Hoare Logics for Time Bounds

Maximilian P. L. Haslbeck Tobias Nipkow*

August 7, 2022

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 ∧ *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″)))
<″x″ := 3, ″y″ := 1>

end

2 IMP — A Simple Imperative Language

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

datatype
com = *SKIP*
| *Assign vname aexp* (_ ::= _ [1000, 61] 61)
| *Seq com com* (_;;/ _ [60, 61] 60)
| *If bexp com com* ((*IF* _/ *THEN* _/ *ELSE* _) [0, 0, 61] 61)
| *While bexp com* ((*WHILE* _/ *DO* _) [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* × *state* ⇒ *state* ⇒ *bool* (**infix** ⇒ 55)
where
Skip: (*SKIP*,s) ⇒ s |
Assign: (x ::= a,s) ⇒ s(x := *aval* a s) |
Seq: ⟦ (c_1,s_1) ⇒ s_2; (c_2,s_2) ⇒ s_3 ⟧ ⟹ (c_1;;c_2, s_1) ⇒ s_3 |
IfTrue: ⟦ *bval* b s; (c_1,s) ⇒ t ⟧ ⟹ (*IF* b *THEN* c_1 *ELSE* c_2, s) ⇒ t |
IfFalse: ⟦ ¬*bval* b s; (c_2,s) ⇒ t ⟧ ⟹ (*IF* b *THEN* c_1 *ELSE* c_2, s) ⇒ t |
WhileFalse: ¬*bval* b s ⟹ (*WHILE* b *DO* c,s) ⇒ s |
WhileTrue:
⟦ *bval* b s_1; (c,s_1) ⇒ s_2; (*WHILE* b *DO* c, s_2) ⇒ s_3 ⟧
⟹ (*WHILE* b *DO* c, s_1) ⇒ s_3

We want to execute the big-step rules:

code_pred *big_step* ⟨*proof*⟩

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

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

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*

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

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 $(IF\ b\ THEN\ SKIP\ ELSE\ SKIP,\ s) \Rightarrow t \Longrightarrow t = s$
⟨*proof*⟩

Rule inversion by hand via the "cases" method:

lemma assumes (*IF b THEN SKIP ELSE SKIP*, *s*) ⇒ *t*
shows *t* = *s*
⟨*proof*⟩

lemma *assign_simp*:
(*x* ::= *a*,*s*) ⇒ *s*′ ⟷ (*s*′ = *s*(*x* := *aval a s*))
⟨*proof*⟩

An example combining rule inversion and derivations

lemma *Seq_assoc*:
(*c1*;; *c2*;; *c3*, *s*) ⇒ *s*′ ⟷ (*c1*;; (*c2*;; *c3*), *s*) ⇒ *s*′
⟨*proof*⟩

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: ∼ 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*) ∼ (*IF b THEN c*;; *WHILE b DO c ELSE SKIP*) (**is** *?w* ∼ *?iw*)
⟨*proof*⟩

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*)
⟨*proof*⟩

lemma *triv_if*:
(*IF b THEN c ELSE c*) ∼ *c*
⟨*proof*⟩

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*))
⟨*proof*⟩

lemma *sim_while_cong_aux*:
(*WHILE b DO c*,*s*) ⇒ *t* ⟹ *c* ∼ *c*′ ⟹ (*WHILE b DO c*′,*s*) ⇒ *t*
⟨*proof*⟩

lemma *sim_while_cong*: *c* ∼ *c*′ ⟹ *WHILE b DO c* ∼ *WHILE b DO c*′
⟨*proof*⟩

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* ∼ *c* ⟨*proof*⟩
lemma *sim_sym*: (*c* ∼ *c*′) = (*c*′ ∼ *c*) ⟨*proof*⟩
lemma *sim_trans*: *c* ∼ *c*′ ⟹ *c*′ ∼ *c*″ ⟹ *c* ∼ *c*″ ⟨*proof*⟩

2.4 Execution is deterministic

This proof is automatic.

theorem *big_step_determ*: ⟦ (*c*,*s*) ⇒ *t*; (*c*,*s*) ⇒ *u* ⟧ ⟹ *u* = *t*
⟨*proof*⟩

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*) ⇒ *t* ⟹ (*c*,*s*) ⇒ *t*′ ⟹ *t*′ = *t*
⟨*proof*⟩

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* × *state* ⇒ *nat* ⇒ *state* ⇒ *bool* (_ ⇒ _ ⇓ _ 55)
where
Skip: (*SKIP*,*s*) ⇒ *Suc 0* ⇓ *s* |
Assign: (*x* ::= *a*,*s*) ⇒ *Suc 0* ⇓ *s*(*x* := *aval a s*) |

Seq: ⟦ (*c1*,*s1*) ⇒ *x* ⇓ *s2*; (*c2*,*s2*) ⇒ *y* ⇓ *s3* ; *z*=*x*+*y* ⟧ ⟹ (*c1*;;*c2*, *s1*) ⇒ *z* ⇓ *s3* |
IfTrue: ⟦ *bval b s*; (*c1*,*s*) ⇒ *x* ⇓ *t*; *y*=*x*+*1* ⟧ ⟹ (*IF b THEN c1 ELSE c2*, *s*) ⇒ *y* ⇓ *t* |
IfFalse: ⟦ ¬*bval b s*; (*c2*,*s*) ⇒ *x* ⇓ *t*; *y*=*x*+*1* ⟧ ⟹ (*IF b THEN c1 ELSE c2*, *s*) ⇒ *y* ⇓ *t* |
WhileFalse: ⟦ ¬*bval b s* ⟧ ⟹ (*WHILE b DO c*,*s*) ⇒ *Suc 0* ⇓ *s* |
WhileTrue: ⟦ *bval b s1*; (*c*,*s1*) ⇒ *x* ⇓ *s2*; (*WHILE b DO c*, *s2*) ⇒ *y* ⇓ *s3*; *1*+*x*+*y*=*z* ⟧
⟹ (*WHILE b DO c*, *s1*) ⇒ *z* ⇓ *s3*

We want to execute the big-step rules:

code_pred *big_step_t* ⟨*proof*⟩

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

values {(*t*, *x*). (*SKIP*, λ_. *0*) ⇒ *x* ⇓ *t*}

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

values {*map t* [*''x''*] |*t x*. (*SKIP*, <*''x''* := *42*>) ⇒ *x* ⇓ *t*}

values {*map t* [*''x''*] |*t x*. (*''x''* ::= *N 2*, <*''x''* := *42*>) ⇒ *x* ⇓ *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*>) ⇒ *x* ⇓ *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*) ⇒ *x* ⇓ *t* ? That *s* = *t*. This is how we can automatically prove it:

inductive_cases *Skip_tE*[*elim*!]: (*SKIP*,*s*) ⇒ *x* ⇓ *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*) ⇒ *p* ⇓ *t*
thm *Assign_tE*

inductive_cases *Seq_tE*[*elim*!]: (*c1*;;*c2*,*s1*) ⇒ *p* ⇓ *s3*
thm *Seq_tE*
inductive_cases *If_tE*[*elim*!]: (*IF b THEN c1 ELSE c2*,*s*) ⇒ *x* ⇓ *t*
thm *If_tE*

inductive_cases *While_tE*[*elim*]: (*WHILE b DO c*,*s*) ⇒ *x* ⇓ *t*
thm *While_tE*

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

An automatic example:

lemma (*IF b THEN SKIP ELSE SKIP*, *s*) ⇒ *x* ⇓ *t* ⟹ *t* = *s*
⟨*proof*⟩

Rule inversion by hand via the "cases" method:

lemma assumes (*IF b THEN SKIP ELSE SKIP*, *s*) ⇒ *x* ⇓ *t*
shows *t* = *s*
⟨*proof*⟩

lemma *assign_t_simp*:
(*x* ::= *a*,*s*) ⇒ *Suc 0* ⇓ *s′* ⟷ (*s′* = *s*(*x* := *aval a s*))
⟨*proof*⟩

An example combining rule inversion and derivations

lemma *Seq_t_assoc*:
((*c1*;; *c2*;; *c3*, *s*) ⇒ *p* ⇓ *s′*) ⟷ ((*c1*;; (*c2*;; *c3*), *s*) ⇒ *p* ⇓ *s′*)
⟨*proof*⟩

3.3 Relation to Big-Step Semantics

lemma (∃ *p*. ((*c*, *s*) ⇒ *p* ⇓ *s′*)) = ((*c*, *s*) ⇒ *s′*)
⟨*proof*⟩

3.4 Execution is deterministic

This proof is automatic.

theorem *big_step_t_determ*: ⟦ (*c*,*s*) ⇒ *p* ⇓ *t*; (*c*,*s*) ⇒ *q* ⇓ *u* ⟧ ⟹ *u* = *t*
⟨*proof*⟩

theorem *big_step_t_determ2*: ⟦ (*c*,*s*) ⇒ *p* ⇓ *t*; (*c*,*s*) ⇒ *q* ⇓ *u* ⟧ ⟹ (*u* = *t* ∧ *p*=*q*)
⟨*proof*⟩

lemma *bigstep_det*: $(c1,\ s) \Rightarrow p1 \Downarrow t1 \Longrightarrow (c1,\ s) \Rightarrow p \Downarrow t \Longrightarrow p1{=}p \wedge t1{=}t$
⟨*proof*⟩

lemma *bigstep_progress*: $(c,\ s) \Rightarrow p \Downarrow t \Longrightarrow p > 0$
⟨*proof*⟩

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*: $(c,\ s) \Rightarrow t \Downarrow s' \Longrightarrow \downarrow_t(c,\ s) = t$
⟨*proof*⟩

lemma *bigstepT_the_state*: $(c,\ s) \Rightarrow t \Downarrow s' \Longrightarrow \downarrow_s(c,\ s) = s'$
⟨*proof*⟩

lemma *SKIPnot*: $(\neg\ (SKIP,\ s) \Rightarrow p \Downarrow t) = (s{\neq}t \vee p{\neq}Suc\ 0)$ ⟨*proof*⟩

lemma *SKIPp*: $\downarrow_t(SKIP,s) = Suc\ 0$
⟨*proof*⟩

lemma *SKIPt*: $\downarrow_s(SKIP,s) = s$
⟨*proof*⟩

lemma *ASSp*: $(THE\ p.\ Ex\ (big\_step\_t\ (x ::= e,\ s)\ p)) = Suc\ 0$
⟨*proof*⟩

lemma *ASSt*: $(THE\ t.\ \exists\, p.\ (x ::= e,\ s) \Rightarrow p \Downarrow t) = s(x := aval\ e\ s)$
⟨*proof*⟩

lemma *ASSnot*: $(\ \neg\ (x ::= e,\ s) \Rightarrow p \Downarrow t\ ) = (p{\neq}Suc\ 0 \vee t{\neq}s(x := aval\ e\ s))$
⟨*proof*⟩

lemma *If_THE_True*: *Suc* (*THE n*. ∃ *a*. (*c1*, *s*) ⇒ *n* ⇓ *a*) = (*THE n*. ∃ *a*. (*IF b THEN c1 ELSE c2*, *s*) ⇒ *n* ⇓ *a*)
if *T*: *bval b s* **and** *c1_t*: *terminates* (*c1*,*s*) **for** *s l*
⟨*proof*⟩

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*⟩

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*
⟨*proof*⟩

lemma *support_impl*: *support* (λ*l s*. *P s* ⟶ *Q l s*) ⊆ *support Q*
⟨*proof*⟩

lemma *support_exist*: *support* (λ*l s*. ∃ *z*::*nat*. *Q z l s*) ⊆ (*UN n*. *support* (*Q n*))
⟨*proof*⟩

lemma *support_all*: *support* (λ*l s*. ∀ *z*. *Q z l s*) ⊆ (*UN n*. *support* (*Q n*))

⟨*proof*⟩

lemma *support_single*: *support* (λ*l s*. *P* (*l a*) *s*) ⊆ {*a*}
⟨*proof*⟩

lemma *support_inv*: ⋀*P*. *support* (λ*l s*. *P s*) = {}
⟨*proof*⟩

lemma *assn2_lupd*: *x* ∉ *support Q* ⟹ *Q* (*l*(*x*:=*n*)) = *Q l*
⟨*proof*⟩

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*
(⊨₁ {(*1*_)}/ (_)/ { _ ⇓(*1*_)} *50*) **where**
⊨₁ {*P*} *c* {*q* ⇓ *Q*} ⟷ (∃ *k*>*0*. (∀ *l s*. *P l s* ⟶ (∃ *t p*. ((*c*,*s*) ⇒ *p* ⇓ *t*) ∧ *p* ≤ *k* ∗ (*q s*) ∧ *Q l t*)))

4.3 Hoare rules

inductive
hoare1 :: *assn2* ⇒ *com* ⇒ *tbd* ⇒ *assn2* ⇒ *bool* (⊢₁ ({(*1*_)}/ (_)/ { _ ⇓ (*1*_)}) *50*)
where

Skip: ⊢₁ {*P*} *SKIP* { (%*s*. *Suc 0*) ⇓ *P*} |

Assign: ⊢₁ {λ*l s*. *P l* (*s*[*a*/*x*])} *x*::=*a* { (%*s*. *Suc 0*) ⇓ *P*} |

If: ⟦ ⊢₁ {λ*l s*. *P l s* ∧ *bval b s*} *c*₁ { *e1* ⇓ *Q*};
⊢₁ {λ*l s*. *P l s* ∧ ¬ *bval b s*} *c*₂ { *e1* ⇓ *Q*} ⟧
⟹ ⊢₁ {*P*} *IF b THEN c*₁ *ELSE c*₂ { (λ*s*. *e1 s* + *Suc 0*) ⇓ *Q*} |

Seq: ⟦ ⊢₁ { (%*l s*. *P*₁ *l s* ∧ *l x* = *e2*′ *s*) } *c*₁ { *e1* ⇓ (%*l s*. *P*₂ *l s* ∧ *e2 s* ≤ *l x*)};
⊢₁ {*P*₂} *c*₂ { *e2* ⇓ *P*₃}; *x* ∉ *support P*₁; *x* ∉ *support P*₂;
⋀*l s*. *P*₁ *l s* ⟹ *e1 s* + *e2*′ *s* ≤ *e s*⟧
⟹ ⊢₁ {*P*₁} *c*₁;;*c*₂ { *e* ⇓ *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*: $[\![\exists k{>}0.\ \forall l\ s.\ P'\ l\ s \longrightarrow (P\ l\ s \wedge (\ e'\ s \leq\ k * (e\ s)));$
$\vdash_1 \{P\}c\{\ e' \Downarrow\ Q\};\ \forall l\ s.\ Q\ l\ s \longrightarrow Q'\ l\ s\ ]\!] \Longrightarrow$
$\vdash_1 \{P'\}c\{e \Downarrow Q'\}$
⟨*proof*⟩

lemma *If2*: $[\![\vdash_1 \{\lambda l\ s.\ P\ l\ s \wedge bval\ b\ s\}\ c_1\ \{\ e \Downarrow Q\};\ \vdash_1 \{\lambda l\ s.\ P\ l\ s \wedge \neg\ bval\ b\ s\}\ c_2\ \{\ e \Downarrow Q\};$
$\bigwedge l\ s.\ P\ l\ s \Longrightarrow e\ s + 1 = e'\ s\ ]\!]$
$\Longrightarrow \vdash_1 \{P\}\ IF\ b\ THEN\ c_1\ ELSE\ c_2\ \{\ e' \Downarrow Q\}$
⟨*proof*⟩

lemma *strengthen_pre*:
$[\![\forall l\ s.\ P'\ l\ s \longrightarrow P\ l\ s\ ;\ \vdash_1 \{P\}\ c\ \{\ e \Downarrow Q\}\ ]\!] \Longrightarrow \vdash_1 \{P'\}\ c\ \{\ e \Downarrow Q\}$
⟨*proof*⟩

lemma *weaken_post*:
$[\![\vdash_1 \{P\}\ c\ \{e \Downarrow Q\};\ \forall l\ s.\ Q\ l\ s \longrightarrow Q'\ l\ s\ ]\!] \Longrightarrow\ \vdash_1 \{P\}\ c\ \{e \Downarrow Q'\}$
⟨*proof*⟩

lemma *ub_cost*:
$[\![(\exists k{>}0.\ \forall l\ s.\ P\ l\ s \longrightarrow e'\ s \leq k * (e\ s));\ \vdash_1 \{P\}\ c\ \{e' \Downarrow Q\}\ ]\!] \Longrightarrow\ \vdash_1 \{P\}\ c\ \{e \Downarrow Q\}$
⟨*proof*⟩

lemma *Assign'*: $\forall l\ s.\ P\ l\ s \longrightarrow Q\ l\ (s[a/x]) \Longrightarrow (\vdash_1 \{P\}\ x ::= a\ \{\ (\%s.\ 1) \Downarrow Q\})$
⟨*proof*⟩

4.4 Soundness

The soundness theorem:

theorem *hoare1_sound*: $\vdash_1 \{P\}c\{e \Downarrow Q\} \implies \models_1 \{P\}c\{e \Downarrow Q\}$
⟨*proof*⟩

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* ($wp_1\ c\ Q$) ⊆ *support Q*
⟨*proof*⟩

lemma *wp1_terminates*: $wp_1\ c\ Q\ l\ s \Longrightarrow \downarrow (c,\ s)$ ⟨*proof*⟩

lemma *wp1_SKIP*[*simp*]: wp_1 *SKIP Q* = *Q* ⟨*proof*⟩

lemma *wp1_Assign*[*simp*]: $wp_1\ (x ::= e)\ Q = (\lambda l\ s.\ Q\ l\ (s(x := aval\ e\ s)))$
⟨*proof*⟩

lemma *wp1_Seq*[*simp*]: $wp_1\ (c_1;;c_2)\ Q = wp_1\ c_1\ (wp_1\ c_2\ Q)$ ⟨*proof*⟩

lemma *wp1_If*[*simp*]: wp_1 (*IF b THEN* c_1 *ELSE* c_2) *Q* = ($\lambda l\ s.\ wp_1$ (*if bval b s then* c_1 *else* c_2) *Q l s*) ⟨*proof*⟩

definition *prec c E* == %*s*. *E* (*THE t*. (∃ *p*. (*c*,*s*) ⇒ *p* ⇓ *t*))

lemma *wp1_prec_Seq_correct*: wp_1 (*c1*;;*c2*) *Q l s* $\Longrightarrow \downarrow_t$ (*c1*, *s*) + *prec c1* ($\lambda s.\ \downarrow_t$ (*c2*, *s*)) *s* ≤ $\downarrow_t$ (*c1*;; *c2*, *s*)
⟨*proof*⟩

abbreviation *new Q* ≡ *SOME x*. *x* ∉ *support Q*

lemma *bigstep_det*: (*c1*, *s*) ⇒ *p1* ⇓ *t1* ⟹ (*c1*, *s*) ⇒ *p* ⇓ *t* ⟹ *p1*=*p* ∧ *t1*=*t*
⟨*proof*⟩

lemma *bigstepT_the_cost*: (*c*, *s*) ⇒ *P* ⇓ *T* ⟹ (*THE n*. ∃ *a*. (*c*, *s*) ⇒ *n* ⇓ *a*) = *P*
⟨*proof*⟩

lemma *bigstepT_the_state*: $(c,\ s) \Rightarrow P \Downarrow T \Longrightarrow (THE\ a.\ \exists\, n.\ (c,\ s) \Rightarrow n \Downarrow a) = T$
⟨*proof*⟩

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*⟩

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

lemma *wp1While*: $wp_1\ (WHILE\ b\ DO\ c)\ Q\ l\ s = (if\ bval\ b\ s\ then\ wp_1\ c\ (wp_1\ (WHILE\ b\ DO\ c)\ Q)\ l\ s\ else\ Q\ l\ s)$
⟨*proof*⟩

lemma *wp1_prec2*: **fixes** *e*::*tbd*
shows $(wp_1\ c1\ Q\ l\ s \wedge$
$l\ x = prec\ c1\ e\ s) = wp_1\ c1\ (\lambda l\ s.\ Q\ l\ s \wedge e\ s = l\ x)\ l\ s$
⟨*proof*⟩

lemma *wp1_prec*: **fixes** *e*::*tbd*
shows $wp_1\ c1\ Q\ l\ s \Longrightarrow$
$l\ x = prec\ c1\ e\ s \Longrightarrow wp_1\ c1\ (\lambda l\ s.\ Q\ l\ s \wedge e\ s = l\ x)\ l\ s$
⟨*proof*⟩

lemma *wp1_is_pre*: $finite\ (support\ Q) \Longrightarrow \vdash_1 \{wp_1\ c\ Q\}\ c\ \{\ \lambda s.\ \downarrow_t\ (c,\ s) \Downarrow Q\}$
⟨*proof*⟩

lemma *valid_wp*: $\models_1 \{P\}c\{p \Downarrow Q\} \longleftrightarrow (\exists\, k{>}0.\ (\forall\, l\ s.\ P\ l\ s \longrightarrow (wp_1\ c\ Q\ l\ s \wedge ((THE\ n.\ (\exists\ t.\ ((c,s) \Rightarrow n \Downarrow t)))) \leq k * p\ s)))$
⟨*proof*⟩

theorem *hoare1_complete*: $finite\ (support\ Q) \Longrightarrow \models_1 \{P\}c\{p \Downarrow Q\} \Longrightarrow \vdash_1 \{P\}c\{p \Downarrow Q\}$
⟨*proof*⟩

corollary *hoare1_sound_complete*: *finite* (*support Q*) $\Longrightarrow \vdash_1 \{P\}c\{p \Downarrow Q\} \longleftrightarrow \models_1 \{P\}c\{p \Downarrow Q\}$
⟨*proof*⟩

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*: $x \notin$ *supportE* $Q \Longrightarrow Q\ (l(x{:=}n)) = Q\ l$
⟨*proof*⟩

lemma *supportE_if*: *supportE* $(\lambda l\ s.$ *if* $b\ s$ *then* $A\ l\ s$ *else* $B\ l\ s)$
$\subseteq$ *supportE* $A \cup$ *supportE* B
⟨*proof*⟩

lemma *support_eq*: *support* $(\lambda l\ s.\ l\ x = E\ l\ s) \subseteq$ *supportE* $E \cup \{x\}$
⟨*proof*⟩

lemma *support_impl_in*: $G\ e \longrightarrow$ *support* $(\lambda l\ s.\ \ H\ e\ l\ s) \subseteq T$
$\Longrightarrow$ *support* $(\lambda l\ s.\ G\ e \longrightarrow\ \ H\ e\ l\ s) \subseteq T$
⟨*proof*⟩

lemma *support_supportE*: $\bigwedge P\ e.$ *support* $(\lambda l\ s.\ \ P\ (e\ l)\ s) \subseteq$ *supportE* e
⟨*proof*⟩
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* $P \cup$ *varacom* $C \cup$ *support Qannot*
| *varacom* ({(*I*,(*S*,(*E*)))} *WHILE* b *DO* C) = *support* $I \cup$ *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* $C\ (\%s.\ k * e\ s) = (\%s.\ k * preT\ C\ e\ s)$
⟨*proof*⟩

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*

lemma *TQ*: *preT C e s* = *e* (*postQ C s*)
⟨*proof*⟩

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*)
⟨*proof*⟩

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*⟩

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*)
⟨*proof*⟩

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*)
⟨*proof*⟩

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*⟩

lemma *supportE_twicepreT*: *supportE* (%*l*. *preT C1* (*preT C2* (*e l*))) ⊆ *supportE e*
⟨*proof*⟩

lemma *supportE_preTz*: *supportE* (%*l*. *preTz C* (*e l*) *n*) ⊆ *supportE e*
⟨*proof*⟩

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*))
⟨*proof*⟩

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

lemma *pff*: ⋀*n*. *support* (λ*l*. *I* (*l*(*x* := *n*))) ⊆ *support I* − {*x*}
⟨*proof*⟩

lemma *pff*: ⋀*n*. *support* (λ*l*. *I* (*l*(*x* := *n*))) ⊆ *support I*
⟨*proof*⟩

lemma *supportAB*: *support* (λ*l s*. *A l s* ∧ *B s*) ⊆ *support A*

⟨*proof*⟩

lemma *support* (*pre* ({(*I*,(*S*,(*E*)))} *WHILE b DO C*) *Q*) ⊆ *support I*
⟨*proof*⟩

lemma *support_pre*: *support* (*pre C Q*) ⊆ *support Q* ∪ *varacom C*
⟨*proof*⟩

lemma *finite_support_pre*[*simp*]: *finite* (*support Q*) ⟹ *finite* (*varacom C*) ⟹ *finite* (*support* (*pre C Q*))
⟨*proof*⟩

fun *time* :: *acom* ⇒ *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*) = {}
⟨*proof*⟩

lemma *supportE_plus*: *supportE* (λ*l s*. *e1 l s* + *e2 l s*) ⊆ *supportE e1* ∪ *supportE e2*
⟨*proof*⟩

lemma *supportE_Suc*: *supportE* (λ*l s*. *Suc* (*e1 l s*)) = *supportE e1*
⟨*proof*⟩

lemma *supportE_single2*: *supportE* (λ*l* . *P*) = {}
⟨*proof*⟩

lemma *supportE_time*: *supportE* (λ*l*. *time C*) = {}
⟨*proof*⟩

lemma ⋀*s*. (∀ *l*. *I* (*l*(*x*:=*0*)) *s*) = (∀ *l*. *l x* = *0* ⟶ *I l s*)
⟨*proof*⟩

lemma $\bigwedge s.\ (\forall\, l.\ I\ (l(x{:=}Suc\ (l\ x)))\ s) = (\forall\, l.\ (\exists\, n.\ l\ x = Suc\ n) \longrightarrow I\ l\ s)$
⟨*proof*⟩

Verification condition:

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 ({*P*′/*Q*/*e*′} *CONSEQ C*) *Q*′ = (*vc C Q* ∧ (∃ *k*>*0*. (∀ *l s*. *P*′ *l s* ⟶ *time C s* ≤ *k* ∗ *e*′ *s* ∧ (∀ *t*. ∃ *l*′. (*pre C Q*) *l*′ *s* ∧ (*Q l*′ *t* ⟶ *Q*′ *l t*))))) |

vc ({(*I*,(*S*,(*E*)))} *WHILE b DO C*) *Q* =
((∀ *l s*. (*I l s* ∧ *bval b s* ⟶ *pre C I l s* ∧ *E s* ≥ *1* + *preT C E s* + *time C s*
∧ *S s* = *S* (*postQ C s*)) ∧
(*I l s* ∧ ¬ *bval b s* ⟶ *Q l s* ∧ *E s* ≥ *1* ∧ *S s* = *s*)) ∧
vc C I)

lemma *pre_mono*:
(∀ *l s*. *P l s* ⟶ *P*′ *l s*) ⟹ *pre C P l s* ⟹ *pre C P*′ *l s*
⟨*proof*⟩

lemma *vc_mono*: (∀ *l s*. *P l s* ⟶ *P*′ *l s*) ⟹ *vc C P* ⟹ *vc C P*′
⟨*proof*⟩

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*: ⋀*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*⟩

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)*
⟨*proof*⟩
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*⟩

lemma *ListUpdateE_updates*: *distinct (map fst xs) ⟹ x ∈ set xs ⟹ ListUpdateE l'' xs (fst x) = snd x*
⟨*proof*⟩

lemma *ListUpdate_updates*: *x ∈ fst ‘ (set xs) ⟹ ListUpdate l'' xs (%e. l) x = l x*
⟨*proof*⟩

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*⟩

lemma *support_True*[*simp*]: *support (λa b. True) = {}*
⟨*proof*⟩

lemma *support_preList*: *support (preList upds C1) ⊆ lesvars upds*
⟨*proof*⟩

lemma *preListpreSet*: *preSet (set xs) C l s ⟹ preList xs C l s*
⟨*proof*⟩

lemma *preSetpreList*: *preList xs C l s ⟹ preSet (set xs) C l s*
⟨*proof*⟩

lemma *preSetpreList_eq*: *preList xs C l s* = *preSet* (*set xs*) *C l s*
⟨*proof*⟩

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*⟩

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* (λ(*x*, *e*). (*x*, λ*s*. *e* (*S s*))) *upds*) *C l s*
⟨*proof*⟩

lemma *postList_preList*: *postList* (*map* (λ(*x*, *e*). (*x*, *preT C e*)) *upds*) *l s* = *preList upds C l s*
⟨*proof*⟩

lemma *postSetpostList*: *postList xs l s* ⟹ *postSet* (*set xs*) *l s*
⟨*proof*⟩

lemma *postListpostSet*: *postSet* (*set xs*) *l s* ⟹ *postList xs l s*
⟨*proof*⟩

lemma *ListAskip*: *preList xs Askip l s* = *postList xs l s*
⟨*proof*⟩

lemma *SetAskip*: *preSet U Askip l s* = *postSet U l s*
⟨*proof*⟩

lemma *ListAassign*: *preList upds* (*Aassign x1 x2*) *l s* = *postList upds l* (*s*[*x2*/*x1*])
⟨*proof*⟩

lemma *SetAassign*: *preSet U* (*Aassign x1 x2*) *l s* = *postSet U l* (*s*[*x2*/*x1*])
⟨*proof*⟩

lemma *ListAconseq*: *preList upds* (*Aconseq x1 x2 x3 C*) *l s* = *preList upds C l s*
⟨*proof*⟩

lemma *SetAconseq*: *preSet U* (*Aconseq x1 x2 x3 C*) *l s* = *preSet U C l s*
⟨*proof*⟩

lemma *ListAif1*: *bval b s* ⟹ *preList upds* (*IF b THEN C1 ELSE C2*) *l s* = *preList upds C1 l s*
⟨*proof*⟩
lemma *SetAif1*: *bval b s* ⟹ *preSet upds* (*IF b THEN C1 ELSE C2*) *l s* = *preSet upds C1 l s*
⟨*proof*⟩
lemma *ListAif2*: ~ *bval b s* ⟹ *preList upds* (*IF b THEN C1 ELSE C2*) *l s* = *preList upds C2 l s*
⟨*proof*⟩

lemma *SetAif2*: ~ *bval b s* ⟹ *preSet upds* (*IF b THEN C1 ELSE C2*) *l s* = *preSet upds C2 l s*
⟨*proof*⟩

lemma *vc_sound*: *vc C Q* ⟹ *finite* (*support Q*) ⟹ *finite* (*varacom C*)
⟹ *fst* ‘ (*set upds*) ∩ *varacom C* = {} ⟹ *distinct* (*map fst upds*)
⟹ ⊢$_1$ {%*l s. pre C Q l s* ∧ *preList upds C l s*} *strip C* { *time C* ⇓ %*l s. Q l s* ∧ *postList upds l s*}
∧ (∀ *l s. pre C Q l s* ⟶ *Q l* (*postQ C s*))
⟨*proof*⟩

corollary *vc_sound′*:
assumes *vc C Q*
finite (*support Q*) *finite* (*varacom C*)
∀ *l s. P l s* ⟶ *pre C Q l s*
shows ⊢$_1$ {*P*} *strip C* {*time C* ⇓ *Q*}
⟨*proof*⟩

lemma *preT_constant*: *preT C* (%_. *a*) = (%_. *a*)
⟨*proof*⟩

corollary *vc_sound''*:
⟦ *vc C Q*; (∃ *k*>*0*. ∀ *l s*. *P l s* ⟶ *pre C Q l s* ∧ *time C s* ≤ *k* ∗ *e s*);
finite (*support Q*); *finite* (*varacom C*)⟧ ⟹ ⊢$_1$ {*P*} *strip C* {*e* ⇓ *Q*}
⟨*proof*⟩

4.6.4 Completeness:

lemma *vc_complete*:
⊢$_1$ {*P*} *c* { *e* ⇓ *Q*} ⟹ ∃ *C*. *strip C* = *c* ∧ *vc C Q*
∧ (∀ *l s*. *P l s* ⟶ *pre C Q l s* ∧ *Q l* (*postQ C s*))
∧ (∃ *k*. ∀ *l s*. *P l s* ⟶ *time C s* ≤ *k* ∗ *e s*)
(**is** _ ⟹ ∃ *C*. *?G P c Q C e*)
⟨*proof*⟩

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 ⇒ *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 ∪ *vars* a_2 |
vars (*Times* a_1 a_2) = *vars* a_1 ∪ *vars* a_2 |
vars (*Div* a_1 a_2) = *vars* a_1 ∪ *vars* a_2

instance ⟨*proof*⟩

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 ∪ *vars* b_2 |
vars (*Less* a_1 a_2) = *vars* a_1 ∪ *vars* a_2

instance ⟨*proof*⟩

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* == ∀ *x* ∈ *X*. *f x* = *g x*

lemma *aval_eq_if_eq_on_vars*[*simp*]:
s_1 = s_2 *on vars a* ⟹ *aval a* s_1 = *aval a* s_2
⟨*proof*⟩

lemma *bval_eq_if_eq_on_vars*:
s_1 = s_2 *on vars b* ⟹ *bval b* s_1 = *bval b* s_2
⟨*proof*⟩

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 ⟨*proof*⟩

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*
⟨*proof*⟩

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*
⟨*proof*⟩

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*)
⟨*proof*⟩

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*)
⟨*proof*⟩

lemma *preT_linear*: *preT C* (%*s*. *k* ∗ *e s*) = (%*s*. *k* ∗ *preT C e s*)
⟨*proof*⟩

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* ∪ *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* ∪ *fune C*$_1$ *LV* ∪ *fune C*$_2$

LV |
  fune ({(_,(*S*,(*E*,*Es*,*SS*)))} *WHILE b DO C*) *LV* = (⋃ *x*∈*LV*. *SS x*)

lemma *fune_mono*: *A* ⊆ *B* ⟹ *fune C A* ⊆ *fune C B*
⟨*proof*⟩

lemma *TQ*: *preT C e s* = *e* (*postQ C s*)
  ⟨*proof*⟩

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*)
  ⟨*proof*⟩

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*⟩

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*)
  ⟨*proof*⟩

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 C*$_1$ *ELSE C*$_2$) *Q* =

$(\lambda l\ s.\ \mathit{if}\ \mathit{bval}\ b\ s\ \mathit{then}\ \mathit{pre}\ C_1\ Q\ l\ s\ \mathit{else}\ \mathit{pre}\ C_2\ Q\ l\ s)\ |$
$\mathit{pre}\ (\{((I,Is),(S,(E,Es,SS)))\}\ \mathit{WHILE}\ b\ \mathit{DO}\ C)\ Q = I$

fun *qdeps* :: *acom* ⇒ *vname set* ⇒ *vname set* **where**
$\mathit{qdeps}\ \mathit{SKIP}\ LV\ = LV\ |$
$\mathit{qdeps}\ (x ::= a)\ LV = LV \cup \mathit{vars}\ a\ |$
$\mathit{qdeps}\ (C_1;;\ C_2)\ LV\ = \mathit{qdeps}\ C_1\ (\mathit{qdeps}\ C_2\ LV)\ |$
$\mathit{qdeps}\ (\{(P',Ps)/\_/\_\}\ \mathit{CONSEQ}\ C)\ \_ = Ps\ |$
$\mathit{qdeps}\ (\mathit{IF}\ b\ \mathit{THEN}\ C_1\ \mathit{ELSE}\ C_2)\ LV = \mathit{vars}\ b \cup \mathit{qdeps}\ C_1\ LV \cup \mathit{qdeps}\ C_2\ LV\ |$
$\mathit{qdeps}\ (\{((I,Is),(S,(E,x,Es)))\}\ \mathit{WHILE}\ b\ \mathit{DO}\ C)\ \_ = Is$

lemma *qdeps_mono*: $A \subseteq B \Longrightarrow \mathit{qdeps}\ C\ A \subseteq \mathit{qdeps}\ C\ B$
⟨*proof*⟩

lemma *supportE_if*: $\mathit{supportE}\ (\lambda l\ s.\ \mathit{if}\ b\ s\ \mathit{then}\ A\ l\ s\ \mathit{else}\ B\ l\ s)$
$\subseteq \mathit{supportE}\ A \cup \mathit{supportE}\ B$
⟨*proof*⟩

lemma *supportE_preT*: $\mathit{supportE}\ (\%l.\ \mathit{preT}\ C\ (e\ l)) \subseteq \mathit{supportE}\ e$
⟨*proof*⟩

lemma *supportE_twicepreT*: $\mathit{supportE}\ (\%l.\ \mathit{preT}\ C1\ (\mathit{preT}\ C2\ (e\ l))) \subseteq \mathit{supportE}\ e$
⟨*proof*⟩

lemma *supportE_preTz*: $\mathit{supportE}\ (\%l.\ \mathit{preTz}\ C\ (e\ l)\ n) \subseteq \mathit{supportE}\ e$
⟨*proof*⟩

lemma *supportE_preTz_Un*:
$\mathit{supportE}\ (\lambda l.\ \mathit{preTz}\ C\ (e\ l)\ (l\ x)) \subseteq \mathit{insert}\ x\ (\mathit{UN}\ n.\ \mathit{supportE}\ (\lambda l.\ \mathit{preTz}\ C\ (e\ l)\ n))$
⟨*proof*⟩

lemma *support_eq*: $\mathit{support}\ (\lambda l\ s.\ l\ x = E\ l\ s) \subseteq \mathit{supportE}\ E \cup \{x\}$
⟨*proof*⟩

lemma *support_impl_in*: $G\ e \longrightarrow \mathit{support}\ (\lambda l\ s.\ \ H\ e\ l\ s) \subseteq T$
$\Longrightarrow \mathit{support}\ (\lambda l\ s.\ G\ e \longrightarrow\ H\ e\ l\ s) \subseteq T$

⟨*proof*⟩

lemma *support_supportE*: $\bigwedge P\ e.\ support\ (\lambda l\ s.\ \ P\ (e\ l)\ s) \subseteq supportE\ e$
⟨*proof*⟩

lemma *support_pre*: $support\ (pre\ C\ Q) \subseteq support\ Q \cup varacom\ C$
⟨*proof*⟩

lemma *finite_support_pre*: $finite\ (support\ Q) \implies finite\ (varacom\ C) \implies finite\ (support\ (pre\ C\ Q))$
⟨*proof*⟩

fun *time* :: *acom* ⇒ *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*,*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*: $supportE\ (\lambda l\ s.\ P) = \{\}$
⟨*proof*⟩

lemma *supportE_plus*: $supportE\ (\lambda l\ s.\ e1\ l\ s + e2\ l\ s) \subseteq supportE\ e1 \cup supportE\ e2$
⟨*proof*⟩

lemma *supportE_Suc*: $supportE\ (\lambda l\ s.\ Suc\ (e1\ l\ s)) = supportE\ e1$
⟨*proof*⟩

lemma *supportE_single2*: *supportE* $(\lambda l\ .\ P) = \{\}$
⟨*proof*⟩

lemma *supportE_time*: *supportE* $(\lambda l.\ time\ C) = \{\}$
⟨*proof*⟩

lemma $\bigwedge s.\ (\forall\, l.\ I\ (l(x{:=}0))\ s) = (\forall\, l.\ l\ x = 0 \longrightarrow I\ l\ s)$
⟨*proof*⟩

lemma $\bigwedge s.\ (\forall\, l.\ I\ (l(x{:=}Suc\ (l\ x)))\ s) = (\forall\, l.\ (\exists\, n.\ l\ x = Suc\ n) \longrightarrow I\ l\ s)$
⟨*proof*⟩

Verification condition:

definition *funStar* **where** *funStar f* = (%*x*. {*y*. (*x*,*y*)∈{(*x*,*y*). *y*∈*f x*}*})

lemma *funStart_prop1*: $x \in (funStar\ f)\ x$ ⟨*proof*⟩
lemma *funStart_prop2*: $f\ x \subseteq (funStar\ f)\ x$ ⟨*proof*⟩

fun *vc* :: *acom* ⇒ *assn2* ⇒ *vname set* ⇒ *vname set* ⇒ *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* ∪ *kdeps* C_2)) ∧ (*vc* C_2 *Q LVQ LVE*)) |
vc (*IF b THEN* C_1 *ELSE* C_2) *Q LVQ LVE* = (*vc* C_1 *Q LVQ LVE* ∧ *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
∧ (∀ *s1 s2 l*. (∀ *x*∈*Ps*. *s1 x*=*s2 x*) ⟶ *P' l s1* = *P' l s2*) — annotation *Ps* (the set of variables *P'* depends on) is correct
∧ (∀ *s1 s2 l*. (∀ *x*∈*Qs*. *s1 x*=*s2 x*) ⟶ *Q l s1* = *Q l s2*) — annotation *Qs* (the set of variables *Q* depends on) is correct
∧ (∀ *s1 s2*. (∀ *x*∈*es*. *s1 x*=*s2 x*) ⟶ *e' s1* = *e' s2*) — annotation *es* (the set of variables *e'* depends on) is correct
∧ (∃ *k*>*0*. (∀ *l s*. *P' l s* ⟶ *time C s* ≤ *k* ∗ *e' s* ∧ (∀ *t*. ∃ *l'*. (*pre C Q*) *l' s* ∧ (*Q l' t* ⟶ *Q' l t*))))) |

vc ({((*I*,*Is*),(*S*,(*E*,*es*,*SS*)))} *WHILE b DO C*) *Q LVQ LVE* = ((∀ *s1 s2 l*. (∀ *x*∈*Is*. *s1 x* = *s2 x*) ⟶ *I l s1* = *I l s2*) — annotation *Is* is correct
∧ (∀ *y*∈*LVE* ∪ *LVQ*. (*let Ss*=*SS y in* (∀ *s1 s2*. (∀ *x*∈*Ss*. *s1 x* = *s2 x*) ⟶ (*S s1*) *y* = (*S s2*) *y*))) — annotation *SS* is correct, for only one step
∧ (∀ *s1 s2*. (∀ *x*∈*es*. *s1 x*=*s2 x*) ⟶ *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 $\geq$ 1 + $preT$ C E s + $time$ C s
$\wedge$ ($\forall$ $v$$\in$($\bigcup$ $y$$\in$$LVE$ $\cup$ LVQ. ($funStar$ SS) y). (S s) v = (S ($postQ$ C s)) v)) $\wedge$
(I l s $\wedge$ $\neg$ $bval$ b s $\longrightarrow$ Q l s $\wedge$ E s $\geq$ 1 $\wedge$ ($\forall$ $v$$\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) $\Rightarrow$ l x = $preT$ C e s))
abbreviation $postSet$ U l s == ($Ball$ U (%u. $case$ u of (x,e,v) $\Rightarrow$ 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 $U2$: $\bigwedge l$ s n x. $x$$\in$ fst ‘ $upds$ $\Longrightarrow$ A (l(x := n)) = A l
shows
fst ‘ set xs $\subseteq$ fst ‘ $upds$ $\Longrightarrow$ A ($ListUpdate$ l'' xs q) = A l''
⟨$proof$⟩

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)
⟨$proof$⟩
lemma $allg\_E$: **fixes** A::$assn2$
assumes
($\bigwedge l$ s n x. x $\in$ fst ‘ $upds$ $\Longrightarrow$ A (l(x := n)) = A l) fst ‘ set xs $\subseteq$ fst ‘ $upds$
shows A ($ListUpdateE$ f xs) = A f
⟨$proof$⟩

lemma $ListUpdateE\_updates$: $distinct$ (map fst xs) $\Longrightarrow$ x $\in$ set xs $\Longrightarrow$ $ListUpdateE$ l'' xs (fst x) = fst (snd x)
⟨$proof$⟩

lemma $ListUpdate\_updates$: x $\in$ fst ‘ (set xs) $\Longrightarrow$ $ListUpdate$ l'' xs (%e. l)

$x = l\ x$
⟨*proof*⟩

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*⟩

lemma [*simp*]: *support* (λ*a b*. *True*) = {}
  ⟨*proof*⟩

lemma *support_preList*: *support* (*preList upds C1*) ⊆ *lesvars upds*
⟨*proof*⟩

lemma *preListpreSet*: *preSet* (*set xs*) *C l s* ⟹ *preList xs C l s*
⟨*proof*⟩

lemma *preSetpreList*: *preList xs C l s* ⟹ *preSet* (*set xs*) *C l s*
⟨*proof*⟩

lemma *preSetpreList_eq*: *preList xs C l s* = *preSet* (*set xs*) *C l s*
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *support_postList*: *support* (*postList xs*) ⊆ *lesvars xs*
⟨*proof*⟩

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*⟩

lemma *postSetpostList*: *postList xs l s* ⟹ *postSet* (*set xs*) *l s*
⟨*proof*⟩

lemma *postListpostSet*: *postSet* (*set xs*) *l s* ⟹ *postList xs l s*
⟨*proof*⟩

lemma *postListpostSet2*: *postList xs l s* = *postSet* (*set xs*) *l s*
⟨*proof*⟩

lemma *ListAskip*: *preList xs Askip l s* = *postList xs l s*
⟨*proof*⟩

lemma *SetAskip*: *preSet U Askip l s* = *postSet U l s*
⟨*proof*⟩

lemma *ListAassign*: *preList upds* (*Aassign x1 x2*) *l s* = *postList upds l* (*s*[*x2*/*x1*])
⟨*proof*⟩

lemma *SetAassign*: *preSet U* (*Aassign x1 x2*) *l s* = *postSet U l* (*s*[*x2*/*x1*])
⟨*proof*⟩

lemma *ListAconseq*: *preList upds* (*Aconseq x1 x2 x3 C*) *l s* = *preList upds C l s*
⟨*proof*⟩

lemma *SetAconseq*: *preSet U* (*Aconseq x1 x2 x3 C*) *l s* = *preSet U C l s*
⟨*proof*⟩

lemma *ListAif1*: *bval b s* ⟹ *preList upds* (*IF b THEN C1 ELSE C2*) *l s* = *preList upds C1 l s*
⟨*proof*⟩
lemma *SetAif1*: *bval b s* ⟹ *preSet upds* (*IF b THEN C1 ELSE C2*) *l s* = *preSet upds C1 l s*

⟨*proof*⟩
lemma *ListAif2*: ~ *bval b s* ⟹ *preList upds* (*IF b THEN C1 ELSE C2*) *l s* = *preList upds C2 l s*
⟨*proof*⟩

lemma *SetAif2*: ~ *bval b s* ⟹ *preSet upds* (*IF b THEN C1 ELSE C2*) *l s* = *preSet upds C2 l s*
⟨*proof*⟩

definition *K* **where** *K C LVQ Q* == (∀ *l s1 s2*. *s1* = *s2 on qdeps C LVQ* ⟶ *pre C Q l s1* = *pre C Q l s2*)

definition *K2* **where** *K2 C e Es Q* == (∀ *s1 s2*. *s1* = *s2 on fune C Es* ⟶ *preT C e s1* = *preT C e s2*)

definition *K3* **where** *K3 upds C Q* = (∀ (*a*,*b*,*c*)∈*set upds*. *K2 C b c Q*)
definition *K4* **where** *K4 upds LV C Q* = (*K C LV Q* ∧ *K3 upds C Q* ∧ (∀ *s1 s2*. *s1* = *s2 on kdeps C* ⟶ *time C s1* = *time C s2*))

lemma *k4If*: *K4 upds LVQ C1 Q* ⟹ *K4 upds LVQ C2 Q* ⟹ *K4 upds LVQ* (*IF b THEN C1 ELSE C2*) *Q*
⟨*proof*⟩

4.8.2 Soundness

lemma *vc_sound*: *vc C Q LVQ LVE* ⟹ *finite* (*support Q*)
⟹ *fst* ‘ (*set upds*) ∩ *varacom C* = {} ⟹ *distinct* (*map fst upds*)
⟹ *finite* (*varacom C*)
⟹ (∀ *l s1 s2*. *s1* = *s2 on LVQ* ⟶ *Q l s1* = *Q l s2*)
⟹ (∀ *l s1 s2*. *s1* = *s2 on LVE* ⟶ *postList upds l s1* = *postList upds l s2*)
⟹ (∀ (*a*,*b*,*c*)∈*set upds*. (∀ *s1 s2*. *s1* = *s2 on c* ⟶ *b s1* = *b s2*)) — *c* are really the variables *b* depends on
⟹ (⋃ (*a*,*b*,*c*)∈*set upds*. *c*) ⊆ *LVE* — in *LV* are all the variables that the expressions in *upds* depend on
⟹ ⊢$_1$ {%*l s*. *pre C Q l s* ∧ *preList upds C l s*} *strip C* { *time C* ⇓ %*l s*. *Q l s* ∧ *postList upds l s*}
∧ ((∀ *l s*. *pre C Q l s* ⟶ *Q l* (*postQ C s*)) ∧ *K4 upds LVQ C Q*)
⟨*proof*⟩

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*⟩

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*⟩

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*
⟨*proof*⟩

lemma *fune_UNIV*: *fune* (*transl C*) *UNIV* = *UNIV*

⟨*proof*⟩

lemma *pre_transl*: *Nielson_VCGi.pre* (*transl C*) *Q* = *Nielson_VCG.pre C Q*
⟨*proof*⟩

lemma *preT_transl*: *Nielson_VCGi.preT* (*transl C*) *E* = *Nielson_VCG.preT C E*
⟨*proof*⟩

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

lemma *time_transl*: *Nielson_VCGi.time* (*transl C*) = *Nielson_VCG.time C*
⟨*proof*⟩

lemma *vc_transl*: *Nielson_VCG.vc C Q* ⟹ *Nielson_VCGi.vc* (*transl C*) *Q UNIV UNIV*
⟨*proof*⟩

lemma *strip_transl*: *Nielson_VCGi.strip* (*transl C*) = *Nielson_VCG.strip C*
⟨*proof*⟩

lemma *vc_restrict_complete*:
assumes ⊢$_1$ {*P*} *c* { *e* ⇓ *Q*}
shows ∃ *C*. *Nielson_VCGi.strip C* = *c* ∧ *Nielson_VCGi.vc C Q UNIV UNIV*
∧ (∀ *l s*. *P l s* ⟶ *Nielson_VCGi.pre C Q l s* ∧ *Q l* (*Nielson_VCGi.postQ C s*))
∧ (∃ *k*. ∀ *l s*. *P l s* ⟶ *Nielson_VCGi.time C s* ≤ *k* ∗ *e s*)
(**is** ∃ *C*. *?G P c Q C e*)
⟨*proof*⟩

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*⟩

lemma *finite* (*support P*) $\Longrightarrow \vdash_1 \{P\}\ strip\ Askip\ \{time\ Askip \Downarrow P\}$
⟨*proof*⟩

lemma *support_single2*: *support* ($\lambda l\ s.\ P\ s$) = {}
⟨*proof*⟩

lemma $\vdash_1$ { %*l s. True* } *strip* (*Aassign a* (*N 1*)) {*time* (*Aassign a* (*N 1*)) $\Downarrow$ %*l s. s a* = *1*}
⟨*proof*⟩

lemma $\vdash_1$ { %*l s. True* } *strip* ((*a* ::= (*N 1*)) ;; *Askip*) {*time* ((*a* ::= (*N 1*)) ;; *Askip*) $\Downarrow$ %*l s. s a* = *1*}
⟨*proof*⟩

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

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 $\bigwedge$*s. 0* ≤ *s a* $\Longrightarrow$ *time C s* ≤ *9*
⟨*proof*⟩

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* $\Downarrow$ %*l s. True* }
⟨*proof*⟩

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: $S = (\%s.\ (if\ s\ a \geq b\ then\ s\ else\ s(a{:=}b))\ )$ **and**
C: $C = (\{(I,(S,(E)))\}\ WHILE\ Less\ (V\ a)\ (N\ b)\ DO\ a ::= Plus\ (V\ a)\ (N\ 1)\ )$
shows $\vdash_1 \{\ \%l\ s.\ 0 \leq s\ a\ \}\ strip\ C\ \{time\ C \Downarrow \%l\ s.\ True\ \}$
⟨*proof*⟩

Example: multiplication by repeated addition **lemma** *helper*: $(A{::}int) * B + B = (A{+}1) * B$ ⟨*proof*⟩

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 $\vdash_1 \{\ \%l\ s.\ 0 \leq s\ ''a''\ \}\ strip\ C\ \{\ f \Downarrow \%l\ s.\ s\ ''y'' = s\ ''a'' * (s\ ''b'')\ \}$
⟨*proof*⟩

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 $\vdash_1 \{\ \%l\ s.\ 0 \leq s\ ''a''\ \}\ strip\ (\ \{\%l\ s.\ 0 \leq s\ ''a''/\ \%l\ s.\ s\ ''y'' = s\ ''a'' * (s\ ''b'')/\ f\}\ CONSEQ\ C)\ \{\ f \Downarrow \%l\ s.\ s\ ''y'' = s\ ''a'' * (s\ ''b'')\ \}$

⟨*proof*⟩

Example: nested loops **lemma** *nested*: **assumes**
I2: *I2* = (%*l s*. *s* ″*a*″ ≥ *0* ∧ *s* ″*b*″ ≥ *0* ∧ *s* ″*a*″ > *s* ″*z*″ ∧ *s* ″*z*″ ≥ *0* ∧ *s* ″*b*″ ≥ *s* ″*g*″ ∧ *s* ″*g*″ ≥ *0* ∧ *s* ″*y*″ = (*s* ″*z*″) ∗ (*s* ″*b*″) + *s* ″*g*″) **and**

I1: *I1* = (%*l s*. *s* ″*a*″ ≥ *0* ∧ *s* ″*b*″ ≥ *0* ∧ *s* ″*a*″ ≥ *s* ″*z*″ ∧ *s* ″*z*″ ≥ *0* ∧ *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*″ ≥ *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*″ ≥ *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 ⊢$_1$ { %*l s*. *0* ≤ *s* ″*a*″ ∧ *s* ″*b*″ ≥ *0* } *strip C* { *f* ⇓ %*l s*. *s* ″*y*″ = *s* ″*a*″ ∗ (*s* ″*b*″) }
⟨*proof*⟩

with logical variables **lemma** *fin_sup_single*: *finite* (*support* (λ*l*. *P* (*l a*)))
⟨*proof*⟩

lemmas *fin_support* = *fin_sup_single*

lemma *finite_support_and*: *finite* (*support A*) ⟹ *finite* (*support B*) ⟹

finite (*support* (λ*l s*. *A l s* ∧ *B l s*))
⟨*proof*⟩

end
theory *Nielson_Sqrt*
imports *Nielson_VCGi HOL−Library.Discrete*
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*: ⋀*s k*. *1* + *s ''x''* = *2* ^ *k* ⟹ *5* ∗ *k* ≤ *96* + *100* ∗ *Discrete.log* (*nat* (*s ''x''*))
⟨*proof*⟩

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* = (λ*l s*. (∃*k*. *1* + *s ''x''* = *2* ^ *k*))
and *e* : *e* = (λ*s*. *Discrete.log* (*nat* (*s ''x''*)) + *1*) **and**
Q[*simp*]: *Q* = (λ*l s*. *True*)
shows ⊢$_1$ {*P*} *c* { *e* ⇓ *Q*}
⟨*proof*⟩

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*
(⊨$_2$ {(*1*_)}/ (_)/ {(*1*_)} *50*) **where**
⊨$_2$ {*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* (⊢$_2$ ({(*1*_)}/ (_)/ {(*1*_)}) *50*)
where

Skip: ⊢$_2$ {%*s*. *eSuc* (*P s*)} *SKIP* {*P*} |

Assign: ⊢$_2$ {λ*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: $[\![\vdash_2 \{P\}c\{Q\}\ ;\ \bigwedge s.\ P\ s \leq P'\ s\ ;\ \bigwedge s.\ Q'\ s \leq Q\ s\ ]\!] \Longrightarrow$
$\vdash_2 \{P'\}c\{\ Q'\}$

derived rules

lemma *strengthen_pre*: $[\![\forall s.\ P\ s \leq P'\ s;\ \vdash_2 \{P\}\ c\ \{Q\}\ ]\!] \Longrightarrow \vdash_2 \{P'\}\ c\ \{Q\}$
⟨*proof*⟩

lemma *weaken_post*: $[\![\vdash_2 \{P\}\ c\ \{Q\};\ \forall s.\ Q\ s \geq Q'\ s\ ]\!] \Longrightarrow\ \vdash_2 \{P\}\ c\ \{Q'\}$
⟨*proof*⟩

lemma *Assign'*: $\forall s.\ P\ s \geq eSuc\ (\ Q(s[a/x])) \Longrightarrow \vdash_2 \{P\}\ x ::= a\ \{Q\}$
⟨*proof*⟩

lemma *progress*: $(c,\ s) \Rightarrow p \Downarrow t \Longrightarrow p > 0$
⟨*proof*⟩

lemma *FalseImplies*: $\vdash_2 \{\%s.\ \infty\}\ c\ \{\ Q\}$
⟨*proof*⟩

5.3 Soundness

The soundness theorem:

lemma *help1*: **assumes** $enat\ a + X \leq Y$
$enat\ b + Z \leq X$
shows $enat\ (a + b) + Z \leq Y$
⟨*proof*⟩

lemma *help2'*: **assumes** $enat\ p + INV\ t\ \leq INV\ s$
$0 < p\ INV\ s = enat\ n$
shows $INV\ t < INV\ s$

⟨*proof*⟩

lemma *help2*: **assumes** *enat p + INV t + 1 ≤ INV s*
INV s = enat n
shows *INV t < INV s*
⟨*proof*⟩

lemma *Seq_sound*: **assumes** $\models_2$ {*P1*} *C1* {*P2*}
$\models_2$ {*P2*} *C2* {*P3*}
shows $\models_2$ {*P1*} *C1* ;; *C2* {*P3*}
⟨*proof*⟩

theorem *hoare2_sound*: $\vdash_2$ {*P*}*c*{ *Q*} ⟹ $\models_2$ {*P*}*c*{ *Q*}
⟨*proof*⟩

5.4 Completeness

definition *wp2* :: *com* ⇒ *qassn* ⇒ *qassn* (*wp*$_2$) **where**
wp$_2$ *c Q* = (λ*s*. (*if* (∃ *t p*. (*c*,*s*) ⇒ *p* ⇓ *t* ∧ *Q t* < ∞) *then enat* (*THE p*. ∃ *t*. (*c*,*s*) ⇒ *p* ⇓ *t*) + *Q* (*THE t*. ∃ *p*. (*c*,*s*) ⇒ *p* ⇓ *t*) *else* ∞))

lemma *wp2_alt*: *wp*$_2$ *c Q* = (λ*s*. (*if* ↓(*c*,*s*) *then enat* (↓$_t$ (*c*, *s*)) + *Q* (↓$_s$ (*c*, *s*)) *else* ∞))
⟨*proof*⟩

theorem *wp2_is_weakestprePotential*: $\models_2$ {*P*}*c*{*Q*} ⟷ (∀ *s*. *wp*$_2$ *c Q s* ≤ *P s*)
⟨*proof*⟩

lemma *wp2_Skip*[*simp*]: *wp*$_2$ *SKIP Q* = (%*s*. *eSuc* (*Q s*))
⟨*proof*⟩

lemma *wp2_Assign*[*simp*]: *wp*$_2$ (*x* ::= *e*) *Q* = (λ*s*. *eSuc* (*Q* (*s*(*x* := *aval e s*))))
⟨*proof*⟩

lemma *wp2_Seq*[*simp*]: *wp*$_2$ (*c*$_1$;;*c*$_2$) *Q* = *wp*$_2$ *c*$_1$ (*wp*$_2$ *c*$_2$ *Q*)
⟨*proof*⟩

lemma *wp2_If*[*simp*]:
$wp_2\ (IF\ b\ THEN\ c_1\ ELSE\ c_2)\ Q = (\lambda s.\ eSuc\ (wp_2\ (if\ bval\ b\ s\ then\ c_1\ else\ c_2)\ Q\ s))$
⟨*proof*⟩

lemma assumes *b*: *bval b s*
shows *wp2WhileTrue*: $wp_2\ c\ (wp_2\ (WHILE\ b\ DO\ c)\ Q)\ s\ +\ 1 \leq wp_2\ (WHILE\ b\ DO\ c)\ Q\ s$
⟨*proof*⟩

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*⟩

lemma assumes *b*: ~ *bval b s*
shows *wp2WhileFalse*: $Q\ s\ +\ 1 \leq wp_2\ (WHILE\ b\ DO\ c)\ Q\ s$
⟨*proof*⟩

lemma *thet_WhileFalse*: ~ *bval b s* $\Longrightarrow \downarrow_t (WHILE\ b\ DO\ c, s) = 1$ ⟨*proof*⟩

lemma *thes_WhileFalse*: ~ *bval b s* $\Longrightarrow \downarrow_s (WHILE\ b\ DO\ c, s) = s$ ⟨*proof*⟩

lemma assumes *b*: ~ *bval b s*
shows *wp2WhileFalse'*: $Q\ s\ +\ 1 = wp_2\ (WHILE\ b\ DO\ c)\ Q\ s$
⟨*proof*⟩

lemma *wp2While*: $(if\ bval\ b\ s\ then\ wp_2\ c\ (wp_2\ (WHILE\ b\ DO\ c)\ Q)\ s\ else\ Q\ s)\ +\ 1 = wp_2\ (WHILE\ b\ DO\ c)\ Q\ s$
⟨*proof*⟩

lemma assumes $\bigwedge Q.\ \vdash_2 \{wp_2\ c\ Q\}\ c\ \{Q\}$
shows $\vdash_2 \{wp_2\ (WHILE\ b\ DO\ c)\ Q\}\ WHILE\ b\ DO\ c\ \{Q\}$
⟨*proof*⟩

lemma *wp2_is_pre*: $\vdash_2 \{wp_2\ c\ Q\}\ c\ \{\ Q\}$
⟨*proof*⟩

lemma *wp2_is_weakestprePotential1*: $\models_2 \{P\}c\{Q\} \Longrightarrow (\forall s.\ wp_2\ c\ Q\ s \leq P\ s)$
⟨*proof*⟩

lemma *wp2_is_weakestprePotential2*: $(\forall s.\ wp_2\ c\ Q\ s \leq P\ s) \Longrightarrow \models_2 \{P\}c\{Q\}$
⟨*proof*⟩

theorem *wp2_is_weakestprePotential*: $(\forall s.\ wp_2\ c\ Q\ s \leq P\ s) \longleftrightarrow \models_2 \{P\}c\{Q\}$
⟨*proof*⟩

theorem *hoare2_complete*: $\models_2 \{P\}c\{Q\} \Longrightarrow \vdash_2 \{P\}c\{\ Q\}$
⟨*proof*⟩

corollary *hoare2_sound_complete*: $\vdash_2 \{P\}c\{Q\} \longleftrightarrow \models_2 \{P\}c\{\ Q\}$
⟨*proof*⟩

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*⟩

lemma *vc_sound′*: ⟦*vc C Q* ; (∀ *s*. *pre C Q s* ≤ *P s*) ⟧ ⟹ ⊢$_2$ {*P*} *strip C* { *Q* }
⟨*proof*⟩

5.5.2 Completeness

lemma *pre_mono*: **assumes** ⋀*s*. *P′ s* ≤ *P s*
shows ⋀*s*. *pre C P′ s* ≤ *pre C P s*
⟨*proof*⟩

lemma *vc_mono*: **assumes** $\bigwedge s.\ P'\ s \le P\ s$
shows *vc C P* $\Longrightarrow$ *vc C P'*
⟨*proof*⟩

lemma $\vdash_2 \{\ P\ \}\ c\ \{\ Q\ \} \Longrightarrow \exists\, C.\ strip\ C = c \wedge vc\ C\ Q \wedge (\forall\, s.\ pre\ C\ Q\ s \le P\ s)$
(**is** _ $\Longrightarrow$ $\exists\, C.\ ?G\ P\ c\ Q\ C$)
⟨*proof*⟩

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*: $\vdash_2$ {λ*s. enat* (*2* + *3*∗*n*) + *emb* (*s ''x'' = int n*)} *''y''* ::= *N 0*;; *wsum* {λ*s. 0* }
⟨*proof*⟩

lemma *example_sound*: $\models_2$ {λ*s. enat* (*2* + *3*∗*n*) + *emb* (*s ''x'' = int n*)} *''y''* ::= *N 0*;; *wsum* {λ*s. 0* }
⟨*proof*⟩

5.6.1 Examples for the use of the VCG

abbreviation *Wsum* ==
{λ*s. 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 $\vdash_2 \{\lambda s.\ enat\ (2 + 3{*}n) + emb\ (s\ ''x'' = int\ n)\}\ ''y'' ::= N\ 0;;$
$wsum\ \{\lambda s.\ 0\ \}$
⟨*proof*⟩

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*
(⊨$_{2'}$ {(*1*_)}/ (_)/ {(*1*_)} *50*) **where**
⊨$_{2'}$ {*P*} *c* {*Q*} ⟷ (∃ *k*>*0*. (∀ *s*. *P s* < ∞ ⟶ (∃ *t p*. ((*c*,*s*) ⇒ *p* ⇓ *t*) ∧
enat k ∗ *P s* ≥ *p* + *enat k* ∗ *Q t*)))

6.2 Hoare Rules

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

Skip: ⊢$_{2'}$ {%*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*: $[\![\vdash_{2'} \{\lambda s.\ enat\ k * P\ s\}c\{\lambda s.\ enat\ k * Q\ s\};\ \ k{>}0\ ]\!] \Longrightarrow$
$\vdash_{2'} \{P\}c\{\ Q\}$
⟨*proof*⟩

inductive
$hoareQ'$ :: $qassn \Rightarrow com \Rightarrow qassn \Rightarrow bool$ $(\vdash_Z (\{(1\_)\}/\ (\_)/\ \{(1\_)\})\ 50)$
where

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

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

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

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

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

Zconseq': $[\![\vdash_Z \{P\}c\{Q\}\ ; \bigwedge s.\ P\ s \leq \ P'\ s\ ; \bigwedge s.\ Q'\ s \leq Q\ s\ ]\!] \Longrightarrow$
$\vdash_Z \{P'\}c\{\ Q'\}$ |

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

lemma *Zconseq*: $[\![\vdash_Z \{P\} c \{Q\}\ ;\ \bigwedge s.\ P\ s \leq enat\ k *\ \ P'\ s\ ;\ \bigwedge s.\ enat\ k * Q'\ s \leq Q\ s;\ k{>}0\ ]\!] \Longrightarrow$
$\vdash_Z \{P'\} c \{\ Q'\}$
⟨*proof*⟩

lemma *ZQ*: $\vdash_Z \{\ P\ \}\ c\ \{\ Q\ \} \Longrightarrow \vdash_{2'} \{\ P\ \}\ c\ \{\ Q\ \}$
⟨*proof*⟩
lemma *QZ*: $\vdash_{2'} \{\ P\ \}\ c\ \{\ Q\ \} \Longrightarrow \vdash_Z \{\ P\ \}\ c\ \{\ Q\ \}$
⟨*proof*⟩

lemma *QZ_iff*: $\vdash_{2'} \{\ P\ \}\ c\ \{\ Q\ \} \longleftrightarrow \vdash_Z \{\ P\ \}\ c\ \{\ Q\ \}$
⟨*proof*⟩

6.3 Soundness

lemma *enatSuc0*[*simp*]: $enat\ (Suc\ 0) * x = x$
⟨*proof*⟩

theorem *hoareQ_sound*: $\vdash_{2'} \{P\} c \{\ Q\} \implies \models_{2'} \{P\} c \{\ Q\}$
⟨*proof*⟩

lemma *conseq'*:
$[\![\vdash_{2'} \{P\}\ c\ \{Q\}\ ;\ \forall s.\ P\ s \leq P'\ s; \forall s.\ Q'\ s \leq Q\ s\ ]\!] \Longrightarrow \vdash_{2'} \{P'\}\ c\ \{Q'\}$
⟨*proof*⟩

lemma *strengthen_pre*:
$[\![\forall s.\ P\ s \leq P'\ s;\ \vdash_{2'} \{P\}\ c\ \{Q\}\ ]\!] \Longrightarrow \vdash_{2'} \{P'\}\ c\ \{Q\}$
⟨*proof*⟩

lemma *weaken_post*:
$[\![\vdash_{2'} \{P\}\ c\ \{Q\};\ \forall s.\ Q\ s \geq Q'\ s\ ]\!] \Longrightarrow \vdash_{2'} \{P\}\ c\ \{Q'\}$
⟨*proof*⟩

lemma *Assign'*: $\forall s.\ P\ s \geq eSuc\ (\ Q(s[a/x])) \Longrightarrow \vdash_{2'} \{P\}\ x ::= a\ \{Q\}$
⟨*proof*⟩

6.4 Completeness

lemma *bigstep_det*: $(c1,\ s) \Rightarrow p1 \Downarrow t1 \Longrightarrow (c1,\ s) \Rightarrow p \Downarrow t \Longrightarrow p1{=}p \wedge t1{=}t$
⟨*proof*⟩

lemma *bigstepT_the_cost*: $(c,\ s) \Rightarrow P \Downarrow T \Longrightarrow (THE\ n.\ \exists\, a.\ (c,\ s) \Rightarrow n \Downarrow a) = P$
⟨*proof*⟩

lemma *bigstepT_the_state*: $(c,\ s) \Rightarrow P \Downarrow T \Longrightarrow (THE\ a.\ \exists\, n.\ (c,\ s) \Rightarrow n \Downarrow a) = T$
⟨*proof*⟩

lemma *SKIPnot*: $(\neg\ (SKIP,\ s) \Rightarrow p \Downarrow t) = (s{\neq}t \vee p{\neq}Suc\ 0)$ ⟨*proof*⟩

lemma *SKIPp*: $(THE\ p.\ \exists\, t.\ (SKIP,\ s) \Rightarrow p \Downarrow t) = Suc\ 0$
⟨*proof*⟩

lemma *SKIPt*: $(THE\ t.\ \exists\, p.\ (SKIP,\ s) \Rightarrow p \Downarrow t) = s$
⟨*proof*⟩

lemma *ASSp*: $(THE\ p.\ Ex\ (big\_step\_t\ (x ::= e,\ s)\ p)) = Suc\ 0$
⟨*proof*⟩

lemma *ASSt*: $(THE\ t.\ \exists\, p.\ (x ::= e,\ s) \Rightarrow p \Downarrow t) = s(x := aval\ e\ s)$
⟨*proof*⟩

lemma *ASSnot*: $(\ \neg\ (x ::= e,\ s) \Rightarrow p \Downarrow t\ ) = (p{\neq}Suc\ 0 \vee t{\neq}s(x := aval\ e\ s))$
⟨*proof*⟩

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

lemma *wpQ_skip*[*simp*]: $wp_Q\ SKIP\ Q = (\%s.\ eSuc\ (Q\ s))$
⟨*proof*⟩

lemma *wpQ_ass*[*simp*]: $wp_Q\ (x ::= e)\ Q = (\lambda s.\ eSuc\ (Q\ (s(x := aval\ e\ s))))$
⟨*proof*⟩

lemma *wpt_Seq*[*simp*]: $wp_Q\ (c_1;;c_2)\ Q = wp_Q\ c_1\ (wp_Q\ c_2\ Q)$
⟨*proof*⟩

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

lemma *hoareQ_inf*: $\vdash_{2'} \{\%s.\ \infty\}\ c\ \{\ Q\}$
⟨*proof*⟩

lemma assumes *b*: *bval b s*
shows *wpQ_WhileTrue*: $wp_Q\ c\ (wp_Q\ (WHILE\ b\ DO\ c)\ Q)\ s\ +\ 1 \leq wp_Q\ (WHILE\ b\ DO\ c)\ Q\ s$
⟨*proof*⟩

lemma assumes *b*: $\sim$ *bval b s*
shows *wpQ_WhileFalse*: $Q\ s\ +\ 1 \leq wp_Q\ (WHILE\ b\ DO\ c)\ Q\ s$
⟨*proof*⟩

lemma *wpQ_is_pre*: $\vdash_{2'} \{wp_Q\ c\ Q\}\ c\ \{\ Q\}$
⟨*proof*⟩

lemma *wpQ_is_pre'*: $\vdash_{2'} \{wp_Q\ c\ (\%s.\ enat\ k * Q\ s\ )\}\ c\ \{(\%s.\ enat\ k * Q\ s\ )\}$
⟨*proof*⟩

lemma *wpQ_is_weakestprePotential1*: $\models_{2'} \{P\}c\{Q\} \Longrightarrow (\exists\, k{>}0.\ \forall\, s.\ wp_Q\ c\ (\%s.\ enat\ k{*}\ Q\ s)\ s \leq enat\ k * P\ s)$
⟨*proof*⟩

theorem *hoareQ_complete*: $\models_{2'} \{P\}c\{Q\} \Longrightarrow \vdash_{2'} \{P\}c\{\ Q\}$
⟨*proof*⟩

theorem *hoareQ_complete'*: $\models_{2'} \{P\}c\{Q\} \Longrightarrow \vdash_{2'} \{P\}c\{\ Q\}$
⟨*proof*⟩

corollary *hoareQ_sound_complete*: $\vdash_{2'} \{P\}c\{Q\} \longleftrightarrow \models_{2'} \{P\}c\{\ Q\}$
⟨*proof*⟩

6.5 Example

lemma fixes X::*int* **assumes** $0 < X$ **shows**
Z: *eSuc* (*enat* (*nat* (2 * X) * *nat* (2 * X))) ≤ *enat* (5 * (*nat* (X * X)))
⟨*proof*⟩

lemma *weakenpre*: $[\![\vdash_{2'} \{P\}c\{Q\}\ ;\quad (\forall s.\ P\ s \leq\ P'\ s)\ ]\!] \Longrightarrow$
$\vdash_{2'} \{P'\}c\{\ Q\}$ ⟨*proof*⟩

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}
⟨*proof*⟩

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}
⟨*proof*⟩

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}
⟨*proof*⟩

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*: $k>0 \Longrightarrow D\geq k \Longrightarrow mydiv\ D\ k = Suc\ (mydiv\ (D-k)\ k)$

⟨*proof*⟩

lemma *mydivcode1*: $mydiv\ 0\ k = 0$
⟨*proof*⟩

lemma *mydivcode2*: $k>0 \Longrightarrow 0<D \Longrightarrow D<k \Longrightarrow mydiv\ D\ k = Suc\ 0$
⟨*proof*⟩

lemma *mydiv_mono*: $a\leq b \Longrightarrow mydiv\ a\ k \leq mydiv\ b\ k$ ⟨*proof*⟩

lemma *mydiv_cancel*: $0 < k \Longrightarrow mydiv\ (k * i)\ k = i$
⟨*proof*⟩

lemma assumes *k*: $k>0$ **and** *B*: $B \leq k*A$
shows *mydiv_le_E*: $mydiv\ B\ k \leq A$
⟨*proof*⟩
lemma *mydiv_mult_leq*: $0 < k \Longrightarrow l\leq k \Longrightarrow mydiv\ (l*A)\ k \leq A$
⟨*proof*⟩

lemma *mydiv_cancel3*: $0 < k \Longrightarrow i \leq k * mydiv\ i\ k$
⟨*proof*⟩

definition *ediv a k* = (*if a*=∞ *then* ∞ *else enat* (*mydiv* (*THE i*. *a*=*enat i*) *k*))

lemma *ediv_enat*[*simp*]: $ediv\ (enat\ a)\ k = enat\ (mydiv\ a\ k)$
⟨*proof*⟩
lemma *ediv_mydiv*[*simp*]: $ediv\ (enat\ a)\ k \leq enat\ f \longleftrightarrow mydiv\ a\ k \leq f$
⟨*proof*⟩

lemma *ediv_mono*: $a\leq b \Longrightarrow ediv\ a\ k \leq ediv\ b\ k$
⟨*proof*⟩

lemma *ediv_cancel2*: $k>0 \Longrightarrow ediv\ (enat\ k * x)\ k = x$
⟨*proof*⟩

lemma *ediv_cancel3*: $k>0 \Longrightarrow A \leq enat\ k * ediv\ A\ k$
⟨*proof*⟩

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* (λ*s*. *enat k*∗ *Q s*) *s* ≤ *enat k*∗ *Q s*) ∧ (∀ *s*. *enat k*∗ *Q s* ≤ *pre C* (λ*s*. *enat k*∗ *Q s*) *s*))

6.6.3 Soundness of VCG

lemma *vc_sound*: *vc C Q* ⟹ ⊢$_{2'}$ {*pre C Q*} *strip C* { *Q* }
⟨*proof*⟩

lemma *vc_sound'*: $[\![vc\ C\ Q\ ;\ (\forall s.\ pre\ C\ Q\ s \leq P\ s)\ ]\!] \Longrightarrow\ \vdash_{2'} \{P\}\ strip\ C\ \{\ Q\ \}$
⟨*proof*⟩

lemma *vc_sound''*: $[\![vc\ C\ Q'\ ;\ (\forall s.\ \ pre\ C\ Q'\ s \leq k * P\ s)\ \ ;\ (\bigwedge s.\ enat\ k * Q\ s \leq Q'\ s);\ k{>}0\ ]\!] \Longrightarrow\ \vdash_{2'} \{P\}\ strip\ C\ \{\ Q\ \}$
⟨*proof*⟩

6.6.4 Completeness

lemma *pre_mono*: **assumes** $\bigwedge s.\ P'\ s \leq P\ s$
shows $\bigwedge s.\ pre\ C\ P'\ s \leq pre\ C\ P\ s$
⟨*proof*⟩

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

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*⟩

lemma $\vdash_Z \{\ P\ \}\ c\ \{\ Q\ \} \Longrightarrow \exists C.\ strip\ C = c \wedge vc\ C\ Q \wedge (\forall s.\ pre\ C\ Q\ s \leq P\ s)$
(**is** $\_ \Longrightarrow\ \ \exists C.\ ?G\ P\ c\ Q\ C$)
⟨*proof*⟩

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*: ⊢$_{2'}$ {λ*s*. *enat* (*2* + *3*∗*n*) + *emb* (*s* ″*x*″ = *int n*)} ″*y*″ ::= *N 0*;; *wsum* {λ*s*. *0* }
⟨*proof*⟩

lemma *example_sound*: ⊨$_{2'}$ {λ*s*. *enat* (*2* + *3*∗*n*) + *emb* (*s* ″*x*″ = *int n*)} ″*y*″ ::= *N 0*;; *wsum* {λ*s*. *0* }
⟨*proof*⟩

schematic_goal ⊢$_{2'}$ {λ*s*. *?A s* + *emb* (*s* ″*x*″ = *int n*)} ″*y*″ ::= *N 0*;; *wsum* {λ*s*. *0* }
⟨*proof*⟩

6.7.1 Example for VCG

lemma ⊢$_{2'}$ {λ*s*. *1*} *SKIP* ;; *SKIP* {λ*s*. *0* }
⟨*proof*⟩

lemma *hoareQ_Seq_assoc*: ⊢$_{2'}$ {*P*} *A*;; *B*;; *C* {*Q*} = (⊢$_{2'}$ {*P*} *A*;; (*B*;; *C*) {*Q*})
⟨*proof*⟩

lemma ⊢$_{2'}$ {λ*s*. *1*} *SKIP* ;; *SKIP* ;; *SKIP* {λ*s*. *0* }
⟨*proof*⟩

abbreviation *Wsum* ==
{λ*s*. *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 $\vdash_{2'} \{\lambda s.\ enat\ (2\ +\ 3{*}n)\ +\ emb\ (s\ ''x''\ =\ int\ n)\}\ ''y''\ ::=\ N\ 0;;$
$wsum\ \{\lambda s.\ 0\ \}$
⟨*proof*⟩

lemma assumes *n0*: $n{>}0$ **shows** $\vdash_{2'} \{\lambda s.\ enat\ (n\ )\ +\ emb\ (s\ ''x''\ =\ int\ n)\}\ ''y''\ ::=\ N\ 0;;\ wsum\ \{\lambda s.\ 0\ \}$
⟨*proof*⟩

lemma $\vdash_{2'} \{\lambda s.\ enat\ (n{+}1)\ +\ emb\ (s\ ''x''\ =\ int\ n)\}\ ''y''\ ::=\ N\ 0;;\ wsum\ \{\lambda s.\ 0\ \}$
⟨*proof*⟩

abbreviation *Wsum1 z* ==
$\{\lambda s.\ enat\ (z\ *\ nat\ (s\ ''x''))\}\ WHILE\ Less\ (N\ 0)\ (V\ ''x'')$
$DO\ (''y''\ ::=\ Plus\ (V\ ''y'')\ (V\ ''x'');;$
$\quad ''x''\ ::=\ Plus\ (V\ ''x'')\ (N\ (-\ 1)))$

abbreviation *Wsum2 n vier* ==
$\{\lambda s.\ enat\ (vier\ *\ (nat\ (s\ ''x'')\ +\ \ n\ +\ 1))\ \}\ WHILE\ Less\ (N\ 0)\ (V\ ''x'')$
$DO\ (''y''\ ::=\ Plus\ (V\ ''y'')\ (V\ ''x'');;$
$\quad ''x''\ ::=\ Plus\ (V\ ''x'')\ (N\ (-\ 1)))$

end
theory *QuantK_Sqrt*
imports *QuantK_VCG HOL−Library.Discrete*
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*=
$''l''::=\ N\ 0\ ;;$
$''m''\ ::=\ N\ 0\ ;;$
$''r''::=\ Plus\ (N\ 1)\ (V\ ''x'');;$
$(WHILE\ (Less\ (Plus\ (N\ 1)\ (V\ ''l''))\ (V\ ''r''))$
$\quad DO\ (''m''\ ::=\ (Div\ (Plus\ (V\ ''l'')\ (V\ ''r''))\ (N\ 2))\ ;;$

(*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*: $\bigwedge s\ k.\ 1 + s\ ''x'' = 2\ \hat{}\ k \Longrightarrow 5 * k \leq 96 + 100 * Discrete.log\ (nat\ (s\ ''x''))$
⟨*proof*⟩

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: $P = (\lambda s.\ \uparrow (\ (\exists k.\ 1 + s\ ''x'' = 2\ \hat{}\ k)) + (Discrete.log\ (nat\ (\ s\ ''x'')) + 1))$ **and**
$Q[simp]$: $Q = (\lambda\_.\ 0)$
shows $\vdash_{2'}\ \{P\}\ c\ \{Q\}$
⟨*proof*⟩

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* ⟨*proof*⟩

lemma *part_emb*[*simp*]: *dom ps* = *UNIV* ⟹ *part* (*emb ps q*) = *ps* ⟨*proof*⟩

lemma *dom_part*[*simp*]: *dom* (*part s*) = *UNIV* ⟨*proof*⟩

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*⟩

lemma *paval′_aval*: *paval′ a ps* = *Some v* ⟹ *aval a* (*emb ps s*) = *v*
⟨*proof*⟩

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*)
⟨*proof*⟩

lemma *paval'_paval*: *vars a* $\subseteq$ *dom ps* $\Longrightarrow$ *paval' a ps* = *Some* (*paval a ps*)
⟨*proof*⟩

lemma *paval_paval'*: *paval' a ps* = *Some v* $\Longrightarrow$ *paval a ps* = *v*
⟨*proof*⟩

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* $\subseteq$ *dom ps* $\Longrightarrow$ *pbval' a ps* = *Some* (*pbval a ps*)
⟨*proof*⟩

lemma *paval_aval_vars*: *vars a* $\subseteq$ *dom ps* $\Longrightarrow$ *paval a ps* = *aval a* (*emb ps s*)
⟨*proof*⟩

lemma *pbval_bval_vars*: *vars b* $\subseteq$ *dom ps* $\Longrightarrow$ *pbval b ps* = *bval b* (*emb ps s*)
⟨*proof*⟩

lemma *paval'dom*: *paval' a ps* = *Some v* $\Longrightarrow$ *vars a* $\subseteq$ *dom ps*
⟨*proof*⟩

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
⟨*proof*⟩

end

lemma *sep_disj_prod_commute*[*simp*]: (*ps*, *0*) ## (*0*, *n*) ⟷ (*0*, *n*) ## (*ps*, *0*) ⟨*proof*⟩

lemma *sep_disj_prod_conv*[*simp*]: (*a*, *x*) ## (*b*, *y*) = (*a*##*b* ∧ *x*##*y*) ⟨*proof*⟩

lemma *sep_plus_prod_conv*[*simp*]: (*ps*, *n*) + (*ps'*, *n'*) = (*ps* + *ps'*, *n* + *n'*) ⟨*proof*⟩

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*) ⟨*proof*⟩

instantiation *nat* :: *sep_algebra*
begin

definition
sep_disj_nat_def[*simp*]: *sep_disj* (*m1*::*nat*) *m2* ≡ *True*

instance
⟨*proof*⟩
end

lemma
fixes $h :: nat$
shows $(P ** Q ** H)\ h = (Q ** H ** P)\ h$
⟨*proof*⟩

lemma
fixes $h :: ('a::sep\_algebra) * ('b::sep\_algebra)$
shows $(P ** Q ** H)\ h = (Q ** H ** P)\ h$
⟨*proof*⟩

lemma
fixes $h :: nat * nat$
shows $(P ** Q ** H)\ h = (Q ** H ** P)\ h$
⟨*proof*⟩

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* □)
⟨*proof*⟩

lemma *pure_alt*: ⟨Φ⟩ = (↑Φ ** *sep_true*)
⟨*proof*⟩

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*]:
$((a \wedge* b) \wedge* c) = (a \wedge* b \wedge* c)$
$(NO\_PURE\ X\ a) \Longrightarrow (a ** b) = (b ** a)$
$(NO\_PURE\ X\ b) \Longrightarrow (b \wedge* a \wedge* c) = (a \wedge* b \wedge* c)$
$(Q ** \langle P \rangle) = (\langle P \rangle ** Q)$
$(Q ** \uparrow P) = (\uparrow P ** Q)$
$NO\_PURE\ X\ Q \Longrightarrow (Q ** \langle P \rangle ** F) = (\langle P \rangle ** Q ** F)$
$NO\_PURE\ X\ Q \Longrightarrow (Q ** \uparrow P ** F) = (\uparrow P ** Q ** F)$
⟨*proof*⟩

lemma *sep_combine1*[*simp*]:
$(\uparrow P ** \uparrow Q) = \uparrow(P \wedge Q)$
$(\langle P \rangle ** \langle Q \rangle) = \langle P \wedge Q \rangle$
$(\uparrow P ** \langle Q \rangle) = \langle P \wedge Q \rangle$
$(\langle P \rangle ** \uparrow Q) = \langle P \wedge Q \rangle$
⟨*proof*⟩

lemma *sep_combine2*[*simp*]:
$(\uparrow P ** \uparrow Q ** F) = (\uparrow(P \wedge Q) ** F)$
$(\langle P \rangle ** \langle Q \rangle ** F) = (\langle P \wedge Q \rangle ** F)$
$(\uparrow P ** \langle Q \rangle ** F) = (\langle P \wedge Q \rangle ** F)$
$(\langle P \rangle ** \uparrow Q ** F) = (\langle P \wedge Q \rangle ** F)$
⟨*proof*⟩

lemma *sep_extract_pure*[*simp*]:
$NO\_MATCH\ True\ P \Longrightarrow (\langle P \rangle ** Q)\ h = (P \wedge (sep\_true ** Q)\ h)$
$(\uparrow P ** Q)\ h = (P \wedge Q\ h)$
$\uparrow True = \square$
$\uparrow False = sep\_false$
⟨*proof*⟩

lemma *sep_pure_front2*[*simp*]:
$(\uparrow P ** A ** \uparrow Q ** F) = (\uparrow(P \wedge Q) ** F ** A)$
⟨*proof*⟩

lemma *ex_h_simps*[*simp*]:
$Ex\ (\uparrow \Phi) \longleftrightarrow \Phi$
$Ex\ (\uparrow \Phi ** P) \longleftrightarrow (\Phi \wedge Ex\ P)$
⟨*proof*⟩

lemma
fixes $h :: ('a \Rightarrow 'b\ option) * nat$
shows $(P ** Q ** H)\ h = (Q ** H ** P)\ h$

⟨*proof*⟩

lemma *map_le_substate_conv*: *map_le* = *sep_substate*
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *restrictE*: *restrict S s1* = *restrict S s2* ⟹ *s1* = *s2 on S*
⟨*proof*⟩

lemma *dom_restrict*[*simp*]: *dom* (*restrict S s*) = *S*
⟨*proof*⟩

lemma *restrict_less_part*: *restrict S t* ≼ *part t*
⟨*proof*⟩

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* ⟨*proof*⟩

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*))
⟨*proof*⟩

lemma *paval_aval2*: *vars a* ⊆ *dom ps* ⟹ *ps* ⪯ *part s* ⟹ *paval a ps* = *aval a s*
⟨*proof*⟩

lemma **fixes** *ps*:: *partstate*
and *s*::*state*
assumes *vars a* ⊆ *dom ps ps* ⪯ *part s*
shows *emb_update2*: *emb* (*ps*(*x* ↦ *paval a ps*)) *s* = (*emb ps s*)(*x* := *aval a* (*emb ps s*))
⟨*proof*⟩

7.2.2 Big step Semantics on partial states

inductive
big_step_t_part :: *com* × *partstate* ⇒ *nat* ⇒ *partstate* ⇒ *bool* (_ ⇒$_A$ _ ⇓ _ *55*)
where
Skip: (*SKIP*,*s*) ⇒$_A$ *Suc 0* ⇓ *s* |
Assign: ⟦ *vars a* ∪ {*x*} ⊆ *dom ps*; *paval a ps* = *v* ; *ps'* = *ps*(*x* ↦ *v*) ⟧ ⟹ (*x* ::= *a*,*ps*) ⇒$_A$ *Suc 0* ⇓ *ps'* |

Seq: ⟦ (*c1*,*s1*) ⇒$_A$ *x* ⇓ *s2*; (*c2*,*s2*) ⇒$_A$ *y* ⇓ *s3* ; *z*=*x*+*y* ⟧ ⟹ (*c1*;;*c2*, *s1*) ⇒$_A$ *z* ⇓ *s3* |

IfTrue: ⟦ *vars b* ⊆ *dom ps* ; *dom ps'* = *dom ps* ; *pbval b ps*; (*c1*,*ps*) ⇒$_A$ *x* ⇓ *ps'*; *y*=*x*+*1* ⟧ ⟹ (*IF b THEN c1 ELSE c2*, *ps*) ⇒$_A$ *y* ⇓ *ps'* |
IfFalse: ⟦ *vars b* ⊆ *dom ps* ; *dom ps'* = *dom ps* ; ¬ *pbval b ps*; (*c2*,*ps*) ⇒$_A$ *x* ⇓ *ps'*; *y*=*x*+*1* ⟧ ⟹ (*IF b THEN c1 ELSE c2*, *ps*) ⇒$_A$ *y* ⇓ *ps'* |
WhileFalse: ⟦ *vars b* ⊆ *dom s*; ¬ *pbval b s* ⟧ ⟹ (*WHILE b DO c*,*s*) ⇒$_A$ *Suc 0* ⇓ *s* |
WhileTrue: ⟦ *pbval b s1*; *vars b* ⊆ *dom s1*; (*c*,*s1*) ⇒$_A$ *x* ⇓ *s2*; (*WHILE b DO c*, *s2*) ⇒$_A$ *y* ⇓ *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*!]: $(SKIP,s) \Rightarrow_A x \Downarrow t$
thm *Skip_tE3*
inductive_cases *Assign_tE3*[*elim*!]: $(x ::= a,s) \Rightarrow_A p \Downarrow t$
thm *Assign_tE3*
inductive_cases *Seq_tE3*[*elim*!]: $(c1;;c2,s1) \Rightarrow_A p \Downarrow s3$
thm *Seq_tE3*
inductive_cases *If_tE3*[*elim*!]: $(IF\ b\ THEN\ c1\ ELSE\ c2,s) \Rightarrow_A x \Downarrow t$
thm *If_tE3*
inductive_cases *While_tE3*[*elim*]: $(WHILE\ b\ DO\ c,s) \Rightarrow_A x \Downarrow t$
thm *While_tE3*

lemmas *big_step_t_part_induct* = *big_step_t_part.induct*[*split_format*(*complete*)]

lemma *big_step_t3_post_dom_conv*: $(c,ps) \Rightarrow_A t \Downarrow ps' \Longrightarrow dom\ ps' = dom\ ps$
⟨*proof*⟩

lemma *add_update_distrib*: $ps1\ x1 = Some\ y \Longrightarrow ps1\ \#\#\ ps2 \Longrightarrow vars\ x2 \subseteq dom\ ps1 \Longrightarrow ps1(x1 \mapsto paval\ x2\ ps1) + ps2 = (ps1 + ps2)(x1 \mapsto paval\ x2\ ps1)$
⟨*proof*⟩

lemma *paval_extend*: $ps1\ \#\#\ ps2 \Longrightarrow vars\ a \subseteq dom\ ps1 \Longrightarrow paval\ a\ (ps1 + ps2) = paval\ a\ ps1$
⟨*proof*⟩

lemma *pbval_extend*: $ps1\ \#\#\ ps2 \Longrightarrow vars\ b \subseteq dom\ ps1 \Longrightarrow pbval\ b\ (ps1 + ps2) = pbval\ b\ ps1$
⟨*proof*⟩

lemma *Framer*: $(C,\ ps1) \Rightarrow_A m \Downarrow ps1' \Longrightarrow ps1\ \#\#\ ps2 \Longrightarrow (C,\ ps1 + ps2) \Rightarrow_A m \Downarrow\ ps1'+ps2$
⟨*proof*⟩

lemma *Framer2*: $(C,\ ps1) \Rightarrow_A m \Downarrow ps1' \Longrightarrow ps1\ \#\#\ ps2 \Longrightarrow ps = ps1 + ps2 \Longrightarrow ps' = ps1' + ps2 \Longrightarrow (C,\ ps) \Rightarrow_A m \Downarrow ps'$
⟨*proof*⟩

7.2.3 Relation to BigStep Semantic on full states

lemma *paval_aval_part*: $paval\ a\ (part\ s) = aval\ a\ s$
⟨*proof*⟩
lemma *pbval_bval_part*: $pbval\ b\ (part\ s) = bval\ b\ s$
⟨*proof*⟩

lemma *part_paval_aval*: $part\ (s(x := aval\ a\ s)) = part\ s(x \mapsto paval\ a\ (part\ s))$
⟨*proof*⟩

lemma *full_to_part*: $(C,\ s) \Rightarrow m \Downarrow s' \Longrightarrow\ (C,\ part\ s) \Rightarrow_A m \Downarrow part\ s'$
⟨*proof*⟩

lemma *part_to_full'*: $(C,\ ps) \Rightarrow_A m \Downarrow ps' \Longrightarrow (C,\ emb\ ps\ s) \Rightarrow m \Downarrow emb\ ps'\ s$
⟨*proof*⟩

lemma *part_to_full*: $(C,\ part\ s) \Rightarrow_A m \Downarrow part\ s' \Longrightarrow (C,\ s) \Rightarrow m \Downarrow s'$
⟨*proof*⟩

lemma *part_full_equiv*: $(C,\ s) \Rightarrow m \Downarrow s' \longleftrightarrow (C,\ part\ s) \Rightarrow_A m \Downarrow part\ s'$
⟨*proof*⟩

7.2.4 more properties

lemma *big_step_t3_gt0*: $(C,\ ps) \Rightarrow_A x \Downarrow ps' \Longrightarrow x > 0$
⟨*proof*⟩

lemma *big_step_t3_same*: $(C,\ ps) \Rightarrow_A m \Downarrow ps' ==>\ ps = ps'\ on\ UNIV - lvars\ C$
⟨*proof*⟩

lemma *avalDirekt3_correct*: $(x ::= N\ v,\ ps) \Rightarrow_A m \Downarrow ps' \Longrightarrow paval'\ a\ ps = Some\ v \Longrightarrow\ (x ::= a,\ ps) \Rightarrow_A m \Downarrow ps'$
⟨*proof*⟩

7.3 Partial State

lemma
fixes $h :: (vname \Rightarrow val\ option) * nat$
shows $(P ** Q ** H)\ h = (Q ** H ** P)\ h$
⟨*proof*⟩

lemma *separate_othogonal_commuted'*: **assumes**
$\bigwedge ps\ n.\ P\ (ps,n) \Longrightarrow ps = 0$
$\bigwedge ps\ n.\ Q\ (ps,n) \Longrightarrow n = 0$
shows $(P ** Q)\ s \longleftrightarrow P\ (0,snd\ s) \wedge Q\ (fst\ s,0)$
⟨*proof*⟩

lemma *separate_othogonal_commuted*: **assumes**
$\bigwedge ps\ n.\ P\ (ps,n) \Longrightarrow ps = 0$
$\bigwedge ps\ n.\ Q\ (ps,n) \Longrightarrow n = 0$
shows $(P ** Q)\ (ps,n) \longleftrightarrow P\ (0,n) \wedge Q\ (ps,0)$
⟨*proof*⟩

lemma *separate_othogonal*: **assumes**
$\bigwedge ps\ n.\ P\ (ps,n) \Longrightarrow n = 0$
$\bigwedge ps\ n.\ Q\ (ps,n) \Longrightarrow ps = 0$
shows $(P ** Q)\ (ps,n) \longleftrightarrow P\ (ps,0) \wedge Q\ (0,n)$
⟨*proof*⟩

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*⟩

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*:
$NO\_MATCH\ (\$X)\ A \Longrightarrow (\$B ** A) = (A ** \$B)$
$(\$X ** \$\,Y) = \$(X+Y)$
⟨*proof*⟩

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** $m \leq n$ ⟨*proof*⟩

lemma [*simp*]: $(Q ** \$0) = Q$ ⟨*proof*⟩

definition *embP* :: $(partstate \Rightarrow bool) \Rightarrow partstate \times nat \Rightarrow bool$ **where**
$embP\ P = (\%(s,n).\ P\ s \wedge n = 0)$

lemma *orthogonal_split*: **assumes** $(embP\ Q \wedge* \$\ n) = (embP\ P \wedge* \$\ m)$

shows $(Q = P \wedge n = m) \vee Q = (\lambda s.\ False) \wedge P = (\lambda s.\ False)$
⟨*proof*⟩

lemma *F*: **assumes** $(embP\ Q \wedge* \$\ n) = (embP\ P \wedge* \$\ m)$
obtains (*blub*) $Q = P$ **and** $n = m$ |
(*da*) $Q = (\lambda s.\ False)$ **and** $P = (\lambda s.\ False)$
⟨*proof*⟩

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** $P = (\lambda s.\ False)$
⟨*proof*⟩

definition *pointsto* :: *vname* ⇒ *val* ⇒ *assn2* $(\_ \hookrightarrow \_\ [56,51]\ 56)$ **where**
$v \hookrightarrow n = (\%(s,c).\ s = [v \mapsto n] \wedge c{=}0)$

notation *pred_ex* (**binder** ∃ *10*)

definition *maps_to_ex* :: *vname* ⇒ *assn2* $(\_ \hookrightarrow -\ [56]\ 56)$
where $x \hookrightarrow - \equiv \exists y.\ x \hookrightarrow y$

fun *lmaps_to_ex* :: *vname set* ⇒ *assn2* **where**
$lmaps\_to\_ex\ xs = (\%(s,c).\ dom\ s = xs \wedge c = 0)$

lemma $(x \hookrightarrow -)\ (s,n) \Longrightarrow x \in dom\ s$
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *Frame'_conv*: *Frame P Q F* = *Frame'* (*P* ** □) □ (*Q* ** □) *F*
⟨*proof*⟩

lemma *Frame'I*: *Frame'* (*P* ** □) □ (*Q* ** □) *F* ⟹ *cnv F F'* ⟹ *Frame P Q F'*
⟨*proof*⟩

lemma *FrameD*: **assumes** *Frame P Q F P s*
shows (*F* ** *Q*) *s*
⟨*proof*⟩

lemma *Frame'_match*: *Frame'* (*P* ** *P'*) □ *Q F* ⟹ *Frame'* (*x* ↪ *v* ** *P*) *P'* (*x* ↪ *v* ** *Q*) *F*
⟨*proof*⟩

lemma *R*: **assumes** ⋀*s*. (*A imp B*) *s* **shows** ((*A* ** $*n*) *imp* (*B* ** $*n*)) *s*

⟨*proof*⟩

lemma *Frame'_matchdollar*: **assumes** *Frame'* (*P* ** *P'* ** $(*n*−*m*)) □ *Q F* **and** *nm*: *n*≥*m*
shows *Frame'* ($*n* ** *P*) *P'* ($*m* ** *Q*) *F*
⟨*proof*⟩

lemma *Frame'_nomatch*: *Frame' P* (*p* ** *P'*) (*x* ↪ *v* ** *Q*) *F*⟹ *Frame'* (*p* ** *P*) *P'* (*x* ↪ *v* ** *Q*) *F*
⟨*proof*⟩

lemma *Frame'_nomatchempty*: *Frame' P P' (x ↪ v ∗∗ Q) F⟹ Frame' (□ ∗∗ P) P' (x ↪ v ∗∗ Q) F*
⟨*proof*⟩

lemma *Frame'_end*: *Frame' P □ □ P*
⟨*proof*⟩

schematic_goal *Frame (x ↪ v1 ∧∗ y ↪ v2) (x ↪ ?v) ?F*
⟨*proof*⟩

schematic_goal *Frame (x ↪ v1 ∧∗ y ↪ v2) (y ↪ ?v) ?F*
⟨*proof*⟩

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 ↪ v1 ∧∗ y ↪ v2) (y ↪ ?v) ?F*
⟨*proof*⟩

schematic_goal *Frame (x ↪ v1 ∗∗ P ∗∗ □ ∗∗ y ↪ v2 ∧∗ z ↪ v2 ∗∗ Q) (z ↪ ?v ∗∗ y ↪ ?v2) ?F*
⟨*proof*⟩

schematic_goal *1 ≤ v ⟹ Frame ($ (2 ∗ v) ∧∗ ″x″ ↪ int v) ($ 1 ∧∗ ″x″ ↪ ?d) ?F*
⟨*proof*⟩

schematic_goal $0 < v \Longrightarrow$ *Frame* (\$ (*2* $*$ *v*) $\wedge*$ $''x''$ $\hookrightarrow$ *int v*) (\$ *1* $\wedge*$ $''x''$ $\hookrightarrow$ *?d*) *?F*
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *symeval_v*: **assumes** *Frame P* (*x* $\hookrightarrow$ *v*) *F*
shows *symeval P* (*V x*) *v*
⟨*proof*⟩

lemma *symeval_plus*: **assumes** *symeval P e1 v1 symeval P e2 v2*
shows *symeval P* (*Plus e1 e2*) (*v1*+*v2*)
⟨*proof*⟩

lemma *symevalb_c*: *symevalb P* (*Bc v*) *v*
⟨*proof*⟩

lemma *symevalb_and*: **assumes** *symevalb P e1 v1 symevalb P e2 v2*
shows *symevalb P* (*And e1 e2*) (*v1* $\wedge$ *v2*)
⟨*proof*⟩

lemma *symevalb_not*: **assumes** *symevalb P e v*
shows *symevalb P* (*Not e*) ($\neg$ *v*)
⟨*proof*⟩

lemma *symevalb_less*: **assumes** *symeval P e1 v1 symeval P e2 v2*
shows *symevalb P* (*Less e1 e2*) (*v1* < *v2*)
⟨*proof*⟩

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*

⟨*proof*⟩

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*
(⊨$_3$ {(*1*_)}/ (_)/ { (*1*_)} *50*) **where**
⊨$_3$ { *P* } *c* { *Q* } ⟷
(∀ *ps n*. *P* (*ps*,*n*)
⟶ (∃ *ps′ m*. ((*c*,*ps*) ⇒$_A$ *m* ⇓ *ps′*)
∧ *n*≥*m* ∧ *Q* (*ps′*, *n*−*m*)))

lemma *alternative*: ⊨$_3$ { *P* } *c* { *Q* } ⟷
(∀ *ps n*. *P* (*ps*,*n*)
⟶ (∃ *ps′ t n′*. ((*c*,*ps*) ⇒$_A$ *t* ⇓ *ps′*)
∧ *n*=*n′*+*t* ∧ *Q* (*ps′*, *n′*)))
⟨*proof*⟩

8.2 Hoare Rules

inductive
hoareT3 :: *assn2* ⇒ *com* ⇒ *assn2* ⇒ *bool* (⊢$_3$ ({(*1*_)}/ (_)/ { (*1*_)}) *50*)
where

Skip: ⊢$_3$ {$*1*} *SKIP* { $*0*} |

Assign: ⊢$_3$ {*lmaps_to_expr_x x a v* ** $*1*} *x*::=*a* { (%(*s*,*c*). *dom s* = *vars a* − {*x*} ∧ *c* = *0*) ** *x* ↪ *v* } |

If: ⟦ ⊢$_3$ { λ(*s*,*n*). *P* (*s*,*n*) ∧ *lmaps_to_axpr b True s* } c_1 { *Q* };
⊢$_3$ { λ(*s*,*n*). *P* (*s*,*n*) ∧ *lmaps_to_axpr b False s* } c_2 { *Q* } ⟧
⟹ ⊢$_3$ { (λ(*s*,*n*). *P* (*s*,*n*) ∧ *vars b* ⊆ *dom s*) ** $*1*} *IF b THEN* c_1 *ELSE* c_2 { *Q*} |

Frame: ⟦ ⊢$_3$ { *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: $[\![\ \vdash_3 \{\ P ** \$m\ \}\ C\ \{\ Q\ ** \$n\ \};\ n \leq m\ ]\!]$
$\Longrightarrow \vdash_3 \{\ P ** \$(m-n)\ \}\ C\ \{\ Q\ \}$ |

constancy: $[\![\ \vdash_3 \{\ P\ \}\ C\ \{\ Q\ \};\ \bigwedge ps\ ps'.\ ps = ps'\ on\ UNIV - lvars\ C \Longrightarrow R\ ps = R\ ps'\ ]\!]$
$\Longrightarrow \vdash_3 \{\ \%(ps,n).\ P\ (ps,n) \wedge R\ ps\ \}\ C\ \{\ \%(ps,n).\ Q\ (ps,n) \wedge R\ ps\ \}$ |

Assign''': $\vdash_3 \{\ \$1 ** (x \hookrightarrow ds)\ \}\ x ::= (N\ v)\ \{\ (x \hookrightarrow v)\ \}$ |

Assign'''': $[\![\ symeval\ P\ a\ v; \vdash_3 \{P\}\ x ::= (N\ v)\ \{Q'\}\ ]\!] \Longrightarrow \vdash_3 \{P\}\ x ::= a\ \{Q'\}$ |

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 $\vdash_3 \{\ P'\ \}\ C\ \{\ Q ** F\ \}$
⟨*proof*⟩

lemma *strengthen_post*: **assumes** $\vdash_3 \{P\}c\{Q\}\ \bigwedge s.\ Q\ s \Longrightarrow Q'\ s$
shows $\vdash_3 \{P\}c\{\ Q'\}$
⟨*proof*⟩

lemma *weakenpre*: $[\![\vdash_3 \{P\}c\{Q\} ; \bigwedge s.\ P'\ s \Longrightarrow P\ s\ ]\!] \Longrightarrow$
$\vdash_3 \{P'\}c\{\ Q\}$
⟨*proof*⟩

8.3 Soundness Proof

lemma *exec_preserves_disj*: $(c,ps) \Rightarrow_A t \Downarrow ps' \Longrightarrow ps'' \#\# ps \Longrightarrow ps'' \#\# ps'$
⟨*proof*⟩

lemma *FrameRuleSound*: **assumes** $\models_3 \{\ P\ \}\ C\ \{\ Q\ \}$
shows $\models_3 \{\ P ** F\ \}\ C\ \{\ Q ** F\ \}$
⟨*proof*⟩

theorem *hoare3_sound*: **assumes** $\vdash_3 \{\ P\ \}c\{\ Q\ \}$
shows $\models_3 \{\ P\ \}\ c\ \{\ Q\ \}$ ⟨*proof*⟩

8.4 Completeness

definition *wp3* :: *com* ⇒ *assn2* ⇒ *assn2* (wp_3) **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 *wp3_SKIP*[*simp*]: $wp_3\ SKIP\ Q = (Q ** \$1)$
⟨*proof*⟩

lemma *wp3_Assign*[*simp*]: $wp_3\ (x ::= e)\ Q = ((\lambda(ps,t).\ vars\ e \cup \{x\} \subseteq dom\ ps \wedge Q\ (ps(x \mapsto paval\ e\ ps),t)) ** \$1)$
⟨*proof*⟩

lemma *wpt_Seq*[*simp*]: $wp_3\ (c_1;;c_2)\ Q = wp_3\ c_1\ (wp_3\ c_2\ Q)$
⟨*proof*⟩

lemma *wp3_If*[*simp*]:
$wp_3\ (IF\ b\ THEN\ c_1\ ELSE\ c_2)\ Q = ((\lambda(ps,t).\ vars\ b \subseteq dom\ ps \wedge wp_3\ (if\ pbval\ b\ ps\ then\ c_1\ else\ c_2)\ Q\ (ps,t)) ** \$1)$
⟨*proof*⟩

lemma *sFTrue*: **assumes** *pbval b ps vars b* ⊆ *dom ps*
shows $wp_3\ (WHILE\ b\ DO\ c)\ Q\ (ps,\ n) = ((\lambda(ps,\ n).\ vars\ b \subseteq dom\ ps \wedge (if\ pbval\ b\ ps\ then\ wp_3\ c\ (wp_3\ (WHILE\ b\ DO\ c)\ Q)\ (ps,\ n)\ else\ Q\ (ps,\ n))) \wedge{*}\ \$\ 1)\ (ps,\ n)$
(**is** *?wp* = (*?I* ∧∗ $ 1) _)
⟨*proof*⟩

lemma *sFFalse*: **assumes** $\sim$ *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*⟩

lemma *sF'*: *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*)
⟨*proof*⟩

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*
⟨*proof*⟩

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*⟩

lemma *wp3_is_pre*: $\vdash_3 \{wp_3\ c\ Q\}\ c\ \{\ Q\}$
⟨*proof*⟩

theorem *hoare3_complete*: $\models_3 \{P\}c\{Q\} \Longrightarrow \vdash_3 \{P\}c\{Q\}$
⟨*proof*⟩

theorem *hoare3_sound_complete*: $\models_3 \{P\}c\{Q\} \longleftrightarrow \vdash_3 \{P\}c\{Q\}$
⟨*proof*⟩

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'* $\wedge$ *ps1'* ## *ps2'* $\wedge$ *n* = *e1* + *e2* + *m* $\wedge$ *Q* (*ps1'*,*e1*))

lemma *wp*$_3$ *C* (*Q*$**$(%_. *True*)) = *F C Q*
⟨*proof*⟩

definition *hoareT3_validGC* :: *assn2* ⇒ *com* ⇒ *assn2* ⇒ *bool*
(⊨$_G$ {(*1*_)}/ (_)/ { (*1*_)} *50*) **where**
⊨$_G$ { *P* } *c* { *Q* } ⟷ ⊨$_3$ { *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*
⟨*proof*⟩

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*) }
⟨*proof*⟩

lemma *prod_0*:
shows (λ(*s*::*char list* ⇒ *int option*, *c*::*nat*). *s* = *Map.empty* ∧ *c* = *0*) *h* ⟹ *h* = *0* ⟨*proof*⟩

lemma *example2*: $\vdash_3$ { (*pointsto* ''x'' n) ** (*pointsto* ''y'' n) ** $1 } ''x'' ::= *Plus* (*V* ''x'') (*N* (− 1)) { (*pointsto* ''x'' (n−1)) ** (*pointsto* ''y'' n) }
⟨*proof*⟩

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* ⇒ *com* ⇒ *assn2* ⇒ *bool*
($\models_{3'}$ {(*1*_)}/ (_)/ { (*1*_)} *50*) **where**
$\models_{3'}$ { *P* } *c* { *Q* } ⟷
(∃ *k*>*0*. (∀ *ps n*. *P* (*ps*,*n*)
⟶ (∃ *ps'* *ps''* *m e e'*. ((*c*,*ps*) ⇒$_A$ *m* ⇓ *ps'* + *ps''*)
∧ *ps'* ## *ps''* ∧ *k* ∗ *n* = *k* ∗ *e* + *e'* + *m*
∧ *Q* (*ps'*,*e*))))

9.2 Hoare Rules

inductive
hoare3a :: *assn2* ⇒ *com* ⇒ *assn2* ⇒ *bool* ($\vdash_{3a}$ ({(*1*_)}/ (_)/ { (*1*_)}) *50*)
where

Skip: $\vdash_{3a}$ {$*1*} *SKIP* { $*0*} |

Assign4: $\vdash_{3a}$ { (λ(*ps*,*t*). *x*∈*dom ps* ∧ *vars a* ⊆ *dom ps* ∧ *Q* (*ps*(*x*↦(*paval a ps*)),*t*)) ** $*1*} *x*::=*a* { *Q* } |

If: ⟦ $\vdash_{3a}$ { λ(*s*,*n*). *P* (*s*,*n*) ∧ *lmaps_to_axpr b True s* } c_1 { *Q* };
$\vdash_{3a}$ { λ(*s*,*n*). *P* (*s*,*n*) ∧ *lmaps_to_axpr b False s* } c_2 { *Q* } ⟧
⟹ $\vdash_{3a}$ { (λ(*s*,*n*). *P* (*s*,*n*) ∧ *vars b* ⊆ *dom s*) ** $*1*} *IF b THEN* c_1 *ELSE* c_2 { *Q*} |

Frame: $[\![\quad \vdash_{3a} \{\ P\ \}\ C\ \{\ Q\ \}\ ]\!]$
$\Longrightarrow \vdash_{3a} \{\ P ** F\ \}\ C\ \{\ Q ** F\ \}$ |

Seq: $[\![\vdash_{3a} \{\ P\ \}\ C_1\ \{\ Q\ \}\ ;\ \vdash_{3a} \{\ Q\ \}\ C_2\ \{\ R\ \}\ ]\!]$
$\Longrightarrow \vdash_{3a} \{\ P\ \}\ C_1\ ;;\ C_2\ \{\ R\ \}$ |

While: $[\![\vdash_{3a} \{\ (\lambda(s,n).\ P\ (s,n) \wedge lmaps\_to\_axpr\ b\ True\ s)\ \}\ C\ \{\ (\lambda(s,n).\ P\ (s,n) \wedge\ vars\ b \subseteq dom\ s)\ ** \$1\ \}\ ]\!]$
$\Longrightarrow \vdash_{3a} \{\ (\lambda(s,n).\ P\ (s,n) \wedge\ vars\ b \subseteq dom\ s) ** \$1\ \}\ WHILE\ b\ DO\ C\ \{\ \lambda(s,n).\ P\ (s,n) \wedge lmaps\_to\_axpr\ b\ False\ s\ \}$ |

conseqS: $[\![\vdash_{3a} \{P\}c\{Q\}\ ;\ \bigwedge s\ n.\ P'\ (s,n) \Longrightarrow P\ (s,n)\ ;\ \bigwedge s\ n.\ Q\ (s,n) \Longrightarrow Q'\ (s,n)\ ]\!] \Longrightarrow$
$\vdash_{3a} \{P'\}c\{\ Q'\}$

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: $[\![\vdash_{3b} \{P\}c\{Q\}\ ;\ \bigwedge s\ n.\ P'\ (s,n) \Longrightarrow P\ (s,k*n)\ ;\ \bigwedge s\ n.\ Q\ (s,n) \Longrightarrow Q'\ (s,n\ div\ k);\ k>0\ ]\!] \Longrightarrow$
$\vdash_{3b} \{P'\}c\{\ Q'\}$

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: $\vdash_{3'} \{\exists\, v\ .\ (\ ((\exists\, v'.\ pointsto\ x\ v') \ast\ast\ (\ pointsto\ x\ v \longrightarrow\ast\ Q)\ \ast\ast\ \$1)$
$and\ sep\_true \ast\ast\ (\%(ps,n).\ vars\ a \subseteq dom\ ps \wedge paval\ a\ ps = v$
$)\ )\}\ x ::= a\ \{\ Q\ \}\ \mid$

If: $[\![\vdash_{3'} \{\ \lambda(s,n).\ P\ (s,n) \wedge lmaps\_to\_axpr\ b\ True\ s\ \}\ c_1\ \{\ Q\ \};$
$\vdash_{3'} \{\ \lambda(s,n).\ P\ (s,n) \wedge lmaps\_to\_axpr\ b\ False\ s\ \}\ c_2\ \{\ Q\ \}\ ]\!]$
$\Longrightarrow \vdash_{3'} \{\ (\lambda(s,n).\ P\ (s,n) \wedge vars\ b \subseteq dom\ s) \ast\ast\ \$1\}\ IF\ b\ THEN\ c_1\ ELSE$
$c_2\ \{\ Q\}\ \mid$

Frame: $[\![\ \vdash_{3'} \{\ P\ \}\ C\ \{\ Q\ \}\ ]\!]$
$\Longrightarrow \vdash_{3'} \{\ P \ast\ast F\ \}\ C\ \{\ Q \ast\ast F\ \}\ \mid$

Seq: $[\![\vdash_{3'} \{\ P\ \}\ C_1\ \{\ Q\ \}\ ;\ \vdash_{3'} \{\ Q\ \}\ C_2\ \{\ R\ \}\ ]\!]$
$\Longrightarrow \vdash_{3'} \{\ P\ \}\ C_1\ ;;\ C_2\ \{\ R\ \}\ \mid$

While: $[\![\vdash_{3'} \{\ (\lambda(s,n).\ P\ (s,n) \wedge lmaps\_to\_axpr\ b\ True\ s)\ \}\ C\ \{\ (\lambda(s,n).$
$P\ (s,n) \wedge\ vars\ b \subseteq dom\ s)\ \ast\ast\ \$1\ \}\ ]\!]$
$\Longrightarrow \vdash_{3'} \{\ (\lambda(s,n).\ P\ (s,n) \wedge\ vars\ b \subseteq dom\ s) \ast\ast\ \$1\ \}\ WHILE\ b$
$DO\ C\ \{\ \lambda(s,n).\ P\ (s,n) \wedge lmaps\_to\_axpr\ b\ False\ s\ \}\ \mid$

conseq: $[\![\vdash_{3'} \{P\}c\{Q\}\ ;\ \bigwedge s\ n.\ P'\ (s,n) \Longrightarrow P\ (s,k*n)\ ;\ \bigwedge s\ n.\ Q\ (s,n) \Longrightarrow$
$Q'\ (s,n\ div\ k);\ k>0\ ]\!] \Longrightarrow$
$\vdash_{3'} \{P'\}c\{\ Q'\}\ \mid$

normalize: $[\![\ \vdash_{3'} \{\ P \ast\ast \$m\ \}\ C\ \{\ Q\ \ast\ast \$n\ \};\ n \leq m\ ]\!]$
$\Longrightarrow \vdash_{3'} \{\ P \ast\ast \$(m-n)\ \}\ C\ \{\ Q\ \}\ \mid$

Assign''': $\vdash_{3'} \{\ \$1\ \ast\ast\ (x \hookrightarrow ds)\ \}\ x ::= (N\ v)\ \{\ (x \hookrightarrow v)\ \}\ \mid$

Assign'''': $[\![\ symeval\ P\ a\ v; \vdash_{3'} \{P\}\ x ::= (N\ v)\ \{Q'\}\ ]\!] \Longrightarrow \vdash_{3'} \{P\}\ x ::=$
$a\ \{Q'\}\ \mid$

Assign4: $\vdash_{3'} \{\ (\lambda(ps,t).\ x \in dom\ ps \wedge vars\ a \subseteq dom\ ps \wedge Q\ (ps(x \mapsto (paval$
$a\ ps)),t)\ )\ \ast\ast\ \$1\}\ x ::= a\ \{\ Q\ \}\ \mid$

False: $\vdash_{3'} \{\ \lambda(ps,n).\ False\ \}\ c\ \{\ Q\ \}\ \mid$

pureI: $(\ P \Longrightarrow \vdash_{3'} \{\ Q\}\ c\ \{\ R\}) \Longrightarrow \vdash_{3'} \{\uparrow P \ast\ast Q\}\ c\ \{\ R\}$

definition *A4* :: *vname* ⇒ *aexp* ⇒ *assn2* ⇒ *assn2*
where *A4 x a Q* = ((λ(*ps*,*t*). *x*∈*dom ps* ∧ *vars a* ⊆ *dom ps* ∧ *Q* (*ps*(*x*↦(*paval a ps*)),*t*)) ** $*1*)
definition *A2* :: *vname* ⇒ *aexp* ⇒ *assn2* ⇒ *assn2*
where *A2 x a Q* = (∃ *v* . (((∃ *v'*. *pointsto x v'*) ** (*pointsto x v* −→* *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*)
⟨*proof*⟩

lemma *A2 x a Q* (*ps*,*n*) ⟹ *A4 x a Q* (*ps*,*n*)
⟨*proof*⟩

lemma *E_extendsR*: ⊢$_{3a}$ { *P* } *c* { *F* } ⟹ ⊢$_{3'}$ { *P* } *c* { *F* }
⟨*proof*⟩

lemma *E_extendsS*: ⊢$_{3b}$ { *P* } *c* { *F* } ⟹ ⊢$_{3'}$ { *P* } *c* { *F* }
⟨*proof*⟩

lemma *Skip'*: *P* = (*F* ** $*1*) ⟹ ⊢$_{3'}$ { *P* } *SKIP* { *F* }
⟨*proof*⟩

9.2.1 experiments with explicit and implicit GarbageCollection

lemma ((∀ *ps n*. *P* (*ps*,*n*)
⟶ (∃ *ps' ps'' m e e'*. ((*c*,*ps*) ⇒$_A$ *m* ⇓ *ps'* + *ps''*)
∧ *ps'* ## *ps''* ∧ *n* = *e* + *e'* + *m*
∧ *Q* (*ps'*,*e*))))
⟷ (∀ *ps n*. *P* (*ps*,*n*) ⟶ (∃ *ps'* *m e* . ((*c*,*ps*) ⇒$_A$ *m* ⇓ *ps'*) ∧ *n* = *e* + *m* ∧ (*Q* ** (λ_. *True*)) (*ps'*,*e*)))

⟨*proof*⟩

9.3 Soundness

theorem *hoareT_sound2_part*: **assumes** ⊢$_{3'}$ { *P* }*c*{ *Q* }
shows ⊨$_{3'}$ { *P* } *c* { *Q* } ⟨*proof*⟩

thm *hoareT_sound2_part E_extendsR*

lemma *hoareT_sound2_partR*: $\vdash_{3a} \{P\}\ c\ \{\ Q\} \Longrightarrow \models_{3'} \{P\}\ c\ \{Q\}$
⟨*proof*⟩

9.3.1 nice example

lemma *Frame_R*: **assumes** $\vdash_{3'} \{\ P\ \}\ C\ \{\ Q\ \}$ *Frame* $P'\ P\ F$
shows $\vdash_{3'} \{\ P'\ \}\ C\ \{\ Q \ast\ast\ F\ \}$
⟨*proof*⟩

lemma *strengthen_post*: **assumes** $\vdash_{3'} \{P\}c\{Q\}$ $\bigwedge s.\ Q\ s \Longrightarrow Q'\ s$
shows $\vdash_{3'} \{P\}c\{\ Q'\}$
⟨*proof*⟩

lemmas *strongAssign* = *Assign''''*[*OF _ strengthen_post*, *OF _ Frame_R*, *OF _ Assign''*]

lemma *weakenpre*: $[\![\vdash_{3'} \{P\}c\{Q\}\ ;\ \bigwedge s.\ P'\ s \Longrightarrow P\ s\ ]\!] \Longrightarrow$
$\vdash_{3'} \{P'\}c\{\ Q\}$
⟨*proof*⟩

lemma *weakenpreR*: $[\![\vdash_{3a} \{P\}c\{Q\}\ ;\ \bigwedge s.\ P'\ s \Longrightarrow P\ s\ ]\!] \Longrightarrow$
$\vdash_{3a} \{P'\}c\{\ Q\}$
⟨*proof*⟩

9.4 Completeness

definition *wp3'* :: *com* ⇒ *assn2* ⇒ *assn2* (*wp*$_{3'}$) **where**
wp$_{3'}$ *c Q* = (λ(*s*,*n*). ∃ *t m*. *n*≥*m* ∧ (*c*,*s*) ⇒$_A$ *m* ⇓ *t* ∧ *Q* (*t*,*n*−*m*))

lemma *wp3Skip*[*simp*]: *wp*$_{3'}$ *SKIP Q* = (*Q* ∗∗ $1)
⟨*proof*⟩

lemma *wp3Assign*[*simp*]: *wp*$_{3'}$ (*x* ::= *e*) *Q* = ((λ(*ps*,*t*). *vars e* ∪ {*x*} ⊆ *dom ps* ∧ *Q* (*ps*(*x* ↦ *paval e ps*),*t*)) ∗∗ $1)
⟨*proof*⟩

lemma *wpt_Seq*[*simp*]: *wp*$_{3'}$ (*c*$_1$;;*c*$_2$) *Q* = *wp*$_{3'}$ *c*$_1$ (*wp*$_{3'}$ *c*$_2$ *Q*)
⟨*proof*⟩

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*)
⟨*proof*⟩

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*⟩

lemma *sFFalse*: **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*⟩

lemma *sF'*: $wp_{3'}$ (*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*)
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *strengthen_postR*: **assumes** $\vdash_{3a}$ {*P*}*c*{*Q*} $\bigwedge$*s*. *Q s* $\Longrightarrow$ *Q' s*
shows $\vdash_{3a}$ {*P*}*c*{ *Q'*}
⟨*proof*⟩

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*⟩

lemma *wpT_is_pre*: $\vdash_{3a}$ {$wp_{3'}$ *c Q*} *c* { *Q*}

⟨*proof*⟩

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*⟩

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** $(\forall ps\ n.\ P\ (ps,n)$
$\longrightarrow (\exists ps'\ \ m\ e\ .\ ((c,ps) \Rightarrow_A m \Downarrow ps'\ )$
$\wedge\ n = e + m \wedge (Q \ast\ast sep\_true)\ (ps',e)))$
⟨*proof*⟩

lemma *hoare3o_valid_GC*: $\models_{3'} \{P\}\ c\ \{\ Q\ \} \Longrightarrow \models_{3'} \{P\}\ c\ \{\ Q \ast\ast sep\_true\}$
⟨*proof*⟩

lemma *hoare3a_sound_GC*: $\vdash_{3a} \{P\}\ c\ \{\ Q\} \Longrightarrow \models_{3'} \{P\}\ c\ \{\ Q \ast\ast sep\_true\}$ ⟨*proof*⟩

lemma *valid_wp*: $\models_{3'} \{P\}\ c\ \{\ Q\} \Longrightarrow (\exists k{>}0.\ \forall s\ n.\ P\ (s,\ n) \longrightarrow wp_{3'}\ c\ (\lambda(ps,\ n).\ (Q \ast\ast sep\_true)\ (ps,\ n\ div\ k))\ (s,\ k * n))$
⟨*proof*⟩

theorem *completeness*: $\models_{3'} \{P\}\ c\ \{\ Q\} \Longrightarrow \vdash_{3b} \{P\}\ c\ \{\ Q \ast\ast sep\_true\}$
⟨*proof*⟩

thm *E_extendsR completeness*

lemma *completenessR*: $\models_{3'} \{P\}\ c\ \{\ Q\} \Longrightarrow \vdash_{3'} \{P\}\ c\ \{\ Q \ast\ast sep\_true\}$
⟨*proof*⟩

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*

⟨*proof*⟩

lemma *vc_sound*: *vc C Q* $\Longrightarrow$ $\vdash_{3'}$ {*pre C Q*} *strip C* { *Q* }
⟨*proof*⟩

lemma *vc2valid*: *vc C Q* $\Longrightarrow$ $\forall$ *s*. *P s* $\longrightarrow$ *pre C Q s* $\Longrightarrow$ $\models_{3'}$ {*P*} *strip C* { *Q*}
⟨*proof*⟩

lemma *pre_mono*: **assumes** $\forall$ *s*. *P s* $\longrightarrow$ *Q s* **shows** $\bigwedge$*s*. *pre C P s* $\Longrightarrow$ *pre C Q s*
⟨*proof*⟩

lemma *vc_mono*: **assumes** $\forall$ *s*. *P s* $\longrightarrow$ *Q s* **shows** *vc C P* $\Longrightarrow$ *vc C Q*

⟨*proof*⟩

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'*}
⟨*proof*⟩

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*⟩

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*⟩

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*⟩

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** $em\ P' = (\%(ps,n).\ P'\ (emb\ ps\ (\%\_.\ 0)) \leq enat\ n\ )$

lemma assumes *s*: $\models_3 \{\ em\ P'\}\ c\ \{\ em\ Q'\ \}$
shows $\models_2 \{\ P'\ \}\ c\ \{\ Q'\ \}$
⟨*proof*⟩

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 $emN :: qassn \Rightarrow Nielson\_Hoare.assn2$ **where** $emN\ P = (\lambda l\ s.\ P\ s < \infty)$

lemma assumes *s*: $\models_1 \{\ emN\ P'\}\ c\ \{\ \%s.\ (THE\ e.\ enat\ e = P'\ s - Q'\ (THE\ t.\ (\exists\, n.\ (c,\ s) \Rightarrow n \Downarrow t\ ))) \Downarrow emN\ Q'\ \}$ (**is** $\models_1 \{\ ?P\ \}\ c\ \{\ ?e \Downarrow ?Q\ \}$)
shows *quantNielson*: $\models_{2'} \{\ P'\ \}\ c\ \{\ Q'\ \}$

⟨*proof*⟩

lemma assumes *s*: $\models_{2'}$ { %*s* . *emb* (∀ *l*. *P l s*) + *enat* (*e s*) } *c* { %*s*. *emb* (∀ *l*. *Q l s*) } (**is** $\models_{2'}$ { *?P* } *c* { *?Q* })
  and *sP*: ⋀*l t*. *P l t* ⟹ ∀ *l*. *P l t*
  and *sQ*: ⋀*l t*. *Q l t* ⟹ ∀ *l*. *Q l t*
 shows *NielsonQuant*: $\models_1$ { *P* } *c* { *e* ⇓ *Q* }
⟨*proof*⟩

10.2.2 Relation SepLogic to quantHoare

definition *em* :: *qassn* ⇒ (*pstate_t* ⇒ *bool*) **where**
 em P = (%(*ps*,*n*). (∀ *ex*. *P* (*Partial_Evaluation.emb ps ex*) ≤ *enat n*))

lemma assumes *s*: $\models_{3'}$ { *em P*} *c* { *em Q* }
shows $\models_{2'}$ { *P* } *c* { *Q* }
⟨*proof*⟩

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*⟩

10.3 A General Validity Predicate with Time

definition *valid* **where**
 valid P c Q n = (∀ *s*. *P s* ⟶ (∃ *s'* *m*. (*c*, *s*) ⇒ *m* ⇓ *s'* ∧ *m* ≤ *n* ∧ *Q s'*))

definition *validk* **where**
 validk P c Q n = (∃ *k*>*0*. (∀ *s*. *P s* ⟶ (∃ *s'* *m*. (*c*, *s*) ⇒ *m* ⇓ *s'* ∧ *m* ≤ *k* ∗ *n* ∧ *Q s'*)))

lemma *validk P c Q n* = (∃ *k*>*0*. *valid P c Q* (*k*∗*n*))
 ⟨*proof*⟩

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*⟩

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*⟩

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*: $a + Map.empty = a$
⟨*proof*⟩

lemma *oneway*: $\models_{3'} \{embP3\ P\ **\ \$n\}\ c\ \{embP2\ Q\} \Longrightarrow validk\ P\ c\ Q\ n$
⟨*proof*⟩

lemma *theother*: *validk P c Q n* $\Longrightarrow \models_{3'} \{embP3\ P\ **\ \$n\}\ c\ \{embP2\ Q\ \}$
⟨*proof*⟩

lemma *validk P c Q n* $\longleftrightarrow \models_{3'} \{embP3\ P\ **\ \$n\}\ c\ \{embP2\ Q\ \}$
⟨*proof*⟩

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.