

A Hoare Logic for Diverging Programs

Johannes Åman Pojhola, Magnus O. Myreen, Miki Tanaka

March 24, 2023

Abstract

This submission contains: (1) a formalisation of a small While language with support for output; (2) a standard total-correctness Hoare logic that has been proved sound and complete; and (3) a new Hoare logic for proofs about programs that diverge: this new logic has also been proved sound and complete.

Background

The theories in this submission are a part of a [similar formalisation in HOL4](#), which, in turn, is a simplified version of a program logic for nonterminating CakeML programs [1] (which was not complete).

The HOL4 version of these theories was developed by Myreen with some lemmas proved by Åman Pojhola. The Isabelle/HOL theories were developed by Tanaka with some input from Åman Pojhola.

Contents

1	Miscellaneous lemmas	2
2	The definition of the While language	3
3	A standard total correctness Hoare logic	5
4	Lemmas about the While language	6
5	Soundness of the standard Hoare logic	12
6	A Hoare logic for diverging programs	13
7	Completeness of the standard Hoare logic	14
8	Completeness of Hoare logic for diverging programs	15
9	Soundness of Hoare logic for diverging programs	17

1 Miscellaneous lemmas

theory *MiscLemmas* **imports** *HOL-Library.Sublist* **begin**

inductive *star* :: ('a ⇒ 'a ⇒ bool) ⇒ 'a ⇒ 'a ⇒ bool **for** *r* **where**
refl[*simp,intro*]: *star* *r* *x* *x*
| *step*: *r* *x* *y* ⇒ *star* *r* *y* *z* ⇒ *star* *r* *x* *z*

inductive *star-n* :: ('a ⇒ 'a ⇒ bool) ⇒ nat ⇒ 'a ⇒ 'a ⇒ bool **for** *r* **where**
refl-n: *star-n* *r* 0 *x* *x*
| *step-n*: *r* *x* *y* ⇒ *star-n* *r* *n* *y* *z* ⇒ *star-n* *r* (Suc *n*) *x* *z*

declare *star-n.intros*[*simp, intro*]

lemma *star-n-decompose*:

star-n *r* *n* *x* *y* ⇒ *star-n* *r* *n'* *x* *z* ⇒ *n'* < *n*
⇒ (∧ *x* *y* *z*. *r* *x* *y* ⇒ *r* *x* *z* ⇒ *y* = *z*)
⇒ (∧ *n* *x* *y* *z*. *star-n* *r* *n* *x* *y* ⇒ *star-n* *r* *n* *x* *z* ⇒ *y* = *z*)
⇒ *star-n* *r* (*n* - *n'*) *z* *y*
⟨*proof*⟩

lemma *star-n-add*:

star-n *r* *n* *x* *z* ⇔ (∃ *n1* *n2* *y*. *star-n* *r* *n1* *x* *y* ∧ *star-n* *r* *n2* *y* *z* ∧ *n* = *n1* + *n2*)
⟨*proof*⟩

lemma *star-star-n*:

star *r* *x* *y* ⇒ ∃ *n*. *star-n* *r* *n* *x* *y*
⟨*proof*⟩

lemma *star-n-star*:

star-n *r* *n* *x* *y* ⇒ *star* *r* *x* *y*
⟨*proof*⟩

lemma *star-eq-star-n*:

star *r* *x* *y* = (∃ *n*. *star-n* *r* *n* *x* *y*)
⟨*proof*⟩

lemma *star-trans*:

star *r* *x* *y* ⇒ *star* *r* *y* *z* ⇒ *star* *r* *x* *z*
⟨*proof*⟩

lemma *step-rev*:

star *r* *x* *y* ⇒ *r* *y* *z* ⇒ *star* *r* *x* *z*
⟨*proof*⟩

lemma *star-n-trans*:

star-n *r* *n* *x* *y* ⇒ *star-n* *r* *n'* *y* *z* ⇒ *star-n* *r* (*n* + *n'*) *x* *z*

<proof>

lemma *step-n-rev*:

$star-n\ r\ n\ x\ y \implies r\ y\ z \implies star-n\ r\ (Suc\ n)\ x\ z$

<proof>

lemma *star-n-lastE*:

$star-n\ r\ (Suc\ n)\ x\ z \implies$

$(\bigwedge n'\ x'\ y'\ z'.\ n = n' \implies x = x' \implies z = z'$

$\implies star-n\ r\ n'\ x'\ y' \implies r\ y'\ z' \implies P) \implies P$

<proof>

lemma

star-n-conjunct1: $star-n\ (\lambda x\ y.\ P\ x\ y \wedge Q\ x\ y)\ n\ s\ t \implies star-n\ P\ n\ s\ t$ **and**

star-n-conjunct2: $star-n\ (\lambda x\ y.\ P\ x\ y \wedge Q\ x\ y)\ n\ s\ t \implies star-n\ Q\ n\ s\ t$ **and**

star-n-commute: $star-n\ (\lambda x\ y.\ P\ x\ y \wedge Q\ x\ y)\ n\ s\ t \implies star-n\ (\lambda x\ y.\ Q\ x\ y \wedge P\ x\ y)\ n\ s\ t$

<proof>

lemma *forall-swap4*:

$(\forall x\ y\ z\ w.\ P\ x\ y\ z\ w) \longleftrightarrow (\forall z\ w\ y\ x.\ P\ x\ y\ z\ w)$ *<proof>*

lemma *prefix-drop-append*:

$prefix\ xs\ ys \implies xs\ @\ drop\ (length\ xs)\ ys = ys$

<proof>

lemma *min-prefix*:

$\forall i.\ prefix\ (f\ i)\ (f\ (Suc\ i)) \implies (\forall i.\ prefix\ (f\ 0)\ (f\ i))$

<proof>

definition *wf-to-wfP* **where**

$wf-to-wfP\ r \equiv \lambda x\ y.\ (x,\ y) \in r$

end

2 The definition of the While language

theory *WhileLang* **imports** *MiscLemmas Coinductive.Coinductive-List* **begin**

type-synonym *name* = *char list*

type-synonym *val* = *nat*

type-synonym *store* = *name* \Rightarrow *val*

type-synonym *exp* = *store* \Rightarrow *val*

datatype *prog* =

Skip

| *Assign name exp*

| *Print exp*
| *Seq prog prog*
| *If exp prog prog*
| *While exp prog*

type-synonym *out* = *val list*
type-synonym *state* = *store* × *out*

definition *output-of* :: *state* ⇒ *out* **where**
output-of ≡ λ(-, *out*). *out*

definition *subst* :: *name* ⇒ *exp* ⇒ *state* ⇒ *state* **where**
subst *n e* ≡ λ(*s*, *out*). (*s*(*n*:= *e s*), *out*)

definition *print* :: *exp* ⇒ *state* ⇒ *state* **where**
print e ≡ λ(*s*, *out*). (*s*, *out* @ [*e s*])

definition *guard* :: *exp* ⇒ *state* ⇒ *bool* **where**
guard x ≡ λ(*s*, -). *x s* ≠ 0

inductive *step* :: *prog* × *state* ⇒ *prog* × *state* ⇒ *bool* **where**
step-skip: *step* (*Skip*, *s*) (*Skip*, *s*)
| *step-assign*: *step* (*Assign n x*, *s*) (*Skip*, *subst n x s*)
| *step-print*: *step* (*Print x*, *s*) (*Skip*, *print x s*)
| *step-seq1*: *step* (*Seq Skip q*, *s*) (*q*, *s*)
| *step-seq2*: *step* (*p0*, *s0*) (*p1*, *s1*) ⇒ *p0* ≠ *Skip* ⇒ *step* (*Seq p0 q*, *s0*) (*Seq p1 q*, *s1*)
| *step-if*: *step* (*If x p q*, *s*) ((*if guard x s then p else q*), *s*)
| *step-while*: *step* (*While x p*, *s*) (*If x (Seq p (While x p)) Skip*, *s*)

declare *step.intros*[*simp*,*intro*]

inductive-cases *skipE*[*elim!*]: *step* (*Skip*, *s*) *ct*
inductive-cases *assignE*[*elim!*]: *step* (*Assign n x*, *s*) *ct*
inductive-cases *printE*[*elim!*]: *step* (*Print x*, *s*) *ct*
inductive-cases *seqE*[*elim!*]: *step* (*Seq c1 c2*, *s*) *ct*
inductive-cases *ifE*[*elim!*]: *step* (*If x c1 c2*, *s*) *ct*
inductive-cases *whileE*[*elim!*]: *step* (*While x p*, *s*) *ct*

lemmas *step-induct* = *step.induct*[*split-format*(*complete*)]

inductive *terminates* **where**
star step (*p*, *s*) (*Skip*, *t*) ⇒ *terminates s p t*

inductive *diverges* **where**
∀ *t*. ¬ *terminates s p t*
∧ *out* = *lSup* { *l*list-of *out* | *out*. ∃ *q t*. *star step* (*p*, *s*) (*q*, *t*, *out*) }

\implies
diverges s p out

lemma *step-exists'*:

$\exists t. \text{step } (\text{prog}, (s, \text{out})) t$
<proof>

theorem *step-exists*:

$\forall s. \exists t. \text{step } s t$
<proof>

theorem *terminates-or-diverges*:

$(\exists t. \text{terminates } s p t) \vee (\exists \text{output}. \text{diverges } s p \text{ output})$
<proof>

lemma *step-deterministic'*:

$\text{step } (\text{prog}, st, \text{out}) t1 \implies \text{step } (\text{prog}, st, \text{out}) t2 \implies t1 = t2$
<proof>

theorem *step-deterministic*:

$\text{step } s t1 \implies \text{step } s t2 \implies t1 = t2$
<proof>

lemma *star-step-refl*:

$\text{star step } (\text{Skip}, t1) (\text{Skip}, t2) \implies t1 = t2$
<proof>

theorem *terminates-deterministic*:

$\text{terminates } s p t1 \implies \text{terminates } s p t2 \implies t1 = t2$
<proof>

theorem *diverges-deterministic*:

$\text{diverges } s p t1 \implies \text{diverges } s p t2 \implies t1 = t2$
<proof>

end

3 A standard total correctness Hoare logic

theory *StdLogic* **imports** *WhileLang* **begin**

inductive *hoare* :: $(\text{state} \Rightarrow \text{bool}) \Rightarrow \text{prog} \Rightarrow (\text{state} \Rightarrow \text{bool}) \Rightarrow \text{bool}$ **where**

h-skip[*simp,intro!*]: $\text{hoare } P \text{ Skip } P$

| *h-assign*[*simp,intro!*]: $\text{hoare } (\lambda s. Q (\text{subst } n x s)) (\text{Assign } n x) Q$

| *h-print*[*simp,intro!*]: $\text{hoare } (\lambda s. Q (\text{print } x s)) (\text{Print } x) Q$

| *h-seq*[intro]: $\text{hoare } P \ p \ M \Longrightarrow \text{hoare } M \ q \ Q \Longrightarrow \text{hoare } P \ (\text{Seq } p \ q) \ Q$
| *h-if*[intro!]: $\text{hoare } (\lambda s. P \ s \wedge \text{guard } x \ s) \ p \ Q$
 $\Longrightarrow \text{hoare } (\lambda s. P \ s \wedge \sim \text{guard } x \ s) \ q \ Q \Longrightarrow \text{hoare } P \ (\text{If } x \ p \ q) \ Q$
| *h-while*: $(\bigwedge s0. \text{hoare } (\lambda s. P \ s \wedge \text{guard } x \ s \wedge s = s0) \ p \ (\lambda s. P \ s \wedge R \ s \ s0))$
 $\Longrightarrow \text{wf} P \ R \Longrightarrow \text{hoare } P \ (\text{While } x \ p) \ (\lambda s. P \ s \wedge \sim \text{guard } x \ s)$
| *h-weaken*: $(\bigwedge s. P \ s \Longrightarrow P' \ s) \Longrightarrow \text{hoare } P' \ p \ Q' \Longrightarrow (\bigwedge s. Q' \ s \Longrightarrow Q \ s) \Longrightarrow$
 $\text{hoare } P \ p \ Q$

definition *hoare-sem* :: $(\text{state} \Rightarrow \text{bool}) \Rightarrow \text{prog} \Rightarrow (\text{state} \Rightarrow \text{bool}) \Rightarrow \text{bool}$ **where**
hoare-sem $P \ p \ Q \equiv$
 $(\forall s. P \ s \longrightarrow (\exists t. \text{terminates } s \ p \ t \wedge Q \ t))$

end

4 Lemmas about the While language

theory *WhileLangLemmas* **imports** *WhileLang Coinductive.Coinductive-List-Prefix*
begin

lemma *NRC-step-deterministic*:
 $\text{star-n step } n \ x \ y \Longrightarrow \text{star-n step } n \ x \ z \Longrightarrow y = z$
 $\langle \text{proof} \rangle$

inductive *exec* **where**
exec-skip: $\text{exec } s \ \text{Skip } s$
| *exec-assign*: $\text{exec } s \ (\text{Assign } n \ x) \ (\text{subst } n \ x \ s)$
| *exec-print*: $\text{exec } s \ (\text{Print } x) \ (\text{print } x \ s)$
| *exec-seq*: $\text{exec } s0 \ p \ s1 \Longrightarrow \text{exec } s1 \ q \ s2 \Longrightarrow \text{exec } s0 \ (\text{Seq } p \ q) \ s2$
| *exec-if*: $\text{exec } s \ (\text{if guard } x \ s \ \text{then } p \ \text{else } q) \ s1 \Longrightarrow \text{exec } s \ (\text{If } x \ p \ q) \ s1$
| *exec-while1*: $\neg \text{guard } x \ s \Longrightarrow \text{exec } s \ (\text{While } x \ p) \ s$
| *exec-while2*: $\text{guard } x \ s \Longrightarrow \text{exec } s \ p \ s1 \Longrightarrow \text{exec } s1 \ (\text{While } x \ p) \ s2$
 $\Longrightarrow \text{exec } s \ (\text{While } x \ p) \ s2$

declare *exec.intros*[intro!]

lemma *NRC-step[simp]*:
 $\text{star-n step } n \ (\text{Skip}, s) \ (\text{Skip}, t) \Longrightarrow s = t$
 $\langle \text{proof} \rangle$

lemma *terminates-Skip*:
 $\text{terminates } s \ \text{Skip } t \longleftrightarrow s = t$
 $\langle \text{proof} \rangle$

lemma *NRC-assign[simp]*:
 $\text{star-n step } n \ (\text{Assign } n' \ x, s) \ (\text{Skip}, t) \Longrightarrow t = \text{subst } n' \ x \ s$
 $\langle \text{proof} \rangle$

lemma *terminates-Assign*:

$terminates\ s\ (Assign\ n\ x)\ t \longleftrightarrow t = subst\ n\ x\ s$
 $\langle proof \rangle$

lemma *NRC-print[simp]*:

$star\text{-}n\ step\ n\ (Print\ x,\ s)\ (Skip,\ t) \implies t = print\ x\ s$
 $\langle proof \rangle$

lemma *terminates-Print*:

$terminates\ s\ (Print\ x)\ t \longleftrightarrow t = print\ x\ s$
 $\langle proof \rangle$

lemma *terminates-If*:

$terminates\ s\ (If\ x\ p\ q)\ t \longleftrightarrow terminates\ s\ (if\ guard\ x\ s\ then\ p\ else\ q)\ t$
 $\langle proof \rangle$

lemma *terminates-While*:

$terminates\ s\ (While\ f\ c)\ t \longleftrightarrow terminates\ s\ (If\ f\ (Seq\ c\ (While\ f\ c))\ Skip)\ t$
 $\langle proof \rangle$

definition *real-step where*

$real\text{-}step \equiv \lambda(p,\ s)\ qt.\ p \neq Skip \wedge step\ (p,\ s)\ qt$

lemma *terminates*:

$terminates\ s\ p\ t \longleftrightarrow star\ real\text{-}step\ (p,\ s)\ (Skip,\ t)$
 $\langle proof \rangle$

lemma *NRC-real-step-Skip[simp]*:

$star\text{-}n\ real\text{-}step\ n\ (Skip,\ s)\ (Skip,\ t) \longleftrightarrow n = 0 \wedge s = t$
 $\langle proof \rangle$

lemma *NRC-real-step-not-Skip*:

$p \neq Skip \implies$
 $(star\text{-}n\ real\text{-}step\ n\ (p,\ s)\ (Skip,\ t) \longleftrightarrow$
 $(\exists k\ m.\ real\text{-}step\ (p,\ s)\ m \wedge star\text{-}n\ real\text{-}step\ k\ m\ (Skip,\ t) \wedge n = k + 1))$
 $\langle proof \rangle$

lemma *real-step-seqE*:

$real\text{-}step\ (Seq\ p\ q,\ s)\ x \implies$
 $(x = (q,\ s) \implies p = Skip \implies P) \implies$
 $(\bigwedge p'\ s'. x = (Seq\ p'\ q,\ s') \implies real\text{-}step\ (p,\ s)\ (p',\ s') \implies p \neq Skip \implies P)$
 $\implies P$
 $\langle proof \rangle$

lemma *real-steps-Seq*:

$star\text{-}n\ real\text{-}step\ n\ (Seq\ p\ q,\ s)\ (Skip,\ t) \longleftrightarrow$
 $(\exists n1\ n2\ m.$
 $star\text{-}n\ real\text{-}step\ n1\ (p,\ s)\ (Skip,\ m) \wedge$
 $star\text{-}n\ real\text{-}step\ n2\ (q,\ m)\ (Skip,\ t) \wedge n = n1 + n2 + 1)$

<proof>

lemma *terminates-Seq*:

terminates s (Seq p q) t \longleftrightarrow $(\exists m. \textit{terminates s p m} \wedge \textit{terminates m q t})$

<proof>

lemma *terminates-eq-exec*:

terminates s p t \longleftrightarrow *exec s p t*

<proof>

lemma *terminates-While-NRC*:

assumes *terminates m p t*

assumes $p = \textit{While f c}$

shows $\exists n. \textit{star-n} (\lambda s t. \textit{guard f s} \wedge \textit{terminates s c t}) n m t \wedge \neg \textit{guard f t}$

<proof>

lemma *not-diverges[simp]*:

$\sim \textit{diverges s Skip l}$

$\sim \textit{diverges s (Assign n x) l}$

$\sim \textit{diverges s (Print x) l}$

<proof>

lemma *star-n-Skip*:

star-n step n (Skip, s) (c', t) $\implies c' = \textit{Skip} \wedge t = s$

<proof>

lemma *star-n-Seq*:

star-n step n (c, s) (c', t) \implies

$\exists n. \textit{star-n step n (Seq c p, s) (Seq c' p, t)}$

<proof>

lemma *RTC-Seq*:

star step (c,s) (c',t) \implies

star step (Seq c p, s) (Seq c' p,t)

<proof>

lemma *step-output-mono*:

step s t $\implies \textit{prefix (output-of (snd s)) (output-of (snd t))}$

<proof>

lemma *NRC-step-output-mono*:

star-n step k (c,s) (c',s') $\implies \textit{prefix (output-of s) (output-of s')}$

<proof>

lemma *lprefix-chain-RTC-step'*:

Complete-Partial-Order.chain lprefix {llist-of out | out. $(\exists q t. \textit{step x (q,t,out)})$ }

<proof>

lemma *star-n-step-decompose*:

$star\text{-}n\ step\ n\ x\ y \implies star\text{-}n\ step\ n'\ x\ z \implies n' < n$
 $\implies star\text{-}n\ step\ (n - n')\ z\ y$
 <proof>

lemma *lprefix-chain-NRC-step'*:

$star\text{-}n\ step\ n\ x\ (q, t, out) \implies$
 $star\text{-}n\ step\ n'\ x\ (q', t', out') \implies$
 $lprefix\ (l\text{list-of}\ out)\ (l\text{list-of}\ out') \vee lprefix\ (l\text{list-of}\ out')\ (l\text{list-of}\ out)$
 <proof>

lemma *lprefix-chain-NRC-step*:

$Complete\text{-}Partial\text{-}Order.\text{chain}\ lprefix\ \{l\text{list-of}\ out\ |\ out.\ (\exists q\ t.\ star\text{-}n\ step\ n\ x\ (q, t, out))\}$
 <proof>

lemma *lprefix-chain-RTC-step*:

$Complete\text{-}Partial\text{-}Order.\text{chain}\ lprefix\ \{l\text{list-of}\ out\ |\ out.\ (\exists q\ t.\ star\ step\ x\ (q, t, out))\}$
 <proof>

lemma *lprefix-chain-NRC-step-ex*:

$Complete\text{-}Partial\text{-}Order.\text{chain}\ lprefix\ \{l\text{list-of}\ out\ |\ out.\ (\exists q\ t\ n.\ star\text{-}n\ step\ n\ x\ (q, t, out))\}$
 <proof>

definition *lprefix-rel where*

$lprefix\text{-}rel\ ls\ ls' \equiv \forall l \in ls.\ \exists l' \in ls'.\ lprefix\ l\ l'$

lemma *diverges-unique*:

$diverges\ s\ p\ l \implies \forall l'.\ diverges\ s\ p\ l' \longrightarrow l' = l$
 <proof>

lemma *terminates-unique*:

$terminates\ s\ p\ t \implies \forall t'.\ terminates\ s\ p\ t' \longrightarrow t' = t$
 <proof>

lemma *star-n-real-step-Seq-exact*:

$star\text{-}n\ real\text{-}step\ n\ (c, s)\ (Skip, t) \implies star\text{-}n\ step\ n\ (Seq\ c\ c', s)\ (Seq\ Skip\ c', t)$
 <proof>

lemma *star-n-real-step-Seq-exact'*:

$star\text{-}n\ real\text{-}step\ n\ (c, s)\ (Skip, t) \implies star\text{-}n\ real\text{-}step\ n\ (Seq\ c\ c', s)\ (Seq\ Skip\ c', t)$
 <proof>

lemma *div-Seq1-always-Seq*:

$\llbracket star\text{-}n\ step\ n\ (Seq\ c\ c', s)\ (q, s');\ c \neq Skip;$
 $\forall a\ b\ n.\ \neg star\text{-}n\ step\ n\ (c, s)\ (Skip, a, b) \rrbracket$

$\implies \exists u. q = \text{Seq } u \ c' \wedge u \neq \text{Skip}$
 ⟨proof⟩

lemma *div-Seq1-steps*:

$\llbracket \text{star-n step } n \ (\text{Seq } c \ c', \ s) \ (\text{Seq } u \ c', \ s'); \ c \neq \text{Skip};$
 $\forall a \ b \ n. \neg \text{star-n step } n \ (c, \ s) \ (\text{Skip}, \ a, \ b) \rrbracket$
 $\implies \text{star-n step } n \ (c, \ s) \ (u, \ s')$

⟨proof⟩

lemma *div-Seq1-lSup-eq*:

$\forall t. \neg \text{terminates } s \ c \ t$
 $\implies \text{lSup } \{\text{lList-of out } \mid \text{out}. \exists q \ t. \text{star step } (c, \ s) \ (q, \ t, \ \text{out})\} =$
 $\text{lSup } \{\text{lList-of out } \mid \text{out}. \exists q \ t. \text{star step } (\text{Seq } c \ c', \ s) \ (q, \ t, \ \text{out})\}$
 ⟨proof⟩

lemma *star-n-real-step-step*:

$\text{star-n real-step } n \ x \ y \implies \text{star-n step } n \ x \ y$
 ⟨proof⟩

lemma *div-Seq2-steps*:

$\llbracket \text{star-n real-step } n' \ (c, \ s) \ (\text{Skip}, \ s'); \ n' < n;$
 $\forall t' \ n. \neg \text{star-n real-step } n \ (c', \ s') \ (\text{Skip}, \ t');$
 $\text{star-n step } n \ (\text{Seq } c \ c', \ s) \ (q, \ t, \ \text{out}) \rrbracket$
 $\implies \exists m \ q \ t'. \text{star-n step } m \ (c', \ s') \ (q, \ t', \ \text{out})$
 ⟨proof⟩

lemma *div-Seq2-lSub-eq*:

$\llbracket \text{terminates } s \ c \ s'; \forall t. \neg \text{terminates } s' \ c' \ t \rrbracket$
 $\implies \text{lSup } \{\text{lList-of out } \mid \text{out}. \exists q \ t. \text{star step } (\text{Seq } c \ c', \ s) \ (q, \ t, \ \text{out})\} =$
 $\text{lSup } \{\text{lList-of out } \mid \text{out}. \exists q \ t. \text{star step } (c', \ s') \ (q, \ t, \ \text{out})\}$
 ⟨proof⟩

lemma *diverges-Seq*:

$\text{diverges } s \ (\text{Seq } c \ c') \ l \iff \text{diverges } s \ c \ l \vee (\exists t. \text{terminates } s \ c \ t \wedge \text{diverges } t \ c' \ l)$
 ⟨proof⟩

lemma *div-true-If-lSup-eq*:

$\llbracket \forall t. \neg \text{terminates } s \ p \ t; \text{guard } f \ s \rrbracket$
 $\implies \text{lSup } \{\text{lList-of out } \mid \text{out}. \exists q \ a \ t. \text{star step } (\text{prog.If } f \ p \ q, \ s) \ (q \ a, \ t, \ \text{out})\} =$
 $\text{lSup } \{\text{lList-of out } \mid \text{out}. \exists q \ t. \text{star step } (p, \ s) \ (q, \ t, \ \text{out})\}$
 ⟨proof⟩

lemma *div-false-If-lSup-Eq*:

$\llbracket \forall t. \neg \text{terminates } s \ q \ t; \neg \text{guard } f \ s \rrbracket$
 $\implies \text{lSup}$
 $\{\text{lList-of out } \mid \text{out}. \exists q \ a \ t. \text{star step } (\text{prog.If } f \ p \ q, \ s) \ (q \ a, \ t, \ \text{out})\} =$
 $\text{lSup } \{\text{lList-of out } \mid \text{out}. \exists q' \ t. \text{star step } (q, \ s) \ (q', \ t, \ \text{out})\}$
 ⟨proof⟩

lemma *diverges-If*:

$diverges\ s\ (If\ f\ p\ q)\ l = diverges\ s\ (if\ guard\ f\ s\ then\ p\ else\ q)\ l$
 ⟨proof⟩

lemma *While-body-add3real-step*:

$\llbracket star\text{-}n\ real\text{-}step\ n\ (c,\ s)\ (Skip,\ s');\ guard\ g\ s \rrbracket$
 $\implies star\text{-}n\ real\text{-}step\ (n + 3)\ (While\ g\ c,\ s)\ (While\ g\ c,\ s')$
 ⟨proof⟩

lemmas *While-body-add3step = While-body-add3real-step[THEN star-n-real-step-step]*

lemma *div-body-While-lSup-eq*:

$\llbracket guard\ g\ s;\ \forall t.\ \neg\ terminates\ s\ c\ t \rrbracket$
 $\implies lSup\ \{l\ list\text{-}of\ out\ | out.\ \exists q\ t.\ star\ step\ (c,\ s)\ (q,\ t,\ out)\} =$
 $lSup\ \{l\ list\text{-}of\ out\ | out.\ \exists q\ t.\ star\ step\ (While\ g\ c,\ s)\ (q,\ t,\ out)\}$
 ⟨proof⟩

lemma *inf-loop-While-steps*:

$\llbracket star\text{-}n\ real\text{-}step\ n'\ (c,\ s)\ (Skip,\ s');\ n' + 3 < n;\ guard\ g\ s;$
 $star\text{-}n\ step\ n\ (While\ g\ c,\ s)\ (q,\ t,\ out) \rrbracket$
 $\implies \exists q'\ t'\ n.\ star\text{-}n\ step\ n\ (While\ g\ c,\ s')\ (q',\ t',\ out)$
 ⟨proof⟩

lemma *inf-loop-While-lSup-eq*:

$\llbracket guard\ g\ s;\ terminates\ s\ c\ (a,\ b);\ \forall aa\ ba.\ \neg\ terminates\ (a,\ b)\ (While\ g\ c)\ (aa,\ ba) \rrbracket$
 $\implies lSup\ \{l\ list\text{-}of\ out\ | out.\ \exists q\ t.\ star\ step\ (While\ g\ c,\ a,\ b)\ (q,\ t,\ out)\} =$
 $lSup\ \{l\ list\text{-}of\ out\ | out.\ \exists q\ t.\ star\ step\ (While\ g\ c,\ s)\ (q,\ t,\ out)\}$
 ⟨proof⟩

lemma *diverges-While*:

$diverges\ s\ (While\ g\ c)\ l \iff diverges\ s\ (If\ g\ (Seq\ c\ (While\ g\ c))\ Skip)\ l$
 ⟨proof⟩

lemma *NRC-terminates*:

assumes $star\text{-}n\ (\lambda x\ y.\ terminates\ x\ c\ y)\ i\ s\ t$
shows $\forall t1.\ star\text{-}n\ (\lambda x\ y.\ terminates\ x\ c\ y)\ i\ s\ t1 \iff (t = t1)$
 ⟨proof⟩

lemma *step-output-append*:

$step\ (c,\ a,\ b)\ (c',\ a',\ b') \implies \exists new.\ b' = b\ @\ new$
 ⟨proof⟩

lemma *step-output-extend'*:

$step\ (c,\ a,\ b)\ (c',\ a',\ b\ @\ new) \implies \forall xs.\ step\ (c,\ a,\ xs)\ (c',\ a',\ xs\ @\ new)$
 ⟨proof⟩

lemma *step-output-extend*:

$step\ (c,\ a,\ b)\ (c',\ a',\ b') \implies \exists new.\ b' = b\ @\ new \wedge (\forall xs.\ step\ (c,\ a,\ xs)\ (c',\ a',\ xs))$

$xs @ new$)
(proof)

lemma *star-n-real-step-output-extend*:
 $star\text{-}n\text{-}real\text{-}step\ n\ (c,\ s)\ (Skip,\ t) \implies$
 $\exists\ new.\ snd\ t = snd\ s @ new \wedge$
 $(\forall\ xs.\ \exists\ n.\ star\text{-}n\text{-}real\text{-}step\ n\ (c,\ fst\ s,\ xs)$
 $(Skip,\ fst\ t,\ xs @ new))$
(proof)

lemma *terminates-history*:
 $terminates\ s\ c\ t \implies$
 $\exists\ new.\ snd\ t = snd\ s @ new \wedge$
 $(\forall\ xs.\ terminates\ (fst\ s,\ xs)\ c\ (fst\ t,\ xs @ new))$
(proof)

lemma *terminates-ignores-history*:
 $terminates\ (s,\ out1)\ c\ (t,\ out2) \implies$
 $terminates\ (s,\ [])\ c\ (t,\ drop\ (length\ out1)\ out2)$
(proof)

end

5 Soundness of the standard Hoare logic

theory *StdLogicSoundness* **imports** *StdLogic WhileLangLemmas* **begin**

theorem *Hoare-soundness*:
 $hoare\ P\ c\ Q \implies hoare\text{-}sem\ P\ c\ Q$
(proof)

end

theory *CoinductiveLemmas* **imports** *Coinductive.Coinductive-List* **begin**

lemma *lSup-lappend*:
 $[[\ Complete\text{-}Partial\text{-}Order.\ chain\ lprefix\ A;\ A \neq \{\}]]$
 $\implies lSup\ (lappend\ xs\ 'A) = lappend\ xs\ (lSup\ A)$
(proof)

lemma *lSup-lmap*:
 $[[\ Complete\text{-}Partial\text{-}Order.\ chain\ lprefix\ A;\ A \neq \{\}]]$
 $\implies lSup\ ((lmap\ f)\ 'A) = lmap\ f\ (lSup\ A)$
(proof)

lemma *lSup-lconcat*:
 $[[\ Complete\text{-}Partial\text{-}Order.\ chain\ lprefix\ A;\ A \neq \{\}]]$
 $\implies lSup\ (lconcat\ 'A) = lconcat\ (lSup\ A)$
(proof)

lemma *cpo-llist-of-upt*:

Complete-Partial-Order.chain lprefix $\{x. \exists i. x = \text{llist-of } [0..<i]\}$
<proof>

lemma *iterates-Suc-is-lSup-upt*:

iterates Suc $0 = \text{lSup } \{x. \exists i. x = \text{llist-of } [0..<i]\}$
<proof>

abbreviation (*input*) *flat* :: 'a list llist \Rightarrow 'a llist **where**

flat $xs \equiv \text{lconcat } (\text{lmap llist-of } xs)$

lemma *flat-inf-llist-lSup*:

flat (inf-llist f) = lSup $\{x. \exists i. x = \text{llist-of } (\text{concat } (\text{map } f [0..<i]))\}$
<proof>

lemma *upper-subset-lSup-eq*:

$\llbracket \text{Complete-Partial-Order.chain lprefix } B; A \subseteq B; \forall x \in B. \exists y \in A. \text{lprefix } x \ y \rrbracket \Longrightarrow \text{lSup } B = \text{lSup } A$
<proof>

lemma *lmap-iterates-id*:

lmap $(\lambda z. x)$ (*iterates Suc* 0) = *iterates id* x
<proof>

end

6 A Hoare logic for diverging programs

theory *DivLogic*

imports *WhileLang StdLogic CoinductiveLemmas*

begin

definition *ignores-output* :: (val \Rightarrow state \Rightarrow bool) \Rightarrow bool **where**

ignores-output $H \equiv \forall i \ s \ out1 \ out2. H \ i \ (s, out1) = H \ i \ (s, out2)$

inductive *pohjola* **where**

p-seq-d: *pohjola* $P \ p \ (D::\text{val llist} \Rightarrow \text{bool}) \Longrightarrow \text{pohjola } P \ (\text{Seq } p \ q) \ D$

| *p-seq-h*: *hoare* $P \ p \ M \Longrightarrow \text{pohjola } M \ q \ D \Longrightarrow \text{pohjola } P \ (\text{Seq } p \ q) \ D$

| *p-if*: *pohjola* $(\lambda s. P \ s \wedge \text{guard } x \ s) \ p \ D \Longrightarrow$

pohjola $(\lambda s. P \ s \wedge \sim \text{guard } x \ s) \ q \ D \Longrightarrow \text{pohjola } P \ (\text{If } x \ p \ q) \ D$

|

p-while1: $(\bigwedge s. P \ s \Longrightarrow$

$(\exists H \ ev.$

$\text{guard } x \ s \wedge H \ 0 \ s \wedge \text{ignores-output } H \wedge$

$D \ (\text{flat } (\text{LCons } (\text{output-of } s) \ (\text{inf-llist } ev))) \wedge$

$(\forall i. \text{hoare } (\lambda s. H \ i \ s \wedge \text{output-of } s = []) \ p$

$(\lambda s. H \ (i+1) \ s \wedge \text{output-of } s = ev \ i \wedge \text{guard } x \ s))) \Longrightarrow$

pohjola $P \ (\text{While } x \ p) \ D$

|

$p\text{-while2}: (\forall s. P s \longrightarrow \text{guard } x s) \Longrightarrow \text{wf} P R \Longrightarrow$
 $(\forall s0. \text{hoare } (\lambda s. P s \wedge b s \wedge s = s0) p (\lambda s. P s \wedge R s s0)) \Longrightarrow$
 $\text{pohjola } (\lambda s. P s \wedge \sim b s) p D \Longrightarrow \text{pohjola } P (\text{While } x p) D$
| $p\text{-const}: \text{pohjola } (\lambda s. \text{False}) p D$
| $p\text{-case}: \text{pohjola } (\lambda s. P s \wedge b s) p D \Longrightarrow \text{pohjola } (\lambda s. P s \wedge \sim b s) p D \Longrightarrow \text{pohjola } P p D$
| $p\text{-weaken}: (\forall s. P s \longrightarrow P' s) \Longrightarrow \text{pohjola } P' p D' \Longrightarrow (\forall s. D' s \longrightarrow D s) \Longrightarrow$
 $\text{pohjola } P p D$
print-theorems

definition *pohjola-sem* **where**

$\text{pohjola-sem } P p D \equiv$
 $\forall s. P s \longrightarrow (\exists l. \text{diverges } s p l \wedge D l)$

end

7 Completeness of the standard Hoare logic

theory *StdLogicCompleteness* **imports** *StdLogic WhileLangLemmas* **begin**

lemma *Hoare-strengthen*:

$(\bigwedge s. P s \Longrightarrow P' s) \Longrightarrow \text{hoare } P' p Q \Longrightarrow \text{hoare } P p Q$ *<proof>*

lemma *Hoare-strengthen-post*:

$(\bigwedge s. Q' s \Longrightarrow Q s) \Longrightarrow \text{hoare } P p Q' \Longrightarrow \text{hoare } P p Q$ *<proof>*

theorem *Hoare-While*:

assumes $h1: (\bigwedge s. P s \Longrightarrow R s)$

assumes $h2: (\bigwedge s. R s \wedge \neg \text{guard } x s \Longrightarrow Q s)$

assumes $h3: \bigwedge s0. \text{hoare } (\lambda s. R s \wedge \text{guard } x s \wedge s = s0) p (\lambda s. R s \wedge m s <$
 $((m s0)::\text{nat}))$

shows $\text{hoare } P (\text{While } x p) Q$

<proof>

lemma *NRC-lemma*:

$\text{star-n } (\lambda s t. \text{guard } f s \wedge \text{terminates } s c t) k0 m t0 \Longrightarrow$

$\text{star-n } (\lambda s t. \text{guard } f s \wedge \text{terminates } s c t) k1 m t1 \Longrightarrow$

$\neg \text{guard } f t0 \wedge \neg \text{guard } f t1 \Longrightarrow$

$t0 = t1 \wedge k0 = k1$

<proof>

lemma *Hoare-terminates*:

$\text{hoare } (\lambda s. \exists t. \text{terminates } s c t \wedge Q t) c Q$

<proof>

theorem *Hoare-completeness*:

$\text{hoare-sem } P c Q \Longrightarrow \text{hoare } P c Q$

<proof>

lemma *hoare-pre-False*:
 hoare (λ -. *False*) *prog Q*
 <proof>

end

8 Completeness of Hoare logic for diverging programs

theory *DivLogicCompleteness*
 imports *DivLogic StdLogicCompleteness StdLogicSoundness*
begin

declare *pohjola.intros*[*intro, simp*]
declare *pohjola.intros*(7)[*simp del*]
declare *pohjola.intros*(7)[*rule del*]
declare *pohjola.intros*(6)[*rule del*]

theorem *pohjola-strengthen*:
 $\llbracket \text{pohjola } P' p D; \forall s. P s \longrightarrow P' s \rrbracket \Longrightarrow \text{pohjola } P p D$
 <proof>

inductive *div-at-iteration* **where**
 guard f s \Longrightarrow *diverges s c l* \Longrightarrow *D l* \Longrightarrow *div-at-iteration 0 s f c D*
| *guard f s* \Longrightarrow *terminates s c t* \Longrightarrow *div-at-iteration n t f c D* \Longrightarrow
 div-at-iteration (Suc n) s f c D

print-theorems

inductive-cases
 div-at-0 [*elim!*]: *div-at-iteration 0 s f c D* **and**
 div-at-S [*elim!*]: *div-at-iteration (Suc n) s f c D*

print-theorems

theorem *div-at-iteration-11*:
 div-at-iteration i s f c D \Longrightarrow
 div-at-iteration j s f c D \Longrightarrow $i = j$
 <proof>

lemma *star-n-While-flatten*:
 star-n (λ s t. *star step* (*While x p, s*) (*While x p, t*)
 \wedge *terminates s p t* \wedge *guard x s*) *i s t*
 $\Longrightarrow \exists n'. \text{star-n step } n' (\text{While } x p, s) (\text{While } x p, t) \wedge n' \geq i$
 <proof>

lemma *diverges-init-state*:

$$\llbracket \text{terminates } s \ c \ t; \text{ diverges } t \ (\text{While } g \ c) \ l; \text{ guard } g \ s \rrbracket \Longrightarrow \text{diverges } s \ (\text{While } g \ c) \ l$$

<proof>

lemma *diverges-init-state-n*:

$$\llbracket \text{star-n } (\lambda s \ t. \text{terminates } s \ c \ t \wedge \text{guard } g \ s) \ n \ s \ t; \text{ diverges } t \ (\text{While } g \ c) \ l; \text{ guard } g \ t \rrbracket$$

$$\Longrightarrow \text{diverges } s \ (\text{While } g \ c) \ l$$

<proof>

lemma *div-at-i-unwind*:

div-at-iteration $i \ s \ g \ c \ D$

$$\longleftrightarrow (\exists t. \text{star-n } (\lambda s \ t. \text{terminates } s \ c \ t \wedge \text{guard } g \ s) \ i \ s \ t \wedge \text{guard } g \ t$$

$$\wedge (\exists l. \text{diverges } t \ c \ l \wedge D \ l))$$

<proof>

lemma *diverging-body-diverges*:

$$\llbracket \text{diverges } s \ c \ l; \text{ guard } g \ s \rrbracket \Longrightarrow \text{diverges } s \ (\text{While } g \ c) \ l$$

<proof>

lemma *non-diverging-inf-loop*:

$$\llbracket \forall i. \neg \text{div-at-iteration } i \ s \ g \ c \ D; \text{ diverges } s \ (\text{While } g \ c) \ l; D \ l \rrbracket$$

$$\Longrightarrow \forall i. \exists t. \text{star-n } (\lambda s \ t. (\exists k. \text{star-n step } k \ (\text{While } g \ c, \ s) \ (\text{While } g \ c, \ t))$$

$$\wedge \text{terminates } s \ c \ t \wedge \text{guard } g \ s) \ i \ s \ t$$

<proof>

lemma *While-lemma*:

$$\llbracket \forall i. \neg \text{div-at-iteration } i \ s \ g \ c \ D; \text{ diverges } s \ (\text{While } g \ c) \ l; D \ l \rrbracket$$

$$\Longrightarrow \exists ts. ts \ 0 = s \wedge (\forall i. \text{guard } g \ (ts \ i) \wedge \text{terminates } (ts \ i) \ c \ (ts \ (\text{Suc } i)))$$

$$\wedge (\exists k. \text{star-n step } (i+k) \ (\text{While } g \ c, \ ts \ 0) \ (\text{While } g \ c, \ ts \ i))$$

<proof>

lemma *H-for-Nil-output*:

$$H \ i \ (a, \ b) \Longrightarrow \text{ignores-output } H \Longrightarrow H \ i \ (a, \ [])$$

<proof>

lemma *output-of-simp[simp]*:

$$\text{output-of } (a, \ b) = b \ \langle \text{proof} \rangle$$

lemma *star-n-step-output-extend*:

$$\text{star-n step } n \ (c, \ s) \ (c', \ t) \Longrightarrow$$

$$\exists \text{new}. \text{snd } t = \text{snd } s \ @ \ \text{new} \wedge$$

$$(\forall xs. \text{star-n step } n \ (c, \ \text{fst } s, \ xs)$$

$$(c', \ \text{fst } t, \ xs \ @ \ \text{new}))$$

<proof>

lemma *lappend-initial-output*:

$\{l\text{list-of } out \mid out.\}$
 $\exists q t. \text{star step } (While\ x\ p,\ a,\ b)\ (q,\ t,\ out)\}$
 $= \text{lappend } (l\text{list-of } b)\ \{\{l\text{list-of } out \mid out.\}$
 $\exists q t. \text{star step } (While\ x\ p,\ a,\ [])\ (q,\ t,\ out)\}$
 $\langle \text{proof} \rangle$

lemma *ts-accum*:

$\forall i. \text{prefix } (output\text{-of } (ts\ i))\ (output\text{-of } (ts\ (Suc\ i))) \implies$
 $\text{concat } (map\ (\lambda i. \text{drop } (length\ (output\text{-of } (ts\ i)))\ (output\text{-of } (ts\ (Suc\ i))))\ [0..<i])$
 $= \text{drop } (length\ (output\text{-of } (ts\ 0)))\ (output\text{-of } (ts\ i))$
 $\langle \text{proof} \rangle$

theorem *Pohjola-diverges*:

$\text{pohjola } (\lambda s. \exists l. \text{diverges } s\ c\ l \wedge D\ l)\ c\ D$
 $\langle \text{proof} \rangle$

theorem *Pohjola-completeness*:

$\text{pohjola-sem } P\ c\ D \implies \text{pohjola } P\ c\ D$
 $\langle \text{proof} \rangle$

end

9 Soundness of Hoare logic for diverging programs

theory *DivLogicSoundness* **imports** *StdLogicSoundness DivLogicCompleteness* **begin**

lemma *p-loop-deterministic*:

$\text{star-n } (\lambda s t. \text{guard } x\ s \wedge \text{terminates } s\ p\ t)\ n\ s\ t \implies$
 $\text{star-n } (\lambda s t. \text{guard } x\ s \wedge \text{terminates } s\ p\ t)\ n\ s\ t' \implies t = t'$
 $\langle \text{proof} \rangle$

lemma *loop-accum*:

$\llbracket \forall i. \text{hoare } (\lambda s. H\ i\ s \wedge \text{output-of } s = [])\ p$
 $(\lambda s. H\ (Suc\ i)\ s \wedge \text{output-of } s = \text{ev } i \wedge \text{guard } x\ s);$
 $\text{guard } x\ (a,\ b); H\ 0\ (a,\ b); \text{ignores-output } H \rrbracket$
 $\implies \forall i. \exists s. \text{star-n } (\lambda s t. \text{guard } x\ s \wedge \text{terminates } s\ p\ t)\ i\ (a,\ b)\ s \wedge$
 $\text{guard } x\ s \wedge H\ i\ s$
 $\langle \text{proof} \rangle$

lemma *output-accum*:

$\llbracket \text{star-n } (\lambda s t. \text{guard } x\ s \wedge \text{terminates } s\ p\ t)\ i\ (a,\ b)\ s;$
 $\text{guard } x\ (a,\ b); H\ 0\ (a,\ b); \text{ignores-output } H;$
 $\forall i. \text{hoare } (\lambda s. H\ i\ s \wedge \text{output-of } s = [])\ p$
 $(\lambda s. H\ (Suc\ i)\ s \wedge \text{output-of } s = \text{ev } i \wedge \text{guard } x\ s) \rrbracket$
 $\implies \text{output-of } s = b\ @\ (\text{concat } (map\ \text{ev } [0..<i]))$
 $\langle \text{proof} \rangle$

lemma *helper-lemma*:

$$\llbracket \text{guard } x (a, b); H \ 0 (a, b); \text{ ignores-output } H;$$

$$\forall i. \text{ hoare } (\lambda s. H \ i \ s \wedge \text{ output-of } s = []) \ p$$

$$(\lambda s. H \ (\text{Suc } i) \ s \wedge \text{ output-of } s = \text{ev } i \wedge \text{ guard } x \ s) \rrbracket$$

$$\implies \forall i. \exists s. \text{ star-n } (\lambda s \ t. \text{ guard } x \ s \wedge \text{ terminates } s \ p \ t) \ i \ (a, b) \ s \wedge$$

$$\text{ guard } x \ s \wedge H \ i \ s \wedge \text{ output-of } s = b \ @ \ (\text{concat } (\text{map } \text{ev } [0..< i]))$$

<proof>

lemma *add-While-loops*:

$$\llbracket \text{star-n } (\lambda s \ t. \text{ guard } x \ s \wedge \text{ terminates } s \ p \ t) \ i \ (a, b) \ s \rrbracket$$

$$\implies \text{star-n } (\lambda s \ t. \text{ star step } (\text{While } x \ p, s) (\text{While } x \ p, t) \wedge \text{ terminates } s \ p \ t \wedge$$

$$\text{ guard } x \ s) \ i \ (a, b) \ s$$

<proof>

lemma *loop-upper-bound*:

$$\llbracket \forall i. \exists s. \text{ star-n } (\lambda s \ t. \text{ guard } x \ s \wedge \text{ terminates } s \ p \ t) \ i \ (a, b) \ s \wedge \text{ guard } x \ s \wedge H \ i \ s;$$

$$\text{star-n step } n \ (\text{While } x \ p, a, []) \ (q, t, \text{out}) \rrbracket \implies$$

$$\exists i \ t' \ n'. \text{ star-n } (\lambda s \ t. \text{ star step } (\text{While } x \ p, s) (\text{While } x \ p, t)$$

$$\wedge \text{ terminates } s \ p \ t \wedge \text{ guard } x \ s) \ i \ (a, b) \ t' \wedge$$

$$\text{star-n step } n' \ (\text{While } x \ p, a, b) \ (\text{While } x \ p, t') \wedge n \leq n'$$

<proof>

theorem *Pohjola-soundness*:

$$\text{pohjola } P \ c \ Q \implies \text{pohjola-sem } P \ c \ Q$$

<proof>

end

10 Examples

theory *Examples* **imports** *DivLogicSoundness*

begin

definition *pure-loop* :: *prog* **where**

pure-loop = *While* (λ -. 1) *Skip*

lemma *pure-loop-correct*:

$$\text{pohjola } (\lambda s. \text{ output-of } s = []) \ \text{pure-loop } (\lambda l. l = \text{LNil})$$

<proof>

definition *zero* :: *prog* **where**

zero = *While* (λ -. 1) (*Print* (λ -. 0))

definition *zero-llist* :: *nat llist* **where**

zero-llist = *iterates id 0*

lemma *zero-correct*:

pohjola ($\lambda s. \text{output-of } s = []$) *zero* ($\lambda l. l = \text{zero-list}$)
<proof>

definition *ex2* **where**

ex2 = *While* ($\lambda-. 1$) *zero*

lemma *ex2-correct*:

pohjola ($\lambda s. \text{output-of } s = []$) *ex2* ($\lambda l. l = \text{zero-list}$)
<proof>

end

References

- [1] J. A. Pohjola, H. Rostedt, and M. O. Myreen. Characteristic formulae for liveness properties of non-terminating CakeML programs. In *Interactive Theorem Proving (ITP)*. LIPIcs, 2019.