A Hoare Logic for Diverging Programs

Johannes Åman Pojhola, Magnus O. Myreen, Miki Tanaka

March 17, 2025

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 port 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

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

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

inductive *star-n* :: $('a \Rightarrow 'a \Rightarrow bool) \Rightarrow nat \Rightarrow 'a \Rightarrow 'a \Rightarrow bool$ **for** r **where**
  refl-n: $star\text{-}n\ r\ 0\ x\ x$
| *step-n*: $r\ x\ y \Longrightarrow star\text{-}n\ r\ n\ y\ z \Longrightarrow star\text{-}n\ r\ (Suc\ n)\ x\ z$

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

lemma *star-n-decompose*:
  $star\text{-}n\ r\ n\ x\ y \Longrightarrow star\text{-}n\ r\ n'\ x\ z \Longrightarrow n' < n$
    $\Longrightarrow (\bigwedge x\ y\ z.\ r\ x\ y \Longrightarrow r\ x\ z \Longrightarrow y = z)$
    $\Longrightarrow (\bigwedge n\ x\ y\ z.\ star\text{-}n\ r\ n\ x\ y \Longrightarrow star\text{-}n\ r\ n\ x\ z \Longrightarrow y = z)$
    $\Longrightarrow star\text{-}n\ r\ (n - n')\ z\ y$
  ⟨*proof*⟩

lemma *star-n-add*:
  $star\text{-}n\ r\ n\ x\ z \longleftrightarrow (\exists\, n1\ n2\ y.\ star\text{-}n\ r\ n1\ x\ y \wedge star\text{-}n\ r\ n2\ y\ z \wedge n = n1 + n2)$
⟨*proof*⟩

lemma *star-star-n*:
  $star\ r\ x\ y \Longrightarrow \exists\, n.\ star\text{-}n\ r\ n\ x\ y$
  ⟨*proof*⟩

lemma *star-n-star*:
  $star\text{-}n\ r\ n\ x\ y \Longrightarrow star\ r\ x\ y$
  ⟨*proof*⟩

lemma *star-eq-star-n*:
  $star\ r\ x\ y = (\exists\, n.\ star\text{-}n\ r\ n\ x\ y)$
  ⟨*proof*⟩

lemma *star-trans*:
  $star\ r\ x\ y \Longrightarrow star\ r\ y\ z \Longrightarrow star\ r\ x\ z$
  ⟨*proof*⟩

lemma *step-rev*:
  $star\ r\ x\ y \Longrightarrow r\ y\ z \Longrightarrow star\ r\ x\ z$
  ⟨*proof*⟩

lemma *star-n-trans*:
  $star\text{-}n\ r\ n\ x\ y \Longrightarrow star\text{-}n\ r\ n'\ y\ z \Longrightarrow star\text{-}n\ r\ (n + n')\ x\ z$

⟨*proof*⟩

lemma *step-n-rev*:
star-n r n x y ⟹ *r y z* ⟹ *star-n r* (*Suc n*) *x z*
⟨*proof*⟩

lemma *star-n-lastE*:
star-n r (*Suc n*) *x z* ⟹
(⋀*n' x' y' z'*. *n* = *n'* ⟹ *x* = *x'* ⟹ *z* = *z'*
⟹ *star-n r n' x' y'* ⟹ *r y' z'* ⟹ *P*) ⟹ *P*
⟨*proof*⟩

lemma
star-n-conjunct1: *star-n* (λ*x y*. *P x y* ∧ *Q x y*) *n s t* ⟹ *star-n P n s t* **and**
star-n-conjunct2: *star-n* (λ*x y*. *P x y* ∧ *Q x y*) *n s t* ⟹ *star-n Q n s t* **and**
star-n-commute: *star-n* (λ*x y*. *P x y* ∧ *Q x y*) *n s t* ⟹ *star-n* (λ*x y*. *Q x y* ∧ *P x y*) *n s t*
⟨*proof*⟩

lemma *forall-swap4*:
(∀ *x y z w*. *P x y z w*) ⟷ (∀ *z w y x*. *P x y z w*) ⟨*proof*⟩

lemma *prefix-drop-append*:
prefix xs ys ⟹ *xs* @ *drop* (*length xs*) *ys* = *ys*
⟨*proof*⟩

lemma *min-prefix*:
∀ *i*. *prefix* (*f i*) (*f* (*Suc i*)) ⟹ (∀ *i*. *prefix* (*f 0*) (*f i*))
⟨*proof*⟩

definition *wf-to-wfP* **where**
wf-to-wfP r ≡ λ*x y*. (*x*, *y*) ∈ *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* ⇒ *val*
type-synonym *exp* = *store* ⇒ *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* { *llist-of out* | *out*. ∃ *q t*. *star step* (*p*, *s*) (*q*, *t*, *out*) }

$\Longrightarrow$
diverges s p out

lemma *step-exists'*:
$\exists t.$ *step* (*prog*, (*s*, *out*)) *t*
⟨*proof*⟩

theorem *step-exists*:
$\forall s.\ \exists t.$ *step s t*
⟨*proof*⟩

theorem *terminates-or-diverges*:
$(\exists t.$ *terminates s p t*$) \vee (\exists$ *output. diverges s p output*$)$
⟨*proof*⟩

lemma *step-deterministic'*:
step (*prog*, *st*, *out*) *t1* $\Longrightarrow$ *step* (*prog*, *st*, *out*) *t2* $\Longrightarrow$ *t1* = *t2*
⟨*proof*⟩

theorem *step-deterministic*:
step s t1 $\Longrightarrow$ *step s t2* $\Longrightarrow$ *t1* = *t2*
⟨*proof*⟩

lemma *star-step-refl*:
star step (*Skip*, *t1*) (*Skip*, *t2*) $\Longrightarrow$ *t1* = *t2*
⟨*proof*⟩

theorem *terminates-deterministic*:
terminates s p t1 $\Longrightarrow$ *terminates s p t2* $\Longrightarrow$ *t1* = *t2*
⟨*proof*⟩

theorem *diverges-deterministic*:
diverges s p t1 $\Longrightarrow$ *diverges s p t2* $\Longrightarrow$ *t1* = *t2*
⟨*proof*⟩

end

3 A standard total correctness Hoare logic

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

inductive *hoare* :: (*state* ⇒ *bool*) ⇒ *prog* ⇒ (*state* ⇒ *bool*) ⇒ *bool* **where**
h-skip[*simp*,*intro*!]: *hoare P Skip P*
| *h-assign*[*simp*,*intro*!]: *hoare* (λ*s. Q* (*subst n x s*)) (*Assign n x*) *Q*
| *h-print*[*simp*,*intro*!]: *hoare* (λ*s. Q* (*print x s*)) (*Print x*) *Q*

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

definition *hoare-sem* :: (*state* $\Rightarrow$ *bool*) $\Rightarrow$ *prog* $\Rightarrow$ (*state* $\Rightarrow$ *bool*) $\Rightarrow$ *bool* **where**
hoare-sem P p Q $\equiv$
($\forall s.$ *P s* $\longrightarrow$ ($\exists t.$ *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*:
star-n step n x y $\Longrightarrow$ *star-n step n x z* $\Longrightarrow$ *y* = *z*
⟨*proof*⟩

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

declare *exec.intros*[*intro*!]

lemma *NRC-step*[*simp*]:
star-n step n (*Skip*,*s*) (*Skip*,*t*) $\Longrightarrow$ *s* = *t*
⟨*proof*⟩

lemma *terminates-Skip*:
terminates s Skip t $\longleftrightarrow$ *s* = *t*
⟨*proof*⟩

lemma *NRC-assign*[*simp*]:
star-n step n (*Assign n′ x*,*s*) (*Skip*, *t*) $\Longrightarrow$ *t* = *subst n′ x s*
⟨*proof*⟩

lemma *terminates-Assign*:
terminates s (Assign n x) t $\longleftrightarrow$ *t = subst n x s*
⟨*proof*⟩

lemma *NRC-print*[*simp*]:
star-n step n (Print x,s) (Skip, t) $\Longrightarrow$ *t = print x s*
⟨*proof*⟩

lemma *terminates-Print*:
terminates s (Print x) t $\longleftrightarrow$ *t = print x s*
⟨*proof*⟩

lemma *terminates-If*:
terminates s (If x p q) t $\longleftrightarrow$ *terminates s (if guard x s then p else q) t*
⟨*proof*⟩

lemma *terminates-While*:
terminates s (While f c) t $\longleftrightarrow$ *terminates s (If f (Seq c (While f c)) Skip) t*
⟨*proof*⟩

definition *real-step* **where**
real-step $\equiv \lambda(p,\ s)\ qt.\ p \neq Skip \wedge step\ (p,s)\ qt$

lemma *terminates*:
terminates s p t $\longleftrightarrow$ *star real-step (p,s) (Skip,t)*
⟨*proof*⟩

lemma *NRC-real-step-Skip*[*simp*]:
star-n real-step n (Skip,s) (Skip,t) $\longleftrightarrow$ *n = 0* $\wedge$ *s = t*
⟨*proof*⟩

lemma *NRC-real-step-not-Skip*:
p $\neq$ *Skip* $\Longrightarrow$
(*star-n real-step n (p,s) (Skip,t)* $\longleftrightarrow$
(∃ *k m. real-step (p,s) m* $\wedge$ *star-n real-step k m (Skip,t)* $\wedge$ *n = k + 1*))
⟨*proof*⟩

lemma *real-step-seqE*:
real-step (Seq p q, s) x $\Longrightarrow$
(*x = (q, s)* $\Longrightarrow$ *p = Skip* $\Longrightarrow$ *P*) $\Longrightarrow$
(⋀*p′ s′. x = (Seq p′ q, s′)* $\Longrightarrow$ *real-step (p, s) (p′, s′)* $\Longrightarrow$ *p* $\neq$ *Skip* $\Longrightarrow$ *P*)
$\Longrightarrow$ *P*
⟨*proof*⟩

lemma *real-steps-Seq*:
star-n real-step n (Seq p q,s) (Skip,t) $\longleftrightarrow$
(∃ *n1 n2 m.*
star-n real-step n1 (p,s) (Skip,m) $\wedge$
star-n real-step n2 (q,m) (Skip,t) $\wedge$ *n = n1 + n2 + 1*)

⟨*proof*⟩

lemma *terminates-Seq*:
terminates s (*Seq p q*) *t* ⟷ (∃ *m*. *terminates s p m* ∧ *terminates m q t*)
⟨*proof*⟩

lemma *terminates-eq-exec*:
terminates s p t ⟷ *exec s p t*
⟨*proof*⟩

lemma *terminates-While-NRC*:
assumes *terminates m p t*
assumes *p* = *While f c*
shows ∃ *n*. *star-n* (λ*s t*. *guard f s* ∧ *terminates s c t*) *n m t* ∧ ¬*guard f t*
⟨*proof*⟩

lemma *not-diverges*[*simp*]:
~*diverges s Skip l*
~*diverges s* (*Assign n x*) *l*
~*diverges s* (*Print x*) *l*
⟨*proof*⟩

lemma *star-n-Skip*:
star-n step n (*Skip*, *s*) (*c*′, *t*) ⟹ *c*′ = *Skip* ∧ *t* = *s*
⟨*proof*⟩

lemma *star-n-Seq*:
star-n step n (*c*, *s*) (*c*′, *t*) ⟹
∃ *n*. *star-n step n* (*Seq c p*, *s*) (*Seq c*′ *p*, *t*)
⟨*proof*⟩

lemma *RTC-Seq*:
star step (*c*,*s*) (*c*′,*t*) ⟹
star step (*Seq c p*, *s*) (*Seq c*′ *p*,*t*)
⟨*proof*⟩

lemma *step-output-mono*:
step s t ⟹ *prefix* (*output-of* (*snd s*)) (*output-of* (*snd t*))
⟨*proof*⟩

lemma *NRC-step-output-mono*:
star-n step k (*c*,*s*) (*c*′,*s*′) ⟹ *prefix* (*output-of s*) (*output-of s*′)
⟨*proof*⟩

lemma *lprefix-chain-RTC-step*′:
Complete-Partial-Order.chain lprefix {*llist-of out* | *out*. (∃ *q t*. *step x* (*q*,*t*,*out*))}
⟨*proof*⟩

lemma *star-n-step-decompose*:

star-n step n x y $\Longrightarrow$ *star-n step n' x z* $\Longrightarrow$ $n' < n$
$\Longrightarrow$ *star-n step* $(n - n')$ *z y*
⟨*proof*⟩

lemma *lprefix-chain-NRC-step'*:
star-n step n x (*q*, *t*, *out*) $\Longrightarrow$
star-n step n' x (*q'*, *t'*, *out'*) $\Longrightarrow$
lprefix (*llist-of out*) (*llist-of out'*) $\vee$ *lprefix* (*llist-of out'*) (*llist-of out*)
⟨*proof*⟩

lemma *lprefix-chain-NRC-step*:
Complete-Partial-Order.chain lprefix {*llist-of out* | *out*. ($\exists$ *q t*. *star-n step n x* (*q*,*t*,*out*))}
⟨*proof*⟩

lemma *lprefix-chain-RTC-step*:
Complete-Partial-Order.chain lprefix {*llist-of out* | *out*. ($\exists$ *q t*. *star step x* (*q*,*t*,*out*))}
⟨*proof*⟩

lemma *lprefix-chain-NRC-step-ex*:
Complete-Partial-Order.chain lprefix {*llist-of out* | *out*. ($\exists$ *q t n*. *star-n step n x* (*q*,*t*,*out*))}
⟨*proof*⟩

definition *lprefix-rel* **where**
lprefix-rel ls ls' $\equiv \forall\, l \in ls.\ \exists\, l' \in ls'.$ *lprefix l l'*

lemma *diverges-unique*:
diverges s p l $\Longrightarrow \forall\, l'.$ *diverges s p l'* $\longrightarrow l' = l$
⟨*proof*⟩

lemma *terminates-unique*:
terminates s p t $\Longrightarrow \forall\, t'.$ *terminates s p t'* $\longrightarrow t' = t$
⟨*proof*⟩

lemma *star-n-real-step-Seq-exact*:
star-n real-step n (*c*, *s*) (*Skip*, *t*) $\Longrightarrow$ *star-n step n* (*Seq c c'*, *s*) (*Seq Skip c'*, *t*)
⟨*proof*⟩

lemma *star-n-real-step-Seq-exact'*:
star-n real-step n (*c*, *s*) (*Skip*, *t*) $\Longrightarrow$ *star-n real-step n* (*Seq c c'*, *s*) (*Seq Skip c'*, *t*)
⟨*proof*⟩

lemma *div-Seq1-always-Seq*:
⟦ *star-n step n* (*Seq c c'*, *s*) (*q*, *s'*); $c \neq$ *Skip*;
$\forall\, a\ b\ n. \neg$ *star-n step n* (*c*, *s*) (*Skip*, *a*, *b*)⟧

$\Longrightarrow \exists\, u.\ q = Seq\ u\ c' \wedge u \neq Skip$
⟨*proof*⟩

lemma *div-Seq1-steps*:
⟦*star-n step n* (*Seq c c'*, *s*) (*Seq u c'*, *s'*); *c* ≠ *Skip*;
∀ *a b n*. ¬ *star-n step n* (*c*, *s*) (*Skip*, *a*, *b*)⟧
$\Longrightarrow$ *star-n step n* (*c*, *s*) (*u*, *s'*)
⟨*proof*⟩

lemma *div-Seq1-lSup-eq*:
∀ *t*. ¬ *terminates s c t*
$\Longrightarrow$ *lSup* {*llist-of out* |*out*. ∃ *q t*. *star step* (*c*, *s*) (*q*, *t*, *out*)} =
lSup {*llist-of out* |*out*. ∃ *q t*. *star step* (*Seq c c'*, *s*) (*q*, *t*, *out*)}
⟨*proof*⟩

lemma *star-n-real-step-step*:
star-n real-step n x y $\Longrightarrow$ *star-n step n x y*
⟨*proof*⟩

lemma *div-Seq2-steps*:
⟦ *star-n real-step n'* (*c*, *s*) (*Skip*, *s'*); *n'* < *n*;
∀ *t' n*. ¬ *star-n real-step n* (*c'*, *s'*) (*Skip*, *t'*);
star-n step n (*Seq c c'*, *s*) (*q*, *t*, *out*)⟧
$\Longrightarrow \exists\, m\ q\ t'.$ *star-n step m* (*c'*, *s'*) (*q*, *t'*, *out*)
⟨*proof*⟩

lemma *div-Seq2-lSub-eq*:
⟦*terminates s c s'*; ∀ *t*. ¬ *terminates s' c' t*⟧
$\Longrightarrow$ *lSup* {*llist-of out* |*out*. ∃ *q t*. *star step* (*Seq c c'*, *s*) (*q*, *t*, *out*)} =
lSup {*llist-of out* |*out*. ∃ *q t*. *star step* (*c'*, *s'*) (*q*, *t*, *out*)}
⟨*proof*⟩

lemma *diverges-Seq*:
diverges s (*Seq c c'*) *l* ⟷ *diverges s c l* ∨ (∃ *t*. *terminates s c t* ∧ *diverges t c' l*)
⟨*proof*⟩

lemma *div-true-If-lSup-eq*:
⟦∀ *t*. ¬ *terminates s p t*; *guard f s*⟧
$\Longrightarrow$ *lSup* {*llist-of out* |*out*. ∃ *qa t*. *star step* (*prog.If f p q*, *s*) (*qa*, *t*, *out*)} =
lSup {*llist-of out* |*out*. ∃ *q t*. *star step* (*p*, *s*) (*q*, *t*, *out*)}
⟨*proof*⟩

lemma *div-false-If-lSup-Eq*:
⟦∀ *t*. ¬ *terminates s q t*; ¬ *guard f s*⟧
$\Longrightarrow$ *lSup*
{*llist-of out* |*out*. ∃ *qa t*. *star step* (*prog.If f p q*, *s*) (*qa*, *t*, *out*)} =
lSup {*llist-of out* |*out*. ∃ *q' t*. *star step* (*q*, *s*) (*q'*, *t*, *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*:
⟦*star-n real-step n (c, s) (Skip, s′); guard g s*⟧
⟹ *star-n real-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*:
⟦*guard g s*; ∀ *t*. ¬ *terminates s c t*⟧
⟹ *lSup* {*llist-of out* |*out*. ∃ *q t*. *star step* (*c*, *s*) (*q*, *t*, *out*)} =
lSup {*llist-of out* |*out*. ∃ *q t*. *star step* (*While g c*, *s*) (*q*, *t*, *out*)}
⟨*proof*⟩

lemma *inf-loop-While-steps*:
⟦ *star-n real-step n′* (*c*, *s*) (*Skip*, *s′*); *n′* + 3 < *n*; *guard g s*;
star-n step n (*While g c*, *s*) (*q*, *t*, *out*)⟧
⟹ ∃ *q′ t′ n*. *star-n step n* (*While g c*, *s′*) (*q′*, *t′*, *out*)
⟨*proof*⟩

lemma *inf-loop-While-lSup-eq*:
⟦*guard g s*; *terminates s c* (*a*, *b*); ∀ *aa ba*. ¬ *terminates* (*a*, *b*) (*While g c*) (*aa*, *ba*)⟧
⟹ *lSup* {*llist-of out* |*out*. ∃ *q t*. *star step* (*While g c*, *a*, *b*) (*q*, *t*, *out*)} =
lSup {*llist-of out* |*out*. ∃ *q t*. *star step* (*While g c*, *s*) (*q*, *t*, *out*)}
⟨*proof*⟩

lemma *diverges-While*:
diverges s (*While g c*) *l* ⟷ *diverges s* (*If g* (*Seq c* (*While g c*)) *Skip*) *l*
⟨*proof*⟩

lemma *NRC-terminates*:
assumes *star-n* (λ*x y*. *terminates x c y*) *i s t*
shows ∀ *t1*. *star-n* (λ*x y*. *terminates x c y*) *i s t1* ⟷ (*t* = *t1*)
⟨*proof*⟩

lemma *step-output-append*:
step (*c*, *a*, *b*) (*c′*, *a′*, *b′*) ⟹ ∃ *new*. *b′* = *b* @ *new*
⟨*proof*⟩

lemma *step-output-extend′*:
step (*c*, *a*, *b*) (*c′*, *a′*, *b* @ *new*) ⟹ ∀ *xs*. *step* (*c*, *a*, *xs*) (*c′*, *a′*, *xs* @ *new*)
⟨*proof*⟩

lemma *step-output-extend*:
step (*c*, *a*, *b*) (*c′*, *a′*, *b′*) ⟹ ∃ *new*. *b′* = *b* @ *new* ∧ (∀ *xs*. *step* (*c*, *a*, *xs*) (*c′*, *a′*,

xs @ *new*))
⟨*proof*⟩

lemma *star-n-real-step-output-extend*:
star-n real-step n (*c*, *s*) (*Skip*, *t*) ⟹
∃ *new*. *snd t* = *snd s* @ *new* ∧
(∀ *xs*. ∃ *n*. *star-n real-step n* (*c*, *fst s*, *xs*)
(*Skip*, *fst t*, *xs* @ *new*))
⟨*proof*⟩

lemma *terminates-history*:
terminates s c t ⟹
∃ *new*. *snd t* = *snd s* @ *new* ∧
(∀ *xs*. *terminates* (*fst s*,*xs*) *c* (*fst t*,*xs* @ *new*))
⟨*proof*⟩

lemma *terminates-ignores-history*:
terminates (*s*, *out1*) *c* (*t*, *out2*) ⟹
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 ⟹ *hoare-sem P c Q*
⟨*proof*⟩

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

lemma *lSup-lappend*:
⟦*Complete-Partial-Order.chain lprefix A*; *A* ≠ {}⟧
⟹ *lSup* (*lappend xs* ‘ *A*) = *lappend xs* (*lSup A*)
⟨*proof*⟩

lemma *lSup-lmap*:
⟦*Complete-Partial-Order.chain lprefix A*; *A* ≠ {}⟧
⟹ *lSup* ((*lmap f*) ‘ *A*) = *lmap f* (*lSup A*)
⟨*proof*⟩

lemma *lSup-lconcat*:
⟦*Complete-Partial-Order.chain lprefix A*; *A* ≠ {}⟧
⟹ *lSup* (*lconcat* ‘ *A*) = *lconcat* (*lSup A*)
⟨*proof*⟩

lemma *cpo-llist-of-upt*:
Complete-Partial-Order.chain lprefix {*x*. ∃ *i*. *x* = *llist-of* [*0*..<*i*]}
⟨*proof*⟩

lemma *iterates-Suc-is-lSup-upt*:
iterates Suc 0 = *lSup* {*x*. ∃ *i*. *x* = *llist-of* [*0*..<*i*]}
⟨*proof*⟩

abbreviation (*input*) *flat* :: *'a list llist* ⇒ *'a llist* **where**
flat xs ≡ *lconcat* (*lmap llist-of xs*)

lemma *flat-inf-llist-lSup*:
flat (*inf-llist f*) = *lSup* {*x*. ∃ *i*. *x* = *llist-of* (*concat* (*map f* [*0*..<*i*]))}
⟨*proof*⟩

lemma *upper-subset-lSup-eq*:
⟦*Complete-Partial-Order.chain lprefix B*; *A* ⊆ *B*;
∀ *x*∈*B*. ∃ *y*∈*A*. *lprefix x y*⟧ ⟹ *lSup B* = *lSup A*
⟨*proof*⟩

lemma *lmap-iterates-id*:
lmap (λ*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* ⇒ *state* ⇒ *bool*) ⇒ *bool* **where**
ignores-output H ≡ ∀ *i s out1 out2*. *H i* (*s*,*out1*) = *H i* (*s*,*out2*)

inductive *pohjola* **where**
p-seq-d: *pohjola P p* (*D*::*val llist* ⇒ *bool*) ⟹ *pohjola P* (*Seq p q*) *D*
| *p-seq-h*: *hoare P p M* ⟹ *pohjola M q D* ⟹ *pohjola P* (*Seq p q*) *D*
| *p-if*: *pohjola* (λ*s*. *P s* ∧ *guard x s*) *p D* ⟹
pohjola (λ*s*. *P s* ∧ ~*guard x s*) *q D* ⟹ *pohjola P* (*If x p q*) *D*
|
p-while1: (⋀*s*. *P s* ⟹
(∃ *H ev*.
guard x s ∧ *H 0 s* ∧ *ignores-output H* ∧
D (*flat* (*LCons* (*output-of s*) (*inf-llist ev*))) ∧
(∀ *i*. *hoare* (λ*s*. *H i s* ∧ *output-of s* = []) *p*
(λ*s*. *H* (*i*+*1*) *s* ∧ *output-of s* = *ev i* ∧ *guard x s*)))) ⟹
pohjola P (*While x p*) *D*
|

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

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

end

7 Completeness of the standard Hoare logic

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

lemma *Hoare-strengthen*:
$(\bigwedge s.\ P\ s \Longrightarrow P'\ s) \Longrightarrow \mathit{hoare}\ P'\ p\ Q \Longrightarrow \mathit{hoare}\ P\ p\ Q$ ⟨*proof*⟩

lemma *Hoare-strengthen-post*:
$(\bigwedge s.\ Q'\ s \Longrightarrow Q\ s) \Longrightarrow \mathit{hoare}\ P\ p\ Q' \Longrightarrow \mathit{hoare}\ P\ p\ Q$ ⟨*proof*⟩

theorem *Hoare-While*:
assumes *h1*: $(\bigwedge s.\ P\ s \Longrightarrow R\ s)$
assumes *h2*: $(\bigwedge s.\ R\ s \wedge \neg \mathit{guard}\ x\ s \Longrightarrow Q\ s)$
assumes *h3*: $\bigwedge s0.\ \mathit{hoare}\ (\lambda s.\ R\ s \wedge \mathit{guard}\ x\ s \wedge s = s0)\ p\ (\lambda s.\ R\ s \wedge m\ s < ((m\ s0)::\mathit{nat}))$
shows $\mathit{hoare}\ P\ (\mathit{While}\ x\ p)\ Q$
⟨*proof*⟩

lemma *NRC-lemma*:
$\mathit{star\text{-}n}\ (\lambda s\ t.\ \mathit{guard}\ f\ s \wedge \mathit{terminates}\ s\ c\ t)\ k0\ m\ t0 \Longrightarrow$
$\mathit{star\text{-}n}\ (\lambda s\ t.\ \mathit{guard}\ f\ s \wedge \mathit{terminates}\ s\ c\ t)\ k1\ m\ t1 \Longrightarrow$
$\neg \mathit{guard}\ f\ t0 \wedge \neg \mathit{guard}\ f\ t1 \Longrightarrow$
$t0 = t1 \wedge k0 = k1$
⟨*proof*⟩

lemma *Hoare-terminates*:
$\mathit{hoare}\ (\lambda s.\ \exists\, t.\ \mathit{terminates}\ s\ c\ t \wedge Q\ t)\ c\ Q$
⟨*proof*⟩

theorem *Hoare-completeness*:
$\mathit{hoare\text{-}sem}\ P\ c\ Q \Longrightarrow \mathit{hoare}\ P\ c\ Q$

⟨*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*:
⟦*pohjola P' p D*; ∀ *s*. *P s* ⟶ *P' s*⟧ ⟹ *pohjola P p D*
⟨*proof*⟩

inductive *div-at-iteration* **where**
guard f s ⟹ *diverges s c l* ⟹ *D l* ⟹ *div-at-iteration 0 s f c D*
| *guard f s* ⟹ *terminates s c t* ⟹ *div-at-iteration n t f c D* ⟹
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 ⟹
div-at-iteration j s f c D ⟹ *i* = *j*
⟨*proof*⟩

lemma *star-n-While-flatten*:
star-n (λ*s t*. *star step* (*While x p*, *s*) (*While x p*, *t*)
∧ *terminates s p t* ∧ *guard x s*) *i s t*
⟹ ∃ *n'*. *star-n step n'* (*While x p*, *s*) (*While x p*, *t*) ∧ *n'* ≥ *i*
⟨*proof*⟩

lemma *diverges-init-state*:
⟦*terminates s c t*; *diverges t* (*While g c*) *l*; *guard g s*⟧ ⟹ *diverges s* (*While g c*) *l*
⟨*proof*⟩

lemma *diverges-init-state-n*:
⟦*star-n* (λ*s t*. *terminates s c t* ∧ *guard g s*) *n s t*; *diverges t* (*While g c*) *l*; *guard g t*⟧
⟹ *diverges s* (*While g c*) *l*
⟨*proof*⟩

lemma *div-at-i-unwind*:
div-at-iteration i s g c D
⟷ (∃ *t*. *star-n* (λ*s t*. *terminates s c t* ∧ *guard g s*) *i s t* ∧ *guard g t*
∧ (∃ *l*. *diverges t c l* ∧ *D l*))
⟨*proof*⟩

lemma *diverging-body-diverges*:
⟦*diverges s c l*; *guard g s*⟧ ⟹ *diverges s* (*While g c*) *l*
⟨*proof*⟩

lemma *non-diverging-inf-loop*:
⟦∀ *i*. ¬ *div-at-iteration i s g c D*; *diverges s* (*While g c*) *l*; *D l*⟧
⟹ ∀ *i*. ∃ *t*. *star-n* (λ*s t*. (∃ *k*. *star-n step k* (*While g c*, *s*) (*While g c*, *t*))
∧ *terminates s c t* ∧ *guard g s*) *i s t*
⟨*proof*⟩

lemma *While-lemma*:
⟦∀ *i*. ¬ *div-at-iteration i s g c D*; *diverges s* (*While g c*) *l*; *D l*⟧
⟹ ∃ *ts*. *ts 0* = *s* ∧ (∀ *i*. *guard g* (*ts i*) ∧ *terminates* (*ts i*) *c* (*ts* (*Suc i*))
∧ (∃ *k*. *star-n step* (*i*+*k*) (*While g c*, *ts 0*) (*While g c*, *ts i*)))
⟨*proof*⟩

lemma *H-for-Nil-output*:
H i (*a*, *b*) ⟹ *ignores-output H* ⟹ *H i* (*a*, [])
⟨*proof*⟩

lemma *output-of-simp*[*simp*]:
output-of (*a*, *b*) = *b* ⟨*proof*⟩

lemma *star-n-step-output-extend*:
star-n step n (*c*, *s*) (*c*′, *t*) ⟹
∃ *new*. *snd t* = *snd s* @ *new* ∧
(∀ *xs*. *star-n step n* (*c*, *fst s*, *xs*)
(*c*′, *fst t*, *xs* @ *new*))
⟨*proof*⟩

lemma *lappend-initial-output*:

$\{llist\text{-}of\ out\ |out.$
$\quad \exists\, q\ t.\ star\ step\ (While\ x\ p,\ a,\ b)\ (q,\ t,\ out)\}$
$=\ lappend\ (llist\text{-}of\ b)\ `\ \{llist\text{-}of\ out\ |out.$
$\quad \exists\, q\ t.\ star\ step\ (While\ x\ p,\ a,\ []\ )\ (q,\ t,\ out)\}$
⟨*proof*⟩

lemma *ts-accum*:
$\forall\, i.\ prefix\ (output\text{-}of\ (ts\ i))\ (output\text{-}of\ (ts\ (Suc\ i))) \Longrightarrow$
$concat\ (map\ (\lambda i.\ drop\ (length\ (output\text{-}of\ (ts\ i)))\ (output\text{-}of\ (ts\ (Suc\ i))))\ [0..<i])$
$= drop\ (length\ (output\text{-}of\ (ts\ 0)))\ (output\text{-}of\ (ts\ i))$
⟨*proof*⟩

theorem *Pohjola-diverges*:
$pohjola\ (\lambda s.\ \exists\, l.\ diverges\ s\ c\ l \wedge D\ l)\ c\ D$
⟨*proof*⟩

theorem *Pohjola-completeness*:
$pohjola\text{-}sem\ P\ c\ D \Longrightarrow pohjola\ P\ c\ D$
⟨*proof*⟩

end

9 Soundness of Hoare logic for diverging programs

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

lemma *p-loop-deterministic*:
$star\text{-}n\ (\lambda s\ t.\ guard\ x\ s \wedge terminates\ s\ p\ t)\ n\ s\ t \Longrightarrow$
$\quad star\text{-}n\ (\lambda s\ t.\ guard\ x\ s \wedge terminates\ s\ p\ t)\ n\ s\ t' \Longrightarrow t = t'$
⟨*proof*⟩

lemma *loop-accum*:
$[\![\forall\, i.\ hoare\ (\lambda s.\ H\ i\ s \wedge output\text{-}of\ s = []\ )\ p$
$\quad (\lambda s.\ H\ (Suc\ i)\ s \wedge output\text{-}of\ s = ev\ i \wedge guard\ x\ s);$
$\quad guard\ x\ (a,\ b);\ H\ 0\ (a,\ b);\ ignores\text{-}output\ H]\!]$
$\quad \Longrightarrow \forall\, i.\ \exists\ s.\ star\text{-}n\ (\lambda s\ t.\ guard\ x\ s \wedge terminates\ s\ p\ t)\ i\ (a,\ b)\ s \wedge$
$\quad\quad guard\ x\ s \wedge H\ i\ s$
⟨*proof*⟩

lemma *output-accum*:
$[\![star\text{-}n\ (\lambda s\ t.\ guard\ x\ s \wedge terminates\ s\ p\ t)\ i\ (a,\ b)\ s;$
$\quad guard\ x\ (a,\ b);\ H\ 0\ (a,\ b);\ ignores\text{-}output\ H;$
$\quad \forall\, i.\ hoare\ (\lambda s.\ H\ i\ s \wedge output\text{-}of\ s = []\ )\ p$
$\quad (\lambda s.\ H\ (Suc\ i)\ s \wedge output\text{-}of\ s = ev\ i \wedge guard\ x\ s)]\!]$
$\quad \Longrightarrow output\text{-}of\ s = b\ @\ (concat\ (map\ ev\ [0..<i]))$
⟨*proof*⟩

lemma *helper-lemma*:
⟦*guard x* (*a*, *b*); *H 0* (*a*, *b*); *ignores-output H*;
∀ *i*. *hoare* (λ*s*. *H i s* ∧ *output-of s* = []) *p*
(λ*s*. *H* (*Suc i*) *s* ∧ *output-of s* = *ev i* ∧ *guard x s*)⟧
⟹ ∀ *i*. ∃ *s*. *star-n* (λ*s t*. *guard x s* ∧ *terminates s p t*) *i* (*a*, *b*) *s* ∧
guard x s ∧ *H i s* ∧ *output-of s* = *b* @ (*concat* (*map ev* [*0*..< *i*]))
⟨*proof*⟩

lemma *add-While-loops*:
⟦*star-n* (λ*s t*. *guard x s* ∧ *terminates s p t*) *i* (*a*, *b*) *s*⟧
⟹ *star-n* (λ*s t*. *star step* (*While x p*, *s*) (*While x p*, *t*) ∧ *terminates s p t* ∧
guard x s)
i (*a*, *b*) *s*
⟨*proof*⟩

lemma *loop-upper-bound*:
⟦∀ *i*. ∃ *s*. *star-n* (λ*s t*. *guard x s* ∧ *terminates s p t*) *i* (*a*, *b*) *s* ∧ *guard x s* ∧ *H i s*;
star-n step n (*While x p*, *a*, []) (*q*, *t*, *out*)⟧ ⟹
∃ *i t′ n′*. *star-n* (λ*s t*. *star step* (*While x p*, *s*) (*While x p*, *t*)
∧ *terminates s p t* ∧ *guard x s*) *i* (*a*, *b*) *t′* ∧
star-n step n′ (*While x p*, *a*, *b*) (*While x p*, *t′*) ∧ *n* ≤ *n′*
⟨*proof*⟩

theorem *Pohjola-soundness*:
pohjola P c Q ⟹ *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*:
pohjola (λ*s*. *output-of s* = []) *pure-loop* (λ*l*. *l* = *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 (λ*s*. *output-of s* = []) *zero* (λ*l*. *l* = *zero-llist*)
⟨*proof*⟩

definition *ex2* **where**
ex2 = *While* (λ-. *1*) *zero*

lemma *ex2-correct*:
pohjola (λ*s*. *output-of s* = []) *ex2* (λ*l*. *l* = *zero-llist*)
⟨*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.