x*=*k*], *safe*) **apply** *fact*
apply (*fact B*) **done**
qed

definition *embe* :: (*pstate_t* ⇒ *bool*) ⇒ *qassn* **where**
embe P = (%*s*. *Inf* {*enat n*|*n*. *P* (*part s*, *n*)})

lemma assumes *s*: $\models_{2'}$ { *embe P* } *c* { *embe Q* } **and** *full*: ⋀*ps n*. *P* (*ps*,*n*) ⟹ *dom ps* = *UNIV*
shows $\models_{3'}$ { *P*} *c* { *Q* }
proof −
from *s* **obtain** *k* **where** *k*: *k*>*0* **and** *s*: ⋀*s*. *embe P s* < ∞ ⟶ (∃ *t p*. (*c*, *s*) ⇒ *p* ⇓ *t* ∧ *enat p* + *enat k* ∗ *embe Q t* ≤ *enat k* ∗ *embe P s*)
unfolding *QuantK_Hoare.hoare2o_valid_def* **by** *auto*

{ **fix** *ps n*
let *?s* = (*Partial_Evaluation.emb ps* (λ_. *0*))
assume *P*: *P* (*ps*, *n*)

with *full* **have** *dom ps* = *UNIV* **by** *auto*
then have *ps*: *part ?s* = *ps* **by** *simp*
from *P* **have** l': $(\{enat\ n\ |n.\ P\ (ps,\ n)\} = \{\}) =$ *False* **by** *auto*
have *t*: *embe P ?s* $< \infty$ **unfolding** *embe_def Inf_enat_def ps* l'
apply(*rule ccontr*) **using** l' **apply** *auto*
by (*metis* (*mono_tags*, *lifting*) *Least_le infinity_ileE*)
with *s* **obtain** *t p* **where** *c*: $(c,\ ?s) \Rightarrow p \Downarrow t$ **and** *ineq*: $enat\ p + enat\ k * embe\ Q\ t \leq enat\ k * embe\ P\ ?s$ **by** *blast*
from *t* **obtain** *z* **where** *z*: *embe P ?s* = *enat z*
using *less_infinityE* **by** *blast*
with *ineq* **obtain** *y* **where** *y*: *embe Q t* = *enat y*
using *k* **by** *fastforce*
then have *l*: *embe Q t* $< \infty$ **by** *auto*
then have *zz*: $(\{enat\ n|n.\ Q\ (part\ t,\ n)\} = \{\}) = False$ **unfolding** *embe_def Inf_enat_def* **apply** *safe* **by** *simp*
from *y* **have** *Q* (*part t*, *y*) **unfolding** *embe_def zz Inf_enat_def* **apply** *auto*
using *zz* **apply** *auto* **by** (*smt Collect_empty_eq LeastI enat.inject*)

from *full_to_part*[*OF c*] *ps* **have** c': $(c,\ ps) \Rightarrow_A p \Downarrow part\ t$ **by** *auto*

have $\bigwedge P\ n.\ P\ (n::nat) \Longrightarrow (LEAST\ n.\ P\ n) \leq n$ **apply**(*rule Least_le*) **by** *auto*

from *z P* **have** *zn*: $z \leq n$ **unfolding** *embe_def ps* **unfolding** *embe_def Inf_enat_def* l'
apply *auto*
by (*metis* (*mono_tags*, *lifting*) *Least_le enat_ord_simps*(*1*))

from *ineq z y* **have** $enat\ p + enat\ k * y \leq enat\ k * z$ **by** *auto*
then have $p + k * y \leq k * z$ **by** *auto*
also have $\ldots \leq k * n$ **using** *zn k* **by** *simp*
finally obtain e' **where** $k * n = k * y + e' + p$ **using** *k* **by** (*metis add.assoc add.commute le_iff_add*)

have $\exists ps'\ ps''\ m\ e\ e'.\ (c,\ ps) \Rightarrow_A m \Downarrow ps' + ps'' \wedge ps'\ \#\#\ ps'' \wedge k * n = k * e + e' + m \wedge Q\ (ps',\ e)$
apply(*rule exI*[**where** *x*=*part t*])
apply(*rule exI*[**where** *x*=*0*])
apply(*rule exI*[**where** *x*=*p*])
apply(*rule exI*[**where** *x*=*y*])
apply(*rule exI*[**where** *x*=e']) **apply** *auto* **by** *fact*+
}

show *?thesis* **unfolding** *hoare3o_valid_def* **apply**(*rule exI*[**where** *x*=*k*], *safe*)
apply *fact* **by** *fact*
qed

10.3 A General Validity Predicate with Time

definition *valid* **where**
$valid\ P\ c\ Q\ n = (\forall s.\ P\ s \longrightarrow (\exists s'\ m.\ (c,\ s) \Rightarrow m \Downarrow s' \wedge m \leq n \wedge Q\ s'))$

definition *validk* **where**
$validk\ P\ c\ Q\ n = (\exists k{>}0.\ (\forall s.\ P\ s \longrightarrow (\exists s'\ m.\ (c,\ s) \Rightarrow m \Downarrow s' \wedge m \leq k * n \wedge Q\ s')))$

lemma $validk\ P\ c\ Q\ n = (\exists k{>}0.\ valid\ P\ c\ Q\ (k{*}n))$
unfolding *valid_def validk_def* **by** *simp*

10.3.1 Relation between valid predicate and Quantitative Hoare Logic

lemma $\models_{2'} \{\%s.\ \ emb\ (P\ s)\ \ + enat\ n\}\ c\ \{\ \lambda s.\ \ emb\ (Q\ s)\ \ \} \Longrightarrow \exists k{>}0.\ valid\ P\ c\ Q\ (k{*}n)$
proof −
assume *valid*: $\models_{2'} \{\lambda s.\ \uparrow (P\ s) + enat\ n\}\ c\ \{\lambda s.\ \uparrow (Q\ s)\}$
then obtain *k* **where** *val*: $\bigwedge s.\ \uparrow (P\ s) + enat\ n < \infty \Longrightarrow (\exists t\ p.\ (c,\ s) \Rightarrow p \Downarrow t \wedge enat\ p + enat\ k * \uparrow (Q\ t) \leq enat\ k * (\uparrow (P\ s) + enat\ n))$
and *k*: $k{>}0$ **unfolding** *QuantK_Hoare.hoare2o_valid_def* **by** *blast*
{
fix *s*
assume *Ps*: $P\ s$
then have $\uparrow (P\ s) + enat\ n < \infty$ **by** *auto*
with *val* **obtain** $t\ m$ **where**
c: $(c,\ s) \Rightarrow m \Downarrow t$ **and** $enat\ m + k * \uparrow (Q\ t) \leq k * (\uparrow (P\ s) + enat\ n)$ **by** *blast*

then have $m \leq k * n \wedge Q\ t$ **using** *k*
using *Ps add.commute add.right_neutral emb.simps(1) emb.simps(2) enat_ord_simps(1) infinity_ileE plus_enat_simps(3)*
by (*metis* (*full_types*) *mult_zero_right not_gr_zero times_enat_simps(1) times_enat_simps(4)*)

with *c*

have $(\exists s'\ m.\ (c,\ s) \Rightarrow m \Downarrow s' \wedge m \leq\ \ k * n \wedge Q\ s')$ **by** *blast*
} **note** *bla=this*
show $\exists k>0.\ valid\ P\ c\ Q\ (k*n)$ **unfolding** *valid_def* **apply**(*rule exI*[**where** $x=k$]) **using** *bla k* **by** *auto*
qed

lemma *valid_quantHoare*: $\exists k>0.\ valid\ P\ c\ Q\ (k*n) \Longrightarrow \models_{2'} \{\%s.\ emb\ (P\ s)\ +\ enat\ n\}\ c\ \{\ \lambda s.\ emb\ (Q\ s)\ \ \}$
proof −
assume $\exists k>0.\ valid\ P\ c\ Q\ (k*n)$
then obtain k **where** *valid*: $valid\ P\ c\ Q\ (k*n)$ **and** k: $k>0$ **by** *blast*
{
fix s
assume $(\%s.\ emb\ (P\ s)\ +\ enat\ n)\ s < \infty$
then have *Ps*: $P\ s$ **apply** *auto*
by (*metis emb.elims enat.distinct*(*2*) *enat.simps*(*5*) *enat_defs*(*4*))
with *valid*[*unfolded valid_def*] **obtain** $t\ m$ **where**
c: $(c,\ s) \Rightarrow m \Downarrow t$ **and** $m \leq k * n\ Q\ t$ **by** *blast*
then have $enat\ m +\ \ k * \uparrow (Q\ t) \leq\ \ k * (\uparrow (P\ s) + enat\ n)$ **using** *Ps* **by** *simp*
with c
have $(\exists s'\ m.\ (c,\ s) \Rightarrow m \Downarrow s' \wedge enat\ m + enat\ k * \uparrow (Q\ s') \leq enat\ k * (\uparrow (P\ s) + enat\ n))$ **by** *blast*
} **note** *funk=this*
show $\models_{2'} \{\%s.\ \ emb\ (P\ s)\ +\ enat\ n\}\ c\ \{\ \lambda s.\ emb\ (Q\ s)\ \ \}$ **unfolding** *QuantK_Hoare.hoare2o_valid_def*
apply(*rule exI*[**where** $x=k$]) **using** *funk k* **by** *auto*
qed

10.3.2 Relation between valid predicate and Hoare Logic based on Separation Logic

definition *embP2 P* $= (\%(ps,n).\ \forall s.\ \ P\ (Partial\_Evaluation.emb\ ps\ s) \wedge n = 0)$
definition *embP3 P* $= (\%(ps,n).\ dom\ ps = UNIV \wedge (\forall s.\ \ P\ (Partial\_Evaluation.emb\ ps\ s)) \wedge n = 0)$

lemma *emp*: $a + Map.empty = a$
by (*simp add*: *plus_fun_conv*)

lemma *oneway*: $\models_{3'} \{embP3\ P\ {**}\ \$n\}\ c\ \{embP2\ Q\} \Longrightarrow validk\ P\ c\ Q\ n$
proof −
assume *partial_true*: $\models_{3'} \{embP3\ P\ {**}\ \$n\}\ c\ \{embP2\ Q\}$

from *partial_true*[*unfolded hoare3o_valid_def*] **obtain** *k* **where** *k*: *k>0*
and
q : $\forall$ *ps na*. (*embP3 P* $\wedge *$ \$ *n*) (*ps*, *na*) $\longrightarrow$
($\exists$ ps' ps'' *m e* e'. (*c*, *ps*) $\Rightarrow_A$ *m* $\Downarrow$ $ps' + ps'' \wedge ps' \ \#\# \ ps'' \wedge$
$k * na = k * e + e' + m \wedge$ *embP2 Q* (ps', *e*)) **by** *blast*
{ fix *s*
assume *P s*
then have *g*: (*embP3 P* $\wedge *$ \$ *n*) (*part s*, *n*)
unfolding *embP3_def dollar_def sep_conj_def* **by** *auto*
from *q g*
obtain ps' ps'' *m e* e' **where** *pbig*: (*c*, *part s*) $\Rightarrow_A$ *m* $\Downarrow$ $ps' + ps''$ **and**
orth: $ps' \ \#\# \ ps''$
and *ii*: $k * n = k * e + e' + m$ **and** *erg*: *embP2 Q* (ps', *e*) **by** *blast*

have ii': $m \leq k * n$ **using** *ii* **by** *auto*

from *part_to_full'*[*OF pbig*] **have** *i*: (*c*, *s*) $\Rightarrow$ *m* $\Downarrow$ *Partial_Evaluation.emb*
($ps' + ps''$) *s* **by** *simp*

from *erg* **have** *z2*: $\bigwedge$*s*. *Q* (*Partial_Evaluation.emb* ps' *s*) **unfolding**
embP2_def **by** *auto*
have *Partial_Evaluation.emb* ($ps' + ps''$) *s* = *Partial_Evaluation.emb*
($ps'' + ps'$) *s*
using *orth* **by** (*simp add*: *sep_add_commute*)
also have *Partial_Evaluation.emb* ($ps'' + ps'$) *s* = *Partial_Evaluation.emb*
(ps') (*Partial_Evaluation.emb* (ps'') *s*)
apply *rule*
unfolding *emb_def plus_fun_conv map_add_def*
by (*metis option.case_eq_if option.simps*(*5*))
finally have *z*: *Partial_Evaluation.emb* ($ps' + ps''$) *s* = *Partial_Evaluation.emb*
(ps') (*Partial_Evaluation.emb* (ps'') *s*) **.**
have *iii*: *Q* (*Partial_Evaluation.emb* ($ps' + ps''$) *s*) **unfolding** *z* **apply**
(*fact*) **.**

from *i* ii' *iii*
have $\exists$ s' *m*. (*c*, *s*) $\Rightarrow$ *m* $\Downarrow$ $s' \wedge m \leq k * n \wedge Q \ s'$ **by** *auto*
}
with *k* **show** *validk P c Q n* **unfolding** *validk_def* **by** *blast*
qed

lemma *theother*: *validk P c Q n* $\Longrightarrow$ $\models_{3'}$ {*embP3 P* $**$ \$*n*} *c* {*embP2 Q* }
proof −
assume *valid*: *validk P c Q n*

then obtain *k* **where** *k* : *k*>*0* **and** *v*: ($\forall$ *s*. *P s* $\longrightarrow$ ($\exists$ *s'* *m*. (*c*, *s*) $\Rightarrow$ *m* $\Downarrow$ *s'* $\wedge$ *m* $\leq$ *k* $*$ *n* $\wedge$ *Q s'*))
unfolding *validk_def* **by** *blast*

{ fix *ps na*
assume *an*: (*embP3 P* $\wedge*$ \$ *n*) (*ps*, *na*)
have *dom*: *dom ps* = *UNIV* **and** *Pps*: $\bigwedge$*s*. *P* (*Partial_Evaluation.emb ps s*) **and** *nan*: *na* = *n* **using** *an* **unfolding** *sep_conj_def*
by (*auto simp*: *embP3_def dollar_def*)

from *v Pps*
obtain *s' m* **where** *big*: (*c*, (*Partial_Evaluation.emb ps* (%_. *0*))) $\Rightarrow$ *m* $\Downarrow$ *s'* **and** *ii*: *m* $\leq$ *k* $*$ *n* **and** *erg*: *Q s'* **by** *blast*

have *part* (*Partial_Evaluation.emb ps* (λ_. *0*)) = *ps* **using** *dom* **by** *simp*
with *full_to_part*[*OF big*] **have** *i*: (*c*, *ps*) $\Rightarrow_A$ *m* $\Downarrow$ *part s'* **by** *auto*

have *iii*: *embP2 Q* (*part s'*, *0*)
unfolding *embP2_def* **apply** *auto* **by** *fact*

have *k* $*$ *na* = *k* $*$ *n* $-$ *m* + *m* **using** *ii k nan* **by** *simp*

have ($\exists$ *ps' ps'' m e e'*. (*c*, *ps*) $\Rightarrow_A$ *m* $\Downarrow$ *ps'* + *ps''* $\wedge$ *ps'* ## *ps''* $\wedge$ *k* $*$ *na* = *k* $*$ *e* + *e'* + *m* $\wedge$ *embP2 Q* (*ps'*, *e*))
apply(*rule exI*[**where** *x*=*part s'*])
apply(*rule exI*[**where** *x*=*0*])
apply(*rule exI*[**where** *x*=*m*])
apply(*rule exI*[**where** *x*=*0*])
apply(*rule exI*[**where** *x*=*k* $*$ *n* $-$ *m*]) **apply** *auto*
by *fact*+
}
with *k* **show** $\models_{3'}$ {*embP3 P* $**$ \$*n*} *c* {*embP2 Q* } **unfolding** *hoare3o_valid_def* **by** *blast*
qed

lemma *validk P c Q n* $\longleftrightarrow$ $\models_{3'}$ {*embP3 P* $**$ \$*n*} *c* {*embP2 Q* }
using *oneway* **and** *theother* **by** *metis*

end
theory *Hoare_Time* **imports**

Nielson_Hoare
Nielson_VCG
Nielson_VCGi
Nielson_VCGi_complete
Nielson_Examples
Nielson_Sqrt

Quant_Hoare
Quant_VCG
Quant_Examples

QuantK_Hoare
QuantK_VCG
QuantK_Examples
QuantK_Sqrt

SepLog_Hoare
SepLog_Examples
SepLogK_Hoare
SepLogK_VCG

Discussion
DiscussionO

begin end

References

[HN18] Maximilian Paul Louis Haslbeck and Tobias Nipkow. Hoare logics for time bounds. In M. Huisman and D. Beyer, editors, *Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2018)*, LNCS. Springer, 2018. To appear.