

CT-WASM

anonymous authors

July 11, 2018

Abstract

This is a mechanised specification of the CT-WASM extension to WebAssembly, based on the previous model of [1].

Contents

1	WebAssembly Core AST	2
2	Syntactic Typeclasses	7
3	WebAssembly Base Definitions	9
4	Host Properties	27
5	Auxiliary Type System Properties	28
6	Lemmas for Soundness Proof	41
6.1	Preservation	41
6.2	Progress	49
7	Soundness Theorems	53
8	Augmented Type Syntax for Concrete Checker	53
9	Executable Type Checker	64
10	Correctness of Type Checker	68
10.1	Soundness	68
10.2	Completeness	70
11	Auxiliary Security Properties	71
12	Security Proofs	82
13	Constant Time (coinductive)	89

14 Constant Time (inductive) **91**

15 Set Based Leakage Model (sketch) **92**

1 WebAssembly Core AST

theory *Wasm-Ast* **imports** *Main AFP/Native-Word/Uint8* **begin**

type-synonym — immediate

i = *nat*

type-synonym — static offset

off = *nat*

type-synonym — alignment exponent

a = *nat*

— primitive types

typeddecl *i32*

typeddecl *i64*

typeddecl *f32*

typeddecl *f64*

— memory

type-synonym *byte* = *uint8*

typedef *bytes* = *UNIV* :: (*byte list*) *set* *<proof>*

setup-lifting *type-definition-bytes*

declare *Quotient-bytes*[*transfer-rule*]

lift-definition *bytes-takefill* :: *byte* \Rightarrow *nat* \Rightarrow *bytes* \Rightarrow *bytes* **is** ($\lambda a\ n\ as.$ *takefill* (*Abs-uint8* *a*) *n as*) *<proof>*

lift-definition *bytes-replicate* :: *nat* \Rightarrow *byte* \Rightarrow *bytes* **is** ($\lambda n\ b.$ *replicate* *n* (*Abs-uint8* *b*)) *<proof>*

definition *msbyte* :: *bytes* \Rightarrow *byte* **where**

msbyte *bs* = *last* (*Rep-bytes* *bs*)

typedef *mem* = *UNIV* :: (*byte list*) *set* *<proof>*

setup-lifting *type-definition-mem*

declare *Quotient-mem*[*transfer-rule*]

lift-definition *read-bytes* :: *mem* \Rightarrow *nat* \Rightarrow *nat* \Rightarrow *bytes* **is** ($\lambda m\ n\ l.$ *take* *l* (*drop* *n m*)) *<proof>*

lift-definition *write-bytes* :: *mem* \Rightarrow *nat* \Rightarrow *bytes* \Rightarrow *mem* **is** ($\lambda m\ n\ bs.$ (*take* *n m*) @ *bs* @ (*drop* (*n* + *length* *bs*) *m*)) *<proof>*

lift-definition *mem-append* :: *mem* \Rightarrow *bytes* \Rightarrow *mem* **is** *append* *<proof>*

typeddecl *host*

typeddecl *host-state*

datatype — secrecy type

sec = *Secret* | *Public*

datatype — trust type
trust = *Trusted* | *Untrusted*

datatype — value types
t = *T-i32 sec* | *T-i64 sec* | *T-f32* | *T-f64*

datatype — packed types
tp = *Tp-i8* | *Tp-i16* | *Tp-i32*

datatype — mutability
mut = *T-immut* | *T-mut*

record *tg* = — global types
tg-mut :: *mut*
tg-t :: *t*

datatype — function types
tf = *Tf t list t list* (- ' -> - 60)

type-synonym — function type with trust
tf-t = *trust* × *tf*

record *t-context* =
trust-t :: *trust*
types-t :: *tf-t list*
func-t :: *tf-t list*
global :: *tg list*
table :: *nat option*
memory :: (*nat* × *sec*) *option*
local :: *t list*
label :: (*t list*) *list*
return :: (*t list*) *option*

record *s-context* =
s-inst :: *t-context list*
s-funcs :: *tf-t list*
s-tab :: *nat list*
s-mem :: (*nat* × *sec*) *list*
s-globs :: *tg list*

datatype
sx = *S* | *U*

datatype
unop-i = *Clz* | *Ctz* | *Popcnt*

datatype

$unop-f = Neg \mid Abs \mid Ceil \mid Floor \mid Trunc \mid Nearest \mid Sqrt$

datatype

$binop-i = Add \mid Sub \mid Mul \mid Div\ sx \mid Rem\ sx \mid And \mid Or \mid Xor \mid Shl \mid Shr\ sx \mid Rotl \mid Rotr$

datatype

$binop-f = Addf \mid Subf \mid Mulf \mid Divf \mid Min \mid Max \mid Copysign$

datatype

$testop = Eqz$

datatype

$relop-i = Eq \mid Ne \mid Lt\ sx \mid Gt\ sx \mid Le\ sx \mid Ge\ sx$

datatype

$relop-f = Eqf \mid Nef \mid Ltf \mid Gtf \mid Lef \mid Gef$

datatype

$cvtop = Convert \mid Reinterpret \mid Classify \mid Declassify$

datatype — values

$v =$
 $ConstInt32\ sec\ i32$
 $\mid ConstInt64\ sec\ i64$
 $\mid ConstFloat32\ f32$
 $\mid ConstFloat64\ f64$

datatype — basic instructions

$b-e =$
 $Unreachable$
 $\mid Nop$
 $\mid Drop$
 $\mid Select\ sec$
 $\mid Block\ tf\ b-e\ list$
 $\mid Loop\ tf\ b-e\ list$
 $\mid If\ tf\ b-e\ list\ b-e\ list$
 $\mid Br\ i$
 $\mid Br-if\ i$
 $\mid Br-table\ i\ list\ i$
 $\mid Return$
 $\mid Call\ i$
 $\mid Call-indirect\ i$
 $\mid Get-local\ i$
 $\mid Set-local\ i$
 $\mid Tee-local\ i$
 $\mid Get-global\ i$
 $\mid Set-global\ i$
 $\mid Load\ t\ (tp \times sx)\ option\ a\ off$

```

| Store t tp option a off
| Current-memory
| Grow-memory
| EConst v (C - 60)
| Unop-i t unop-i
| Unop-f t unop-f
| Binop-i t binop-i
| Binop-f t binop-f
| Testop t testop
| Relop-i t relop-i
| Relop-f t relop-f
| Cvtop t cvtop t sx option

datatype cl = — function closures
  Func-native i tf-t t list b-e list
| Func-host tf-t host

record inst = — instances
  types :: tf-t list
  funcs :: i list
  tab :: i option
  mem :: i option
  globs :: i list

type-synonym tabinst = (cl option) list

record global =
  g-mut :: mut
  g-val :: v

record s = — store
  inst :: inst list
  funcs :: cl list
  tab :: tabinst list
  mem :: (mem × sec) list
  globs :: global list

datatype e = — administrative instruction
  Basic b-e ($- 60)
| Trap
| Callcl cl
| Label nat e list e list
| Local nat i v list e list

datatype Lholed =
  — L0 = v* [iholei] e*
  LBase e list e list
  — L(i+1) = v* (label n e* Li) e*
| LRec e list nat e list Lholed e list

```

```

datatype action =
  | Unop-i32-action unop-i
  | Unop-i64-action unop-i
  | Unop-f32-action unop-f f32
  | Unop-f64-action unop-f f64
  | Binop-i32-Some-action binop-i i32 i32
  | Binop-i32-None-action binop-i i32 i32
  | Binop-i64-Some-action binop-i i64 i64
  | Binop-i64-None-action binop-i i64 i64
  | Binop-f32-Some-action binop-f f32 f32
  | Binop-f32-None-action binop-f f32 f32
  | Binop-f64-Some-action binop-f f64 f64
  | Binop-f64-None-action binop-f f64 f64
  | Testop-i32-action testop
  | Testop-i64-action testop
  | Relop-i32-action relop-i
  | Relop-i64-action relop-i
  | Relop-f32-action relop-f f32 f32
  | Relop-f64-action relop-f f64 f64
  | Convert-Some-action t t v
  | Convert-None-action t t v
  | Reinterpret-action
  | Classify-action
  | Declassify-action
  | Unreachable-action
  | Nop-action
  | Drop-action
  | Select-action sec i32
  | Block-action
  | Loop-action
  | If-false-action i32
  | If-true-action i32
  | Label-const-action
  | Label-trap-action
  | Br-action
  | Br-if-false-action i32
  | Br-if-true-action i32
  | Br-table-action i32
  | Br-table-length-action i32
  | Local-const-action
  | Local-trap-action
  | Return-action
  | Tee-local-action
  | Trap-action
  | Call-action
  | Call-indirect-Some-action i32
  | Call-indirect-None-action i32
  | Callcl-native-action nat

```

```

| Callcl-host-Some-action s v list s v list trust tf host host-state
| Callcl-host-None-action s v list trust tf host host-state
| Get-local-action
| Set-local-action
| Get-global-action
| Set-global-action
| Load-Some-action t nat a off
| Load-None-action t nat a off
| Load-packed-Some-action tp sx nat a off
| Load-packed-None-action tp sx nat a off
| Store-Some-action t nat a off
| Store-None-action t nat a off
| Store-packed-Some-action t tp nat a off
| Store-packed-None-action t tp nat a off
| Current-memory-action nat
| Grow-memory-Some-action nat nat
| Grow-memory-None-action nat nat
| Label-action
| Local-action

```

end

2 Syntactic Typeclasses

theory *Wasm-Type-Abs* **imports** *Main* **begin**

class *wasm-base* = *zero*

class *wasm-int* = *wasm-base* +

```

fixes int-clz :: 'a ⇒ 'a
fixes int-ctz :: 'a ⇒ 'a
fixes int-popcnt :: 'a ⇒ 'a

fixes int-add :: 'a ⇒ 'a ⇒ 'a
fixes int-sub :: 'a ⇒ 'a ⇒ 'a
fixes int-mul :: 'a ⇒ 'a ⇒ 'a
fixes int-div-u :: 'a ⇒ 'a ⇒ 'a option
fixes int-div-s :: 'a ⇒ 'a ⇒ 'a option
fixes int-rem-u :: 'a ⇒ 'a ⇒ 'a option
fixes int-rem-s :: 'a ⇒ 'a ⇒ 'a option
fixes int-and :: 'a ⇒ 'a ⇒ 'a
fixes int-or :: 'a ⇒ 'a ⇒ 'a
fixes int-xor :: 'a ⇒ 'a ⇒ 'a
fixes int-shl :: 'a ⇒ 'a ⇒ 'a
fixes int-shr-u :: 'a ⇒ 'a ⇒ 'a
fixes int-shr-s :: 'a ⇒ 'a ⇒ 'a
fixes int-rotr :: 'a ⇒ 'a ⇒ 'a
fixes int-rotl :: 'a ⇒ 'a ⇒ 'a

```

```

fixes int-eqz :: 'a ⇒ bool

fixes int-eq :: 'a ⇒ 'a ⇒ bool
fixes int-lt-u :: 'a ⇒ 'a ⇒ bool
fixes int-lt-s :: 'a ⇒ 'a ⇒ bool
fixes int-gt-u :: 'a ⇒ 'a ⇒ bool
fixes int-gt-s :: 'a ⇒ 'a ⇒ bool
fixes int-le-u :: 'a ⇒ 'a ⇒ bool
fixes int-le-s :: 'a ⇒ 'a ⇒ bool
fixes int-ge-u :: 'a ⇒ 'a ⇒ bool
fixes int-ge-s :: 'a ⇒ 'a ⇒ bool

fixes int-of-nat :: nat ⇒ 'a
fixes nat-of-int :: 'a ⇒ nat
begin
  abbreviation (input)
    int-ne where
      int-ne x y ≡ ¬ (int-eq x y)
end

class wasm-float = wasm-base +

  fixes float-neg    :: 'a ⇒ 'a
  fixes float-abs   :: 'a ⇒ 'a
  fixes float-ceil  :: 'a ⇒ 'a
  fixes float-floor :: 'a ⇒ 'a
  fixes float-trunc :: 'a ⇒ 'a
  fixes float-nearest :: 'a ⇒ 'a
  fixes float-sqrt  :: 'a ⇒ 'a

  fixes float-add :: 'a ⇒ 'a ⇒ 'a
  fixes float-sub :: 'a ⇒ 'a ⇒ 'a
  fixes float-mul :: 'a ⇒ 'a ⇒ 'a
  fixes float-div :: 'a ⇒ 'a ⇒ 'a
  fixes float-min :: 'a ⇒ 'a ⇒ 'a
  fixes float-max :: 'a ⇒ 'a ⇒ 'a
  fixes float-copysign :: 'a ⇒ 'a ⇒ 'a

  fixes float-eq :: 'a ⇒ 'a ⇒ bool
  fixes float-lt :: 'a ⇒ 'a ⇒ bool
  fixes float-gt :: 'a ⇒ 'a ⇒ bool
  fixes float-le :: 'a ⇒ 'a ⇒ bool
  fixes float-ge :: 'a ⇒ 'a ⇒ bool
begin
  abbreviation (input)
    float-ne where
      float-ne x y ≡ ¬ (float-eq x y)
end

```


end

3 WebAssembly Base Definitions

theory *Wasm-Base-Defs* **imports** *Wasm-Ast Wasm-Type-Abs* **begin**

instantiation *i32* :: *wasm-int* **begin instance** $\langle proof \rangle$ **end**
instantiation *i64* :: *wasm-int* **begin instance** $\langle proof \rangle$ **end**
instantiation *f32* :: *wasm-float* **begin instance** $\langle proof \rangle$ **end**
instantiation *f64* :: *wasm-float* **begin instance** $\langle proof \rangle$ **end**

consts

ui32-trunc-f32 :: *f32* \Rightarrow *i32 option*
si32-trunc-f32 :: *f32* \Rightarrow *i32 option*
ui32-trunc-f64 :: *f64* \Rightarrow *i32 option*
si32-trunc-f64 :: *f64* \Rightarrow *i32 option*

ui64-trunc-f32 :: *f32* \Rightarrow *i64 option*
si64-trunc-f32 :: *f32* \Rightarrow *i64 option*
ui64-trunc-f64 :: *f64* \Rightarrow *i64 option*
si64-trunc-f64 :: *f64* \Rightarrow *i64 option*

f32-convert-ui32 :: *i32* \Rightarrow *f32*
f32-convert-si32 :: *i32* \Rightarrow *f32*
f32-convert-ui64 :: *i64* \Rightarrow *f32*
f32-convert-si64 :: *i64* \Rightarrow *f32*

f64-convert-ui32 :: *i32* \Rightarrow *f64*
f64-convert-si32 :: *i32* \Rightarrow *f64*
f64-convert-ui64 :: *i64* \Rightarrow *f64*
f64-convert-si64 :: *i64* \Rightarrow *f64*

wasm-wrap :: *i64* \Rightarrow *i32*
wasm-extend-u :: *i32* \Rightarrow *i64*
wasm-extend-s :: *i32* \Rightarrow *i64*
wasm-demote :: *f64* \Rightarrow *f32*
wasm-promote :: *f32* \Rightarrow *f64*

serialise-i32 :: *i32* \Rightarrow *bytes*
serialise-i64 :: *i64* \Rightarrow *bytes*
serialise-f32 :: *f32* \Rightarrow *bytes*
serialise-f64 :: *f64* \Rightarrow *bytes*
wasm-bool :: *bool* \Rightarrow *i32*
int32-minus-one :: *i32*

definition *mem-size* :: *mem* \Rightarrow *nat* **where**

$mem\text{-}size\ m = length\ (Rep\text{-}mem\ m)$

definition $mem\text{-}grow :: mem \Rightarrow nat \Rightarrow mem$ **where**
 $mem\text{-}grow\ m\ n = mem\text{-}append\ m\ (bytes\text{-}replicate\ (n * 64000)\ 0)$

definition $load :: mem \Rightarrow nat \Rightarrow off \Rightarrow nat \Rightarrow bytes\ option$ **where**
 $load\ m\ n\ off\ l = (if\ (mem\text{-}size\ m \geq (n + off + l))$
 $\quad then\ Some\ (read\text{-}bytes\ m\ (n + off)\ l)$
 $\quad else\ None)$

definition $sign\text{-}extend :: sx \Rightarrow nat \Rightarrow bytes \Rightarrow bytes$ **where**
 $sign\text{-}extend\ sx\ l\ bytes = (let\ msb = msb\ (msbyte\ bytes)\ in$
 $\quad let\ byte = (case\ sx\ of\ U \Rightarrow 0 \mid S \Rightarrow if\ msb\ then\ -1\ else\ 0)\ in$
 $\quad bytes\text{-}takefill\ byte\ l\ bytes)$

definition $load\text{-}packed :: sx \Rightarrow mem \Rightarrow nat \Rightarrow off \Rightarrow nat \Rightarrow nat \Rightarrow bytes\ option$
where
 $load\text{-}packed\ sx\ m\ n\ off\ lp\ l = map\text{-}option\ (sign\text{-}extend\ sx\ l)\ (load\ m\ n\ off\ lp)$

definition $store :: mem \Rightarrow nat \Rightarrow off \Rightarrow bytes \Rightarrow nat \Rightarrow mem\ option$ **where**
 $store\ m\ n\ off\ bs\ l = (if\ (mem\text{-}size\ m \geq (n + off + l))$
 $\quad then\ Some\ (write\text{-}bytes\ m\ (n + off)\ (bytes\text{-}takefill\ 0\ l\ bs))$
 $\quad else\ None)$

definition $store\text{-}packed :: mem \Rightarrow nat \Rightarrow off \Rightarrow bytes \Rightarrow nat \Rightarrow mem\ option$
where
 $store\text{-}packed = store$

consts
 $wasm\text{-}deserialise :: bytes \Rightarrow t \Rightarrow v$

$host\text{-}apply :: s \Rightarrow tf \Rightarrow host \Rightarrow v\ list \Rightarrow host\text{-}state \Rightarrow (s \times v\ list)\ option$

definition $typeof :: v \Rightarrow t$ **where**
 $typeof\ v = (case\ v\ of$
 $\quad ConstInt32\ sec - \Rightarrow (T\text{-}i32\ sec)$
 $\quad \mid ConstInt64\ sec - \Rightarrow (T\text{-}i64\ sec)$
 $\quad \mid ConstFloat32 - \Rightarrow T\text{-}f32$
 $\quad \mid ConstFloat64 - \Rightarrow T\text{-}f64)$

definition $trust\text{-}compat :: trust \Rightarrow trust \Rightarrow bool$ **where**
 $trust\text{-}compat\ tr\ tr' = (tr = Trusted \vee (tr = Untrusted \wedge tr' = Untrusted))$

definition $classify\text{-}t :: t \Rightarrow t$ **where**
 $classify\text{-}t\ t = (case\ t\ of$
 $\quad T\text{-}i32 - \Rightarrow T\text{-}i32\ Secret$
 $\quad \mid T\text{-}i64 - \Rightarrow T\text{-}i64\ Secret$
 $\quad \mid T\text{-}f32 \Rightarrow T\text{-}f32$
 $\quad \mid T\text{-}f64 \Rightarrow T\text{-}f64)$

definition *classify* :: $v \Rightarrow v$ **where**

classify $v = (\text{case } v \text{ of}$
 $\text{ConstInt32 } \text{sec } c \Rightarrow \text{ConstInt32 } \text{Secret } c$
 $| \text{ConstInt64 } \text{sec } c \Rightarrow \text{ConstInt64 } \text{Secret } c$
 $| \text{ConstFloat32 } c \Rightarrow \text{ConstFloat32 } c$
 $| \text{ConstFloat64 } c \Rightarrow \text{ConstFloat64 } c)$

definition *declassify-t* :: $t \Rightarrow t$ **where**

declassify-t $t = (\text{case } t \text{ of}$
 $T\text{-}i32 \Rightarrow T\text{-}i32 \text{ Public}$
 $| T\text{-}i64 \Rightarrow T\text{-}i64 \text{ Public}$
 $| T\text{-}f32 \Rightarrow T\text{-}f32$
 $| T\text{-}f64 \Rightarrow T\text{-}f64)$

definition *declassify* :: $v \Rightarrow v$ **where**

declassify $v = (\text{case } v \text{ of}$
 $\text{ConstInt32 } \text{sec } c \Rightarrow \text{ConstInt32 } \text{Public } c$
 $| \text{ConstInt64 } \text{sec } c \Rightarrow \text{ConstInt64 } \text{Public } c$
 $| \text{ConstFloat32 } c \Rightarrow \text{ConstFloat32 } c$
 $| \text{ConstFloat64 } c \Rightarrow \text{ConstFloat64 } c)$

definition *option-projl* :: $('a \times 'b) \text{ option} \Rightarrow 'a \text{ option}$ **where**

option-projl $x = \text{map-option } \text{fst } x$

definition *option-projr* :: $('a \times 'b) \text{ option} \Rightarrow 'b \text{ option}$ **where**

option-projr $x = \text{map-option } \text{snd } x$

definition *t-length* :: $t \Rightarrow \text{nat}$ **where**

t-length $t = (\text{case } t \text{ of}$
 $T\text{-}i32 \Rightarrow 4$
 $| T\text{-}i64 \Rightarrow 8$
 $| T\text{-}f32 \Rightarrow 4$
 $| T\text{-}f64 \Rightarrow 8)$

definition *tp-length* :: $tp \Rightarrow \text{nat}$ **where**

tp-length $tp = (\text{case } tp \text{ of}$
 $Tp\text{-}i8 \Rightarrow 1$
 $| Tp\text{-}i16 \Rightarrow 2$
 $| Tp\text{-}i32 \Rightarrow 4)$

definition *t-sec* :: $t \Rightarrow \text{sec}$ **where**

t-sec $t = (\text{case } t \text{ of}$
 $T\text{-}i32 \text{ sec} \Rightarrow \text{sec}$
 $| T\text{-}i64 \text{ sec} \Rightarrow \text{sec}$
 $| T\text{-}f32 \Rightarrow \text{Public}$
 $| T\text{-}f64 \Rightarrow \text{Public})$

abbreviation *is-public-t* :: $t \Rightarrow \text{bool}$ **where**

$is-public-t\ t \equiv ((t-sec\ t) = Public)$

abbreviation $is-secret-t :: t \Rightarrow bool$ **where**

$is-secret-t\ t \equiv ((t-sec\ t) = Secret)$

definition $is-int-t :: t \Rightarrow bool$ **where**

$is-int-t\ t = (case\ t\ of$
 $\quad T-i32\ - \Rightarrow True$
 $\quad | T-i64\ - \Rightarrow True$
 $\quad | T-f32\ \Rightarrow False$
 $\quad | T-f64\ \Rightarrow False)$

definition $is-float-t :: t \Rightarrow bool$ **where**

$is-float-t\ t = (case\ t\ of$
 $\quad T-i32\ - \Rightarrow False$
 $\quad | T-i64\ - \Rightarrow False$
 $\quad | T-f32\ \Rightarrow True$
 $\quad | T-f64\ \Rightarrow True)$

definition $is-mut :: tg \Rightarrow bool$ **where**

$is-mut\ tg = (tg-mut\ tg = T-mut)$

definition $safe-binop-i :: binop-i \Rightarrow bool$ **where**

$safe-binop-i\ bop =$
 $(case\ bop\ of$
 $\quad Div\ - \Rightarrow False$
 $\quad | Rem\ - \Rightarrow False$
 $\quad | - \Rightarrow True)$

definition $app-unop-i :: unop-i \Rightarrow 'i::wasm-int \Rightarrow 'i::wasm-int$ **where**

$app-unop-i\ iop\ c =$
 $(case\ iop\ of$
 $\quad Ctz \Rightarrow int-ctz\ c$
 $\quad | Clz \Rightarrow int-clz\ c$
 $\quad | Popcnt \Rightarrow int-popcnt\ c)$

definition $app-unop-f :: unop-f \Rightarrow 'f::wasm-float \Rightarrow 'f::wasm-float$ **where**

$app-unop-f\ fop\ c =$
 $(case\ fop\ of$
 $\quad Neg \Rightarrow float-neg\ c$
 $\quad | Abs \Rightarrow float-abs\ c$
 $\quad | Ceil \Rightarrow float-ceil\ c$
 $\quad | Floor \Rightarrow float-floor\ c$
 $\quad | Trunc \Rightarrow float-trunc\ c$
 $\quad | Nearest \Rightarrow float-nearest\ c$
 $\quad | Sqrt \Rightarrow float-sqrt\ c)$

definition $app-binop-i :: binop-i \Rightarrow 'i::wasm-int \Rightarrow 'i::wasm-int \Rightarrow ('i::wasm-int)$
option where

$app\text{-}binop\text{-}i\ iop\ c1\ c2 = (case\ iop\ of$
 $\quad Add \Rightarrow Some\ (int\text{-}add\ c1\ c2)$
 $\quad | Sub \Rightarrow Some\ (int\text{-}sub\ c1\ c2)$
 $\quad | Mul \Rightarrow Some\ (int\text{-}mul\ c1\ c2)$
 $\quad | Div\ U \Rightarrow int\text{-}div\text{-}u\ c1\ c2$
 $\quad | Div\ S \Rightarrow int\text{-}div\text{-}s\ c1\ c2$
 $\quad | Rem\ U \Rightarrow int\text{-}rem\text{-}u\ c1\ c2$
 $\quad | Rem\ S \Rightarrow int\text{-}rem\text{-}s\ c1\ c2$
 $\quad | And \Rightarrow Some\ (int\text{-}and\ c1\ c2)$
 $\quad | Or \Rightarrow Some\ (int\text{-}or\ c1\ c2)$
 $\quad | Xor \Rightarrow Some\ (int\text{-}xor\ c1\ c2)$
 $\quad | Shl \Rightarrow Some\ (int\text{-}shl\ c1\ c2)$
 $\quad | Shr\ U \Rightarrow Some\ (int\text{-}shr\text{-}u\ c1\ c2)$
 $\quad | Shr\ S \Rightarrow Some\ (int\text{-}shr\text{-}s\ c1\ c2)$
 $\quad | Rotl \Rightarrow Some\ (int\text{-}rotr\ c1\ c2)$
 $\quad | Rotr \Rightarrow Some\ (int\text{-}rotr\ c1\ c2))$

definition $app\text{-}binop\text{-}f :: binop\text{-}f \Rightarrow 'f::wasm\text{-}float \Rightarrow 'f::wasm\text{-}float \Rightarrow ('f::wasm\text{-}float)$
option where

$app\text{-}binop\text{-}f\ fop\ c1\ c2 = (case\ fop\ of$
 $\quad Addf \Rightarrow Some\ (float\text{-}add\ c1\ c2)$
 $\quad | Subf \Rightarrow Some\ (float\text{-}sub\ c1\ c2)$
 $\quad | Mulf \Rightarrow Some\ (float\text{-}mul\ c1\ c2)$
 $\quad | Divf \Rightarrow Some\ (float\text{-}div\ c1\ c2)$
 $\quad | Min \Rightarrow Some\ (float\text{-}min\ c1\ c2)$
 $\quad | Max \Rightarrow Some\ (float\text{-}max\ c1\ c2)$
 $\quad | Copysign \Rightarrow Some\ (float\text{-}copysign\ c1\ c2))$

definition $app\text{-}testop\text{-}i :: testop \Rightarrow 'i::wasm\text{-}int \Rightarrow bool$ **where**
 $app\text{-}testop\text{-}i\ testop\ c = (case\ testop\ of\ Eqz \Rightarrow int\text{-}eqz\ c)$

definition $app\text{-}relop\text{-}i :: relop\text{-}i \Rightarrow 'i::wasm\text{-}int \Rightarrow 'i::wasm\text{-}int \Rightarrow bool$ **where**
 $app\text{-}relop\text{-}i\ rop\ c1\ c2 = (case\ rop\ of$

$\quad Eq \Rightarrow int\text{-}eq\ c1\ c2$
 $\quad | Ne \Rightarrow int\text{-}ne\ c1\ c2$
 $\quad | Lt\ U \Rightarrow int\text{-}lt\text{-}u\ c1\ c2$
 $\quad | Lt\ S \Rightarrow int\text{-}lt\text{-}s\ c1\ c2$
 $\quad | Gt\ U \Rightarrow int\text{-}gt\text{-}u\ c1\ c2$
 $\quad | Gt\ S \Rightarrow int\text{-}gt\text{-}s\ c1\ c2$
 $\quad | Le\ U \Rightarrow int\text{-}le\text{-}u\ c1\ c2$
 $\quad | Le\ S \Rightarrow int\text{-}le\text{-}s\ c1\ c2$
 $\quad | Ge\ U \Rightarrow int\text{-}ge\text{-}u\ c1\ c2$
 $\quad | Ge\ S \Rightarrow int\text{-}ge\text{-}s\ c1\ c2)$

definition $app\text{-}relop\text{-}f :: relop\text{-}f \Rightarrow 'f::wasm\text{-}float \Rightarrow 'f::wasm\text{-}float \Rightarrow bool$ **where**
 $app\text{-}relop\text{-}f\ rop\ c1\ c2 = (case\ rop\ of$

$\quad Eqf \Rightarrow float\text{-}eq\ c1\ c2$
 $\quad | Nef \Rightarrow float\text{-}ne\ c1\ c2$
 $\quad | Ltf \Rightarrow float\text{-}lt\ c1\ c2$

| $Gtf \Rightarrow \text{float-gt } c1 \ c2$
| $Lef \Rightarrow \text{float-le } c1 \ c2$
| $Gef \Rightarrow \text{float-ge } c1 \ c2$)

definition *types-agree* :: $t \Rightarrow v \Rightarrow \text{bool}$ **where**
types-agree $t \ v = (\text{typeof } v = t)$

definition *types-agree-insecure* :: $t \Rightarrow v \Rightarrow \text{bool}$ **where**
types-agree-insecure $t \ v = (\text{let } v\text{-}t = \text{typeof } v \text{ in}$
 $\text{is-int-}t \ v\text{-}t = \text{is-int-}t \ t \wedge t\text{-length } v\text{-}t = t\text{-length } t)$

definition *cl-type* :: $cl \Rightarrow \text{tf-}t$ **where**
cl-type $cl = (\text{case } cl \text{ of } \text{Func-native } - \text{ tf } - \Rightarrow \text{tf} \mid \text{Func-host } \text{tf } - \Rightarrow \text{tf})$

definition *rglob-is-mut* :: $\text{global} \Rightarrow \text{bool}$ **where**
rglob-is-mut $g = (g\text{-mut } g = T\text{-mut})$

definition *stypes* :: $s \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow \text{tf-}t$ **where**
stypes $s \ i \ j = ((\text{types } ((\text{inst } s)!i))!j)$

definition *sfunc-ind* :: $s \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat}$ **where**
sfunc-ind $s \ i \ j = ((\text{inst.funcs } ((\text{inst } s)!i))!j)$

definition *sfunc* :: $s \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow \text{cl}$ **where**
sfunc $s \ i \ j = (\text{funcs } s)!(\text{sfunc-ind } s \ i \ j)$

definition *sglob-ind* :: $s \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat}$ **where**
sglob-ind $s \ i \ j = ((\text{inst.globs } ((\text{inst } s)!i))!j)$

definition *sglob* :: $s \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow \text{global}$ **where**
sglob $s \ i \ j = (\text{globs } s)!(\text{sglob-ind } s \ i \ j)$

definition *sglob-val* :: $s \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow v$ **where**
sglob-val $s \ i \ j = g\text{-val } (\text{sglob } s \ i \ j)$

definition *smem-ind* :: $s \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow \text{option}$ **where**
smem-ind $s \ i = (\text{inst.mem } ((\text{inst } s)!i))$

definition *stab-s* :: $s \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow \text{cl option}$ **where**
stab-s $s \ i \ j = (\text{let } \text{stabinst} = ((\text{tab } s)!i) \text{ in } (\text{if } (\text{length } (\text{stabinst})) > j) \text{ then } (\text{stabinst}!j) \text{ else None}))$

definition *stab* :: $s \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow \text{cl option}$ **where**
stab $s \ i \ j = (\text{case } (\text{inst.tab } ((\text{inst } s)!i)) \text{ of } \text{Some } k \Rightarrow \text{stab-s } s \ k \ j \mid \text{None} \Rightarrow \text{None})$

definition *supdate-glob-s* :: $s \Rightarrow \text{nat} \Rightarrow v \Rightarrow s$ **where**
supdate-glob-s $s \ k \ v = s[\text{globs} := (\text{globs } s)[k := ((\text{globs } s)!k)(\text{g-val} := v)]]$

definition $\text{supdate-glob} :: s \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow v \Rightarrow s$ **where**
 $\text{supdate-glob } s \ i \ j \ v = (\text{let } k = \text{sglob-ind } s \ i \ j \text{ in } \text{supdate-glob-s } s \ k \ v)$

definition $\text{is-const} :: e \Rightarrow \text{bool}$ **where**
 $\text{is-const } e = (\text{case } e \text{ of } \text{Basic } (C \ -) \Rightarrow \text{True} \mid - \Rightarrow \text{False})$

definition $\text{const-list} :: e \text{ list} \Rightarrow \text{bool}$ **where**
 $\text{const-list } xs = \text{list-all is-const } xs$

inductive $\text{store-extension} :: s \Rightarrow s \Rightarrow \text{bool}$ **where**
 $\llbracket \text{insts} = \text{insts}' ; fs = fs' ; \text{tclss} = \text{tclss}' ; \text{list-all2 } (\lambda(bs, sec) (bs', sec'). \text{mem-size } bs \leq \text{mem-size } bs' \wedge sec = sec') \text{ bss } bss' ; gs = gs' \rrbracket \Longrightarrow$
 $\text{store-extension } (s.\text{inst} = \text{insts}, s.\text{funcs} = fs, s.\text{tab} = \text{tclss}, s.\text{mem} = bss, s.\text{globs} = gs)$
 $(s.\text{inst} = \text{insts}', s.\text{funcs} = fs', s.\text{tab} = \text{tclss}', s.\text{mem} = bss', s.\text{globs} = gs')$

abbreviation $\text{to-e-list} :: b\text{-e list} \Rightarrow e \text{ list}$ ($\$*$ - 60) **where**
 $\text{to-e-list } b\text{-es} \equiv \text{map Basic } b\text{-es}$

abbreviation $\text{v-to-e-list} :: v \text{ list} \Rightarrow e \text{ list}$ ($\$*$ - 60) **where**
 $\text{v-to-e-list } ves \equiv \text{map } (\lambda v. \$C \ v) \ ves$

inductive $L\text{filled} :: \text{nat} \Rightarrow L\text{holed} \Rightarrow e \text{ list} \Rightarrow e \text{ list} \Rightarrow \text{bool}$ **where**

$L0 : \llbracket \text{const-list } vs ; \text{lholed} = (L\text{Base } vs \text{ es}') \rrbracket \Longrightarrow L\text{filled } 0 \text{ lholed } es \ (vs @ es @ es')$
 $\mid LN : \llbracket \text{const-list } vs ; \text{lholed} = (L\text{Rec } vs \ n \text{ es}' \ l \text{ es}'') ; L\text{filled } k \ l \text{ es } l\text{filledk} \rrbracket \Longrightarrow L\text{filled}$
 $(k+1) \text{ lholed } es \ (vs @ [Label \ n \text{ es}' \ l\text{filledk}] @ es'')$

inductive $L\text{filled-exact} :: \text{nat} \Rightarrow L\text{holed} \Rightarrow e \text{ list} \Rightarrow e \text{ list} \Rightarrow \text{bool}$ **where**

$L0 : \llbracket \text{lholed} = (L\text{Base } [] \ []) \rrbracket \Longrightarrow L\text{filled-exact } 0 \text{ lholed } es \ es$
 $\mid LN : \llbracket \text{const-list } vs ; \text{lholed} = (L\text{Rec } vs \ n \text{ es}' \ l \text{ es}'') ; L\text{filled-exact } k \ l \text{ es } l\text{filledk} \rrbracket \Longrightarrow$
 $L\text{filled-exact } (k+1) \text{ lholed } es \ (vs @ [Label \ n \text{ es}' \ l\text{filledk}] @ es'')$

definition $\text{load-store-t-bounds} :: a \Rightarrow tp \text{ option} \Rightarrow t \Rightarrow \text{bool}$ **where**
 $\text{load-store-t-bounds } a \ tp \ t = (\text{case } tp \text{ of}$
 $\text{None} \Rightarrow 2^a \leq t\text{-length } t$
 $\mid \text{Some } tp \Rightarrow 2^a \leq tp\text{-length } tp \wedge tp\text{-length } tp < t\text{-length}$
 $t \wedge \text{is-int-t } t)$

definition $\text{memory-public-agree} :: (\text{mem} \times \text{sec}) \Rightarrow (\text{mem} \times \text{sec}) \Rightarrow \text{bool}$ **where**
 $\text{memory-public-agree } x \ y = (x = y \vee (\text{mem-size } (fst \ x) = \text{mem-size } (fst \ y) \wedge$
 $(snd \ x = \text{Secret}) \wedge (snd \ y = \text{Secret})))$

abbreviation *memories-public-agree* :: (*mem* × *sec*) *list* ⇒ (*mem* × *sec*) *list* ⇒ *bool* **where**

memories-public-agree *xs ys* ≡ *list-all2* *memory-public-agree* *xs ys*

definition *public-agree* :: *v* ⇒ *v* ⇒ *bool* **where**

public-agree *x y* = (*y* = *x* ∨ ((*typeof* *y*) = (*typeof* *x*) ∧ *is-secret-t* (*typeof* *x*)))

abbreviation *publics-agree* :: *v* *list* ⇒ *v* *list* ⇒ *bool* **where**

publics-agree *xs ys* ≡ *list-all2* *public-agree* *xs ys*

definition *global-public-agree* :: *global* ⇒ *global* ⇒ *bool* **where**

global-public-agree *x y* = (*g-mut* *x* = *g-mut* *y* ∧ *public-agree* (*g-val* *x*) (*g-val* *y*))

abbreviation *globals-public-agree* :: *global* *list* ⇒ *global* *list* ⇒ *bool* **where**

globals-public-agree *xs ys* ≡ *list-all2* *global-public-agree* *xs ys*

definition *store-public-agree* :: *s* ⇒ *s* ⇒ *bool* **where**

store-public-agree *s s'* = (*inst* *s* = *inst* *s'* ∧
funcs *s* = *funcs* *s'* ∧
tab *s* = *tab* *s'* ∧
memories-public-agree (*mem* *s*) (*mem* *s'*) ∧
globals-public-agree (*globs* *s*) (*globs* *s'*))

inductive *expr-public-agree* :: *e* ⇒ *e* ⇒ *bool* **where**

expr-public-agree *e e*
| *public-agree* *v v'* ⇒
expr-public-agree (*\$C* *v*) (*\$C* *v'*)
| *list-all2* *expr-public-agree* (*\$** *bes*) (*\$** *bes'*) ⇒
expr-public-agree (*\$Block* *tf* *bes*) (*\$Block* *tf* *bes'*)
| *list-all2* *expr-public-agree* (*\$** *bes*) (*\$** *bes'*) ⇒
expr-public-agree (*\$Loop* *tf* *bes*) (*\$Loop* *tf* *bes'*)
| *list-all2* *expr-public-agree* (*\$** *bes1*) (*\$** *bes1'*); *list-all2* *expr-public-agree* (*\$** *bes2*) (*\$** *bes2'*) ⇒
expr-public-agree (*\$If* *tf* *bes1* *bes2*) (*\$If* *tf* *bes1'* *bes2'*)
| *list-all2* *expr-public-agree* *les les'*; *list-all2* *expr-public-agree* *es es'* ⇒
expr-public-agree (*Label* *n* *les* *es*) (*Label* *n* *les'* *es'*)
| *publics-agree* *vs vs'*; *list-all2* *expr-public-agree* *es es'* ⇒
expr-public-agree (*Local* *n* *i* *vs* *es*) (*Local* *n* *i* *vs'* *es'*)

abbreviation *exprs-public-agree* :: *e* *list* ⇒ *e* *list* ⇒ *bool* **where**

exprs-public-agree *es es'* ≡ *list-all2* *expr-public-agree* *es es'*

inductive *lholed-public-agree* :: *Lholed* ⇒ *Lholed* ⇒ *bool* **where**

exprs-public-agree *ves ves'*; *exprs-public-agree* *es es'* ⇒ *lholed-public-agree* (*LBase* *ves* *es*) (*LBase* *ves'* *es'*)
| *lholed-public-agree* *LN LN'*; *exprs-public-agree* *ves ves'*; *exprs-public-agree* *les les'*;
exprs-public-agree *es es'* ⇒
lholed-public-agree (*LRec* *ves* *n* *les* *LN* *es*) (*LRec* *ves'* *n* *les'* *LN'* *es'*)

definition $cvt-i32 :: sx\ option \Rightarrow v \Rightarrow i32\ option$ **where**

$$\begin{aligned}
cvt-i32\ sx\ v = & (case\ v\ of \\
& ConstInt32\ -\ c \Rightarrow None \\
& | ConstInt64\ -\ c \Rightarrow Some\ (wasm-wrap\ c) \\
& | ConstFloat32\ c \Rightarrow (case\ sx\ of \\
& \quad Some\ U \Rightarrow ui32-trunc-f32\ c \\
& \quad | Some\ S \Rightarrow si32-trunc-f32\ c \\
& \quad | None \Rightarrow None) \\
& | ConstFloat64\ c \Rightarrow (case\ sx\ of \\
& \quad Some\ U \Rightarrow ui32-trunc-f64\ c \\
& \quad | Some\ S \Rightarrow si32-trunc-f64\ c \\
& \quad | None \Rightarrow None))
\end{aligned}$$

definition $cvt-i64 :: sx\ option \Rightarrow v \Rightarrow i64\ option$ **where**

$$\begin{aligned}
cvt-i64\ sx\ v = & (case\ v\ of \\
& ConstInt32\ -\ c \Rightarrow (case\ sx\ of \\
& \quad Some\ U \Rightarrow Some\ (wasm-extend-u\ c) \\
& \quad | Some\ S \Rightarrow Some\ (wasm-extend-s\ c) \\
& \quad | None \Rightarrow None) \\
& | ConstInt64\ -\ c \Rightarrow None \\
& | ConstFloat32\ c \Rightarrow (case\ sx\ of \\
& \quad Some\ U \Rightarrow ui64-trunc-f32\ c \\
& \quad | Some\ S \Rightarrow si64-trunc-f32\ c \\
& \quad | None \Rightarrow None) \\
& | ConstFloat64\ c \Rightarrow (case\ sx\ of \\
& \quad Some\ U \Rightarrow ui64-trunc-f64\ c \\
& \quad | Some\ S \Rightarrow si64-trunc-f64\ c \\
& \quad | None \Rightarrow None))
\end{aligned}$$

definition $cvt-f32 :: sx\ option \Rightarrow v \Rightarrow f32\ option$ **where**

$$\begin{aligned}
cvt-f32\ sx\ v = & (case\ v\ of \\
& ConstInt32\ -\ c \Rightarrow (case\ sx\ of \\
& \quad Some\ U \Rightarrow Some\ (f32-convert-ui32\ c) \\
& \quad | Some\ S \Rightarrow Some\ (f32-convert-si32\ c) \\
& \quad | - \Rightarrow None) \\
& | ConstInt64\ -\ c \Rightarrow (case\ sx\ of \\
& \quad Some\ U \Rightarrow Some\ (f32-convert-ui64\ c) \\
& \quad | Some\ S \Rightarrow Some\ (f32-convert-si64\ c) \\
& \quad | - \Rightarrow None) \\
& | ConstFloat32\ c \Rightarrow None \\
& | ConstFloat64\ c \Rightarrow Some\ (wasm-demote\ c))
\end{aligned}$$

definition $cvt-f64 :: sx\ option \Rightarrow v \Rightarrow f64\ option$ **where**

$$\begin{aligned}
cvt-f64\ sx\ v = & (case\ v\ of \\
& ConstInt32\ -\ c \Rightarrow (case\ sx\ of \\
& \quad Some\ U \Rightarrow Some\ (f64-convert-ui32\ c) \\
& \quad | Some\ S \Rightarrow Some\ (f64-convert-si32\ c) \\
& \quad | - \Rightarrow None)
\end{aligned}$$

$\mid \text{ConstInt64} - c \Rightarrow (\text{case } sx \text{ of}$
 $\quad \text{Some } U \Rightarrow \text{Some } (f64\text{-convert-ui64 } c)$
 $\quad \mid \text{Some } S \Rightarrow \text{Some } (f64\text{-convert-si64 } c)$
 $\quad \mid - \Rightarrow \text{None})$
 $\mid \text{ConstFloat32 } c \Rightarrow \text{Some } (wasm\text{-promote } c)$
 $\mid \text{ConstFloat64 } c \Rightarrow \text{None})$

definition $cvt :: t \Rightarrow sx \text{ option} \Rightarrow v \Rightarrow v \text{ option}$ **where**

$cvt \ t \ sx \ v = (\text{case } t \text{ of}$
 $\quad (T\text{-i32 } sec) \Rightarrow (\text{case } (cvt\text{-i32 } sx \ v) \text{ of } \text{Some } c \Rightarrow \text{Some } (\text{ConstInt32 } sec \ c) \mid \text{None} \Rightarrow \text{None})$
 $\quad \mid (T\text{-i64 } sec) \Rightarrow (\text{case } (cvt\text{-i64 } sx \ v) \text{ of } \text{Some } c \Rightarrow \text{Some } (\text{ConstInt64 } sec \ c) \mid \text{None} \Rightarrow \text{None})$
 $\quad \mid T\text{-f32} \Rightarrow (\text{case } (cvt\text{-f32 } sx \ v) \text{ of } \text{Some } c \Rightarrow \text{Some } (\text{ConstFloat32 } c) \mid \text{None} \Rightarrow \text{None})$
 $\quad \mid T\text{-f64} \Rightarrow (\text{case } (cvt\text{-f64 } sx \ v) \text{ of } \text{Some } c \Rightarrow \text{Some } (\text{ConstFloat64 } c) \mid \text{None} \Rightarrow \text{None}))$

definition $bits :: v \Rightarrow bytes$ **where**

$bits \ v = (\text{case } v \text{ of}$
 $\quad \text{ConstInt32} - c \Rightarrow (\text{serialise-i32 } c)$
 $\quad \mid \text{ConstInt64} - c \Rightarrow (\text{serialise-i64 } c)$
 $\quad \mid \text{ConstFloat32 } c \Rightarrow (\text{serialise-f32 } c)$
 $\quad \mid \text{ConstFloat64 } c \Rightarrow (\text{serialise-f64 } c))$

definition $bitzero :: t \Rightarrow v$ **where**

$bitzero \ t = (\text{case } t \text{ of}$
 $\quad (T\text{-i32 } sec) \Rightarrow \text{ConstInt32 } sec \ 0$
 $\quad \mid (T\text{-i64 } sec) \Rightarrow \text{ConstInt64 } sec \ 0$
 $\quad \mid T\text{-f32} \Rightarrow \text{ConstFloat32 } 0$
 $\quad \mid T\text{-f64} \Rightarrow \text{ConstFloat64 } 0)$

definition $n\text{-zeros} :: t \text{ list} \Rightarrow v \text{ list}$ **where**

$n\text{-zeros } ts = (\text{map } (\lambda t. \text{bitzero } t) \ ts)$

lemma $is\text{-int-}t\text{-exists}$:

assumes $is\text{-int-}t \ t$

shows $\exists sec. t = (T\text{-i32 } sec) \vee t = (T\text{-i64 } sec)$

$\langle proof \rangle$

lemma $is\text{-float-}t\text{-exists}$:

assumes $is\text{-float-}t \ t$

shows $\exists sec. t = T\text{-f32} \vee t = T\text{-f64}$

$\langle proof \rangle$

lemma $int\text{-float-disjoint}$: $is\text{-int-}t \ t = \neg(is\text{-float-}t \ t)$

$\langle proof \rangle$

lemma *types-agree-imp-types-agree-insecure*:

assumes *types-agree* $t\ v$

shows *types-agree-insecure* $t\ v$

$\langle proof \rangle$

lemma *stab-unfold*:

assumes *stab* $s\ i\ j = \text{Some } cl$

shows $\exists k. \text{inst.tab } ((\text{inst } s)!i) = \text{Some } k \wedge \text{length } ((\text{tab } s)!k) > j \wedge ((\text{tab } s)!k)!j = \text{Some } cl$

$\langle proof \rangle$

lemma *inj-basic*: *inj Basic*

$\langle proof \rangle$

lemma *inj-basic-econst*: *inj* $(\lambda v. \$C\ v)$

$\langle proof \rangle$

lemma *to-e-list-1*: $[\$ a] = \$* [a]$

$\langle proof \rangle$

lemma *to-e-list-2*: $[\$ a, \$ b] = \$* [a, b]$

$\langle proof \rangle$

lemma *to-e-list-3*: $[\$ a, \$ b, \$ c] = \$* [a, b, c]$

$\langle proof \rangle$

lemma *v-exists-b-e*: $\exists \text{ves}. (\$ \$* \text{ves}) = (\$* \text{ves})$

$\langle proof \rangle$

lemma *Lfilled-exact-imp-Lfilled*:

assumes *Lfilled-exact* $n\ \text{lholed}\ es\ LI$

shows *Lfilled* $n\ \text{lholed}\ es\ LI$

$\langle proof \rangle$

lemma *Lfilled-exact-app-imp-exists-Lfilled*:

assumes *const-list* ves

Lfilled-exact $n\ \text{lholed}\ (\text{ves}@es)\ LI$

shows $\exists \text{lholed}'. \text{Lfilled } n\ \text{lholed}'\ es\ LI$

$\langle proof \rangle$

lemma *Lfilled-imp-exists-Lfilled-exact*:

assumes *Lfilled* $n\ \text{lholed}\ es\ LI$

shows $\exists \text{lholed}'\ \text{ves}\ es\text{-c}. \text{const-list } \text{ves} \wedge \text{Lfilled-exact } n\ \text{lholed}'\ (\text{ves}@es@es\text{-c})\ LI$

$\langle proof \rangle$

lemma *n-zeros-typeof*:

n-zeros $ts = vs \implies (ts = \text{map } \text{typeof } vs)$

$\langle proof \rangle$

end
theory *Wasm* **imports** *Wasm-Base-Defs* **begin**

inductive *b-e-typing* :: [*t-context*, *b-e list*, *tf*] \Rightarrow *bool* ($- \vdash - : - 60$) **where**

- *num ops*
 - $\text{const} : \mathcal{C} \vdash [C \ v] : ([\] \rightarrow [(typeof \ v)])$
 - $| \text{unop-i:is-int-t } t \Rightarrow \mathcal{C} \vdash [Unop-i \ t \ -] : ([t] \rightarrow [t])$
 - $| \text{unop-f:is-float-t } t \Rightarrow \mathcal{C} \vdash [Unop-f \ t \ -] : ([t] \rightarrow [t])$
 - $| \text{binop-i:}[\text{is-int-t } t; (\text{is-secret-t } t \rightarrow \text{safe-binop-i } iop)] \Rightarrow \mathcal{C} \vdash [Binop-i \ t \ iop] : ([t, t] \rightarrow [t])$
 - $| \text{binop-f:is-float-t } t \Rightarrow \mathcal{C} \vdash [Binop-f \ t \ -] : ([t, t] \rightarrow [t])$
 - $| \text{testop:is-int-t } t \Rightarrow \mathcal{C} \vdash [Testop \ t \ -] : ([t] \rightarrow [(T-i32 \ (t\text{-sec } t))])$
 - $| \text{relop-i:is-int-t } t \Rightarrow \mathcal{C} \vdash [Relop-i \ t \ -] : ([t, t] \rightarrow [(T-i32 \ (t\text{-sec } t))])$
 - $| \text{relop-f:is-float-t } t \Rightarrow \mathcal{C} \vdash [Relop-f \ t \ -] : ([t, t] \rightarrow [(T-i32 \ (t\text{-sec } t))])$
- *convert*
 - $| \text{convert:}[(t1 \neq t2); t\text{-sec } t1 = t\text{-sec } t2; (sx = None) = ((\text{is-float-t } t1 \wedge \text{is-float-t } t2) \vee (\text{is-int-t } t1 \wedge \text{is-int-t } t2 \wedge (t\text{-length } t1 < t\text{-length } t2)))] \Rightarrow \mathcal{C} \vdash [Cvtop \ t1 \ \text{Convert } t2 \ sx] : ([t2] \rightarrow [t1])$
- *reinterpret*
 - $| \text{reinterpret:}[(t1 \neq t2); t\text{-sec } t1 = t\text{-sec } t2; t\text{-length } t1 = t\text{-length } t2] \Rightarrow \mathcal{C} \vdash [Cvtop \ t1 \ \text{Reinterpret } t2 \ None] : ([t2] \rightarrow [t1])$
- *classify*
 - $| \text{classify:}[\text{is-int-t } t2; \text{is-public-t } t2; \text{classify-t } t2 = t1] \Rightarrow \mathcal{C} \vdash [Cvtop \ t1 \ \text{Classify } t2 \ None] : ([t2] \rightarrow [t1])$
- *declassify*
 - $| \text{declassify:}[(\text{trust-t } C) = \text{Trusted}; \text{is-int-t } t2; \text{is-secret-t } t2; \text{declassify-t } t2 = t1] \Rightarrow \mathcal{C} \vdash [Cvtop \ t1 \ \text{Declassify } t2 \ None] : ([t2] \rightarrow [t1])$
- *unreachable, nop, drop, select*
 - $| \text{unreachable} : \mathcal{C} \vdash [Unreachable] : (ts \rightarrow ts')$
 - $| \text{nop} : \mathcal{C} \vdash [Nop] : ([\] \rightarrow [\])$
 - $| \text{drop} : \mathcal{C} \vdash [Drop] : ([t] \rightarrow [\])$
 - $| \text{select:}[\text{sec} = \text{Secret} \rightarrow \text{is-secret-t } t] \Rightarrow \mathcal{C} \vdash [\text{Select } sec] : ([t, t, (T-i32 \ sec)] \rightarrow [t])$
- *block*
 - $| \text{block:}[\text{tf} = (tn \rightarrow tm); \mathcal{C}(\text{label} := ([tm] @ (\text{label } C))) \vdash es : (tn \rightarrow tm)] \Rightarrow \mathcal{C} \vdash [\text{Block } \text{tf } es] : (tn \rightarrow tm)$
- *loop*
 - $| \text{loop:}[\text{tf} = (tn \rightarrow tm); \mathcal{C}(\text{label} := ([tn] @ (\text{label } C))) \vdash es : (tn \rightarrow tm)] \Rightarrow \mathcal{C} \vdash [\text{Loop } \text{tf } es] : (tn \rightarrow tm)$
- *if then else*
 - $| \text{if-wasm:}[\text{tf} = (tn \rightarrow tm); \mathcal{C}(\text{label} := ([tm] @ (\text{label } C))) \vdash es1 : (tn \rightarrow tm); \mathcal{C}(\text{label} := ([tm] @ (\text{label } C))) \vdash es2 : (tn \rightarrow tm)] \Rightarrow \mathcal{C} \vdash [\text{If } \text{tf } es1 \ es2] : (tn @ [(T-i32 \ \text{Public})] \rightarrow tm)$
- *br*
 - $| \text{br:}[i < \text{length}(\text{label } C); (\text{label } C)!i = ts] \Rightarrow \mathcal{C} \vdash [\text{Br } i] : (t1s @ ts \rightarrow t2s)$
- *br-if*
 - $| \text{br-if:}[i < \text{length}(\text{label } C); (\text{label } C)!i = ts] \Rightarrow \mathcal{C} \vdash [\text{Br-if } i] : (ts @ [(T-i32 \ \text{Public})] \rightarrow ts)$

--- br-table
 $| \text{br-table} : \llbracket \text{list-all } (\lambda i. i < \text{length}(\text{label } C) \wedge (\text{label } C)!i = ts) (\text{is}@[i]) \rrbracket \implies C \vdash$
 $\llbracket \text{Br-table is } i \rrbracket : (t1s @ ts @ [(T\text{-i32 Public})] \text{-->} t2s)$
 --- return
 $| \text{return} : \llbracket (\text{return } C) = \text{Some } ts \rrbracket \implies C \vdash \llbracket \text{Return} \rrbracket : (t1s @ ts \text{-->} t2s)$
 --- call
 $| \text{call} : \llbracket \text{trust-compat } (\text{trust-t } C) \text{ tr}; i < \text{length}(\text{func-t } C); (\text{func-t } C)!i = (tr, tf) \rrbracket \implies$
 $C \vdash \llbracket \text{Call } i \rrbracket : tf$
 --- call-indirect
 $| \text{call-indirect} : \llbracket \text{trust-compat } (\text{trust-t } C) \text{ tr}; i < \text{length}(\text{types-t } C); (\text{types-t } C)!i =$
 $(tr, (t1s \text{-->} t2s)); (\text{table } C) \neq \text{None} \rrbracket \implies C \vdash \llbracket \text{Call-indirect } i \rrbracket : (t1s @ [(T\text{-i32}$
 $\text{Public})] \text{-->} t2s)$
 --- get-local
 $| \text{get-local} : \llbracket i < \text{length}(\text{local } C); (\text{local } C)!i = t \rrbracket \implies C \vdash \llbracket \text{Get-local } i \rrbracket : ([\] \text{-->} [t])$
 --- set-local
 $| \text{set-local} : \llbracket i < \text{length}(\text{local } C); (\text{local } C)!i = t \rrbracket \implies C \vdash \llbracket \text{Set-local } i \rrbracket : ([t] \text{-->} [\])$
 --- tee-local
 $| \text{tee-local} : \llbracket i < \text{length}(\text{local } C); (\text{local } C)!i = t \rrbracket \implies C \vdash \llbracket \text{Tee-local } i \rrbracket : ([t] \text{-->} [t])$
 --- get-global
 $| \text{get-global} : \llbracket i < \text{length}(\text{global } C); \text{tg-t } ((\text{global } C)!i) = t \rrbracket \implies C \vdash \llbracket \text{Get-global } i \rrbracket :$
 $([\] \text{-->} [t])$
 --- set-global
 $| \text{set-global} : \llbracket i < \text{length}(\text{global } C); \text{tg-t } ((\text{global } C)!i) = t; \text{is-mut } ((\text{global } C)!i) \rrbracket \implies$
 $C \vdash \llbracket \text{Set-global } i \rrbracket : ([t] \text{-->} [\])$
 --- load
 $| \text{load} : \llbracket (\text{memory } C) = \text{Some } (n, \text{sec}); t\text{-sec } t = \text{sec}; \text{load-store-t-bounds } a (\text{option-projl}$
 $\text{tp-sx } t) \rrbracket \implies C \vdash \llbracket \text{Load } t \text{ tp-sx } a \text{ off} \rrbracket : ([(T\text{-i32 Public})] \text{-->} [t])$
 --- store
 $| \text{store} : \llbracket (\text{memory } C) = \text{Some } (n, \text{sec}); t\text{-sec } t = \text{sec}; \text{load-store-t-bounds } a \text{ tp } t \rrbracket \implies$
 $C \vdash \llbracket \text{Store } t \text{ tp } a \text{ off} \rrbracket : ([(T\text{-i32 Public}), t] \text{-->} [\])$
 $\text{--- current-memory}$
 $| \text{current-memory} : (\text{memory } C) = \text{Some } (n, \text{sec}) \implies C \vdash \llbracket \text{Current-memory} \rrbracket : ([\] \text{-->}$
 $[(T\text{-i32 Public})])$
 --- Grow-memory
 $| \text{grow-memory} : (\text{memory } C) = \text{Some } (n, \text{sec}) \implies C \vdash \llbracket \text{Grow-memory} \rrbracket : ([(T\text{-i32}$
 $\text{Public})] \text{-->} [(T\text{-i32 Public})])$
 --- empty program
 $| \text{empty} : C \vdash [\] : ([\] \text{-->} [\])$
 --- composition
 $| \text{composition} : \llbracket C \vdash es : (t1s \text{-->} t2s); C \vdash [e] : (t2s \text{-->} t3s) \rrbracket \implies C \vdash es @ [e] : (t1s$
 $\text{-->} t3s)$
 --- weakening
 $| \text{weakening} : C \vdash es : (t1s \text{-->} t2s) \implies C \vdash es : (ts @ t1s \text{-->} ts @ t2s)$

inductive $\text{cl-typing} :: [s\text{-context}, cl, tf\text{-t}] \Rightarrow \text{bool}$ **where**

$\llbracket i < \text{length } (s\text{-inst } S); ((s\text{-inst } S)!i) = C; tf = (t1s \text{-->} t2s); C \llbracket \text{trust-t} := tr,$
 $\text{local} := (\text{local } C) @ t1s @ ts, \text{label} := ([t2s] @ (\text{label } C)), \text{return} := \text{Some } t2s \rrbracket \vdash$
 $es : ([\] \text{-->} t2s) \rrbracket \implies \text{cl-typing } S (\text{Func-native } i (tr, tf) ts es) (tr, (t1s \text{-->} t2s))$
 $| \text{cl-typing } S (\text{Func-host } tf h) tf$

inductive *e-typing* :: [s-context, t-context, e list, tf] \Rightarrow bool (--- \vdash - : - 60)
and *s-typing* :: [s-context, trust, (t list) option, nat, v list, e list, t list] \Rightarrow bool (--- \Vdash - -; - : - 60) **where**

$\mathcal{C} \vdash b\text{-es} : tf \implies \mathcal{S} \cdot \mathcal{C} \vdash \$*b\text{-es} : tf$

$\mid \llbracket \mathcal{S} \cdot \mathcal{C} \vdash es : (t1s \rightarrow t2s); \mathcal{S} \cdot \mathcal{C} \vdash [e] : (t2s \rightarrow t3s) \rrbracket \implies \mathcal{S} \cdot \mathcal{C} \vdash es @ [e] : (t1s \rightarrow t3s)$

$\mid \mathcal{S} \cdot \mathcal{C} \vdash es : (t1s \rightarrow t2s) \implies \mathcal{S} \cdot \mathcal{C} \vdash es : (ts @ t1s \rightarrow ts @ t2s)$

$\mid \mathcal{S} \cdot \mathcal{C} \vdash [Trap] : tf$

$\mid \llbracket \mathcal{S} \cdot (\text{trust-}t \ \mathcal{C}) \cdot \text{Some } ts \Vdash\text{-}i \text{ vs}; es : ts; \text{length } ts = n \rrbracket \implies \mathcal{S} \cdot \mathcal{C} \vdash [Local \ n \ i \ \text{vs} \ es] : (\llbracket \text{---} \rightarrow ts \rrbracket)$

$\mid \llbracket \text{trust-compat } (\text{trust-}t \ \mathcal{C}) \ tr; \text{cl-typing } \mathcal{S} \ cl \ (tr, tf) \rrbracket \implies \mathcal{S} \cdot \mathcal{C} \vdash [Callcl \ cl] : tf$

$\mid \llbracket \mathcal{S} \cdot \mathcal{C} \vdash e0s : (ts \rightarrow t2s); \mathcal{S} \cdot \mathcal{C} \llbracket label := ([ts] @ (label \ \mathcal{C})) \rrbracket \vdash es : (\llbracket \text{---} \rightarrow t2s \rrbracket); \text{length } ts = n \rrbracket \implies \mathcal{S} \cdot \mathcal{C} \vdash [Label \ n \ e0s \ es] : (\llbracket \text{---} \rightarrow t2s \rrbracket)$

$\mid \llbracket i < (\text{length } (s\text{-inst } \mathcal{S})); tvs = \text{map typeof } vs; \mathcal{C} = ((s\text{-inst } \mathcal{S})!i) \llbracket \text{trust-}t := tr, \text{local} := (\text{local } ((s\text{-inst } \mathcal{S})!i) @ tvs), \text{return} := rs \rrbracket; \mathcal{S} \cdot \mathcal{C} \vdash es : (\llbracket \text{---} \rightarrow ts \rrbracket); (rs = \text{Some } ts) \vee rs = \text{None} \rrbracket \implies \mathcal{S} \cdot tr \cdot rs \Vdash\text{-}i \text{ vs}; es : ts$

definition *globi-agree* $gs \ n \ g = (n < \text{length } gs \wedge gs!n = g)$

definition *memi-agree* $sm \ j \ m = ((\exists j' \ m'. j = \text{Some } j' \wedge j' < \text{length } sm \wedge m = \text{Some } m' \wedge sm!j' = m') \vee j = \text{None} \wedge m = \text{None})$

definition *funci-agree* $fs \ n \ f = (n < \text{length } fs \wedge fs!n = f)$

inductive *inst-typing* :: [s-context, inst, t-context] \Rightarrow bool **where**

$\llbracket \text{list-all2 } (\text{funci-agree } (s\text{-funcs } \mathcal{S})) \ fs \ tfs; \text{list-all2 } (\text{globi-agree } (s\text{-globs } \mathcal{S})) \ gs \ tgs; (i = \text{Some } i' \wedge i' < \text{length } (s\text{-tab } \mathcal{S}) \wedge (s\text{-tab } \mathcal{S})!i' = (\text{the } n)) \vee (i = \text{None} \wedge n = \text{None}); \text{memi-agree } (s\text{-mem } \mathcal{S}) \ j \ m \rrbracket \implies \text{inst-typing } \mathcal{S} \llbracket \text{types} = ts, \text{funcs} = fs, \text{tab} = i, \text{mem} = j, \text{globs} = gs \rrbracket \llbracket \text{trust-}t = tr, \text{types-}t = ts, \text{func-}t = tfs, \text{global} = tgs, \text{table} = n, \text{memory} = m, \text{local} = \llbracket \text{---} \rrbracket, \text{label} = \llbracket \text{---} \rrbracket, \text{return} = \text{None} \rrbracket$

definition *glob-agree* $g \ tg = (tg\text{-mut } tg = g\text{-mut } g \wedge tg\text{-}t \ tg = \text{typeof } (g\text{-val } g))$

definition *tab-agree* $\mathcal{S} \ tcl = (\text{case } tcl \text{ of } \text{None} \Rightarrow \text{True} \mid \text{Some } cl \Rightarrow \exists tf. \text{cl-typing } \mathcal{S} \ cl \ tf)$

definition *mem-agree* $bs \ m = (\lambda (bs, sec) (m, sec'). m \leq \text{mem-size } bs \wedge sec = sec') \ bs \ m$

inductive *store-typing* :: $[s, s\text{-context}] \Rightarrow \text{bool}$ **where**

$\llbracket \mathcal{S} = \langle s\text{-inst} = Cs, s\text{-funcs} = tfs, s\text{-tab} = ns, s\text{-mem} = ms, s\text{-globs} = tgs \rangle; \text{list-all2 } (inst\text{-typing } \mathcal{S}) \text{ insts } Cs; \text{list-all2 } (cl\text{-typing } \mathcal{S}) fs \ tfs; \text{list-all } (tab\text{-agree } \mathcal{S}) (\text{concat } tclss); \text{list-all2 } (\lambda tcls \ n. \ n \leq \text{length } tcls) \ tclss \ ns; \text{list-all2 } mem\text{-agree } bss \ ms; \text{list-all2 } glob\text{-agree } gs \ tgs \rrbracket \Longrightarrow \text{store-typing } \langle s\text{-inst} = insts, s\text{-funcs} = fs, s\text{-tab} = tclss, s\text{-mem} = bss, s\text{-globs} = gs \rangle \ \mathcal{S}$

inductive *config-typing* :: $[nat, s, v \text{ list}, e \text{ list}, (trust \times t \text{ list})] \Rightarrow \text{bool}$ ($\vdash' - - ; - : - 60$) **where**

$\llbracket \text{store-typing } s \ \mathcal{S}; \mathcal{S} \cdot tr \cdot None \Vdash\text{-}i \ vs; es : ts \rrbracket \Longrightarrow \vdash\text{-}i \ s; vs; es : (tr, ts)$

inductive *reduce-simple* :: $[e \text{ list}, action, e \text{ list}] \Rightarrow \text{bool}$ ($\langle - \rangle \rightsquigarrow \langle - \rangle 60$) **where**

— *integer unary ops*

$\text{unop-i32} : \langle \llbracket \$C \ (ConstInt32 \ sec' \ c), \$ (Unop\text{-}i \ (T\text{-}i32 \ sec) \ iop) \rrbracket \rangle \ (Unop\text{-}i32\text{-}action \ iop) \rightsquigarrow \langle \llbracket \$C \ (ConstInt32 \ sec \ (app\text{-}unop\text{-}i \ iop \ c)) \rrbracket \rangle$
 $\mid \text{unop-i64} : \langle \llbracket \$C \ (ConstInt64 \ sec' \ c), \$ (Unop\text{-}i \ (T\text{-}i64 \ sec) \ iop) \rrbracket \rangle \ (Unop\text{-}i64\text{-}action \ iop) \rightsquigarrow \langle \llbracket \$C \ (ConstInt64 \ sec \ (app\text{-}unop\text{-}i \ iop \ c)) \rrbracket \rangle$

— *float unary ops*

$\mid \text{unop-f32} : \langle \llbracket \$C \ (ConstFloat32 \ c), \$ (Unop\text{-}f \ T\text{-}f32 \ fop) \rrbracket \rangle \ (Unop\text{-}f32\text{-}action \ fop \ c) \rightsquigarrow \langle \llbracket \$C \ (ConstFloat32 \ (app\text{-}unop\text{-}f \ fop \ c)) \rrbracket \rangle$
 $\mid \text{unop-f64} : \langle \llbracket \$C \ (ConstFloat64 \ c), \$ (Unop\text{-}f \ T\text{-}f64 \ fop) \rrbracket \rangle \ (Unop\text{-}f64\text{-}action \ fop \ c) \rightsquigarrow \langle \llbracket \$C \ (ConstFloat64 \ (app\text{-}unop\text{-}f \ fop \ c)) \rrbracket \rangle$

— *int32 binary ops*

$\mid \text{binop-i32-Some} : \llbracket app\text{-}binop\text{-}i \ iop \ c1 \ c2 = (Some \ c) \rrbracket \Longrightarrow \langle \llbracket \$C \ (ConstInt32 \ sec' \ c1), \$C \ (ConstInt32 \ sec'' \ c2), \$ (Binop\text{-}i \ (T\text{-}i32 \ sec) \ iop) \rrbracket \rangle \ (Binop\text{-}i32\text{-}Some\text{-}action \ iop \ c1 \ c2) \rightsquigarrow \langle \llbracket \$C \ (ConstInt32 \ sec \ c) \rrbracket \rangle$
 $\mid \text{binop-i32-None} : \llbracket app\text{-}binop\text{-}i \ iop \ c1 \ c2 = None \rrbracket \Longrightarrow \langle \llbracket \$C \ (ConstInt32 \ sec' \ c1), \$C \ (ConstInt32 \ sec'' \ c2), \$ (Binop\text{-}i \ (T\text{-}i32 \ sec) \ iop) \rrbracket \rangle \ (Binop\text{-}i32\text{-}None\text{-}action \ iop \ c1 \ c2) \rightsquigarrow \langle \llbracket Trap \rrbracket \rangle$

— *int64 binary ops*

$\mid \text{binop-i64-Some} : \llbracket app\text{-}binop\text{-}i \ iop \ c1 \ c2 = (Some \ c) \rrbracket \Longrightarrow \langle \llbracket \$C \ (ConstInt64 \ sec' \ c1), \$C \ (ConstInt64 \ sec'' \ c2), \$ (Binop\text{-}i \ (T\text{-}i64 \ sec) \ iop) \rrbracket \rangle \ (Binop\text{-}i64\text{-}Some\text{-}action \ iop \ c1 \ c2) \rightsquigarrow \langle \llbracket \$C \ (ConstInt64 \ sec \ c) \rrbracket \rangle$
 $\mid \text{binop-i64-None} : \llbracket app\text{-}binop\text{-}i \ iop \ c1 \ c2 = None \rrbracket \Longrightarrow \langle \llbracket \$C \ (ConstInt64 \ sec' \ c1), \$C \ (ConstInt64 \ sec'' \ c2), \$ (Binop\text{-}i \ (T\text{-}i64 \ sec) \ iop) \rrbracket \rangle \ (Binop\text{-}i64\text{-}None\text{-}action \ iop \ c1 \ c2) \rightsquigarrow \langle \llbracket Trap \rrbracket \rangle$

— *float32 binary ops*

$\mid \text{binop-f32-Some} : \llbracket app\text{-}binop\text{-}f \ fop \ c1 \ c2 = (Some \ c) \rrbracket \Longrightarrow \langle \llbracket \$C \ (ConstFloat32 \ c1), \$C \ (ConstFloat32 \ c2), \$ (Binop\text{-}f \ T\text{-}f32 \ fop) \rrbracket \rangle \ (Binop\text{-}f32\text{-}Some\text{-}action \ fop \ c1 \ c2) \rightsquigarrow \langle \llbracket \$C \ (ConstFloat32 \ c) \rrbracket \rangle$
 $\mid \text{binop-f32-None} : \llbracket app\text{-}binop\text{-}f \ fop \ c1 \ c2 = None \rrbracket \Longrightarrow \langle \llbracket \$C \ (ConstFloat32 \ c1), \$C \ (ConstFloat32 \ c2), \$ (Binop\text{-}f \ T\text{-}f32 \ fop) \rrbracket \rangle \ (Binop\text{-}f32\text{-}None\text{-}action \ fop \ c1 \ c2) \rightsquigarrow \langle \llbracket Trap \rrbracket \rangle$

— *float64 binary ops*

$\mid \text{binop-f64-Some} : \llbracket app\text{-}binop\text{-}f \ fop \ c1 \ c2 = (Some \ c) \rrbracket \Longrightarrow \langle \llbracket \$C \ (ConstFloat64 \ c1), \$C \ (ConstFloat64 \ c2), \$ (Binop\text{-}f \ T\text{-}f64 \ fop) \rrbracket \rangle \ (Binop\text{-}f64\text{-}Some\text{-}action \ fop \ c1 \ c2) \rightsquigarrow$

$\llbracket \$C \text{ (ConstFloat64 } c) \rrbracket$
 $| \text{binop-f64-None:} \llbracket \text{app-binop-f fop } c1 \text{ } c2 = \text{None} \rrbracket \implies \llbracket \$C \text{ (ConstFloat64 } c1), \$C \text{ (ConstFloat64 } c2), \$(\text{Binop-f } T\text{-f64 fop}) \rrbracket \text{ (Binop-f64-None-action fop } c1 \text{ } c2) \rightsquigarrow \llbracket \text{Trap} \rrbracket$
 --- testops
 $| \text{testop-i32:} \llbracket \$C \text{ (ConstInt32 sec' } c), \$(\text{Testop (T-i32 sec) testop}) \rrbracket \text{ (Testop-i32-action testop) } \rightsquigarrow \llbracket \$C \text{ ConstInt32 sec (wasm-bool (app-testop-i testop } c)) \rrbracket$
 $| \text{testop-i64:} \llbracket \$C \text{ (ConstInt64 sec' } c), \$(\text{Testop (T-i64 sec) testop}) \rrbracket \text{ (Testop-i64-action testop) } \rightsquigarrow \llbracket \$C \text{ ConstInt32 sec (wasm-bool (app-testop-i testop } c)) \rrbracket$
 --- int relops
 $| \text{relop-i32:} \llbracket \$C \text{ (ConstInt32 sec' } c1), \$C \text{ (ConstInt32 sec'' } c2), \$(\text{Relop-i (T-i32 sec) iop}) \rrbracket \text{ (Relop-i32-action iop) } \rightsquigarrow \llbracket \$C \text{ (ConstInt32 sec (wasm-bool (app-relop-i iop } c1 \text{ } c2))) \rrbracket$
 $| \text{relop-i64:} \llbracket \$C \text{ (ConstInt64 sec' } c1), \$C \text{ (ConstInt64 sec'' } c2), \$(\text{Relop-i (T-i64 sec) iop}) \rrbracket \text{ (Relop-i64-action iop) } \rightsquigarrow \llbracket \$C \text{ (ConstInt32 sec (wasm-bool (app-relop-i iop } c1 \text{ } c2))) \rrbracket$
 --- float relops
 $| \text{relop-f32:} \llbracket \$C \text{ (ConstFloat32 } c1), \$C \text{ (ConstFloat32 } c2), \$(\text{Relop-f T-f32 fop}) \rrbracket \text{ (Relop-f32-action fop } c1 \text{ } c2) \rightsquigarrow \llbracket \$C \text{ (ConstInt32 Public (wasm-bool (app-relop-f fop } c1 \text{ } c2))) \rrbracket$
 $| \text{relop-f64:} \llbracket \$C \text{ (ConstFloat64 } c1), \$C \text{ (ConstFloat64 } c2), \$(\text{Relop-f T-f64 fop}) \rrbracket \text{ (Relop-f64-action fop } c1 \text{ } c2) \rightsquigarrow \llbracket \$C \text{ (ConstInt32 Public (wasm-bool (app-relop-f fop } c1 \text{ } c2))) \rrbracket$
 --- convert
 $| \text{convert-Some:} \llbracket \text{types-agree-insecure } t1 \text{ } v; \text{cvt } t2 \text{ sx } v = (\text{Some } v') \rrbracket \implies \llbracket \$C \text{ (} v), \$(\text{Cvtop } t2 \text{ Convert } t1 \text{ sx}) \rrbracket \text{ (Convert-Some-action } t1 \text{ } t2 \text{ } v) \rightsquigarrow \llbracket \$C \text{ (} v') \rrbracket$
 $| \text{convert-None:} \llbracket \text{types-agree-insecure } t1 \text{ } v; \text{cvt } t2 \text{ sx } v = \text{None} \rrbracket \implies \llbracket \$C \text{ (} v), \$(\text{Cvtop } t2 \text{ Convert } t1 \text{ sx}) \rrbracket \text{ (Convert-None-action } t1 \text{ } t2 \text{ } v) \rightsquigarrow \llbracket \text{Trap} \rrbracket$
 --- reinterpret
 $| \text{reinterpret:types-agree-insecure } t1 \text{ } v \implies \llbracket \$C \text{ (} v), \$(\text{Cvtop } t2 \text{ Reinterpret } t1 \text{ None}) \rrbracket \text{ (Reinterpret-action) } \rightsquigarrow \llbracket \$C \text{ (wasm-deserialise (bits } v) \text{ } t2)) \rrbracket$
 --- classify
 $| \text{classify:types-agree-insecure } t1 \text{ } v \implies \llbracket \$C \text{ (} v), \$(\text{Cvtop } t2 \text{ Classify } t1 \text{ None}) \rrbracket \text{ (Classify-action) } \rightsquigarrow \llbracket \$C \text{ (classify } v) \rrbracket$
 --- declassify
 $| \text{declassify:types-agree-insecure } t1 \text{ } v \implies \llbracket \$C \text{ (} v), \$(\text{Cvtop } t2 \text{ Declassify } t1 \text{ None}) \rrbracket \text{ (Declassify-action) } \rightsquigarrow \llbracket \$C \text{ (declassify } v) \rrbracket$
 --- unreachable
 $| \text{unreachable:} \llbracket \$ \text{ Unreachable} \rrbracket \text{ (Unreachable-action) } \rightsquigarrow \llbracket \text{Trap} \rrbracket$
 --- nop
 $| \text{nop:} \llbracket \$ \text{ Nop} \rrbracket \text{ (Nop-action) } \rightsquigarrow \llbracket \square \rrbracket$
 --- drop
 $| \text{drop:} \llbracket \$C \text{ (} v), \$ \text{ Drop} \rrbracket \text{ (Drop-action) } \rightsquigarrow \llbracket \square \rrbracket$
 --- select
 $| \text{select-false:int-eq } n \text{ } 0 \implies \llbracket \$C \text{ (} v1), \$C \text{ (} v2), \$C \text{ (ConstInt32 sec } n), \$ \text{ Select sec' } \rrbracket \text{ (Select-action sec' } n) \rightsquigarrow \llbracket \$C \text{ (} v2) \rrbracket$
 $| \text{select-true:int-ne } n \text{ } 0 \implies \llbracket \$C \text{ (} v1), \$C \text{ (} v2), \$C \text{ (ConstInt32 sec } n), \$ \text{ Select sec' } \rrbracket \text{ (Select-action sec' } n) \rightsquigarrow \llbracket \$C \text{ (} v1) \rrbracket$
 --- block

$| \text{block} : [\text{const-list } vs; \text{length } vs = n; \text{length } t1s = n; \text{length } t2s = m] \implies (\downarrow vs @$
 $[\$(\text{Block } (t1s \rightarrow t2s) \text{ es})]) (\text{Block-action}) \rightsquigarrow (\downarrow [\text{Label } m [] (vs @ (\$* \text{ es}))])$
 --- loop
 $| \text{loop} : [\text{const-list } vs; \text{length } vs = n; \text{length } t1s = n; \text{length } t2s = m] \implies (\downarrow vs @$
 $[\$(\text{Loop } (t1s \rightarrow t2s) \text{ es})]) (\text{Loop-action}) \rightsquigarrow (\downarrow [\text{Label } n [\$(\text{Loop } (t1s \rightarrow t2s) \text{ es})] (vs$
 $@ (\$* \text{ es}))])$
 --- if
 $| \text{if-false} : \text{int-eq } n \ 0 \implies (\downarrow [\$(C (\text{ConstInt32 } \text{sec } n), \$(\text{If } \text{tf } e1s \ e2s))]) (\text{If-false-action}$
 $n) \rightsquigarrow (\downarrow [\$(\text{Block } \text{tf } e2s)])$
 $| \text{if-true} : \text{int-ne } n \ 0 \implies (\downarrow [\$(C (\text{ConstInt32 } \text{sec } n), \$(\text{If } \text{tf } e1s \ e2s))]) (\text{If-true-action}$
 $n) \rightsquigarrow (\downarrow [\$(\text{Block } \text{tf } e1s)])$
 --- label
 $| \text{label-const} : \text{const-list } vs \implies (\downarrow [\text{Label } n \text{ es } vs]) (\text{Label-const-action}) \rightsquigarrow (\downarrow vs)$
 $| \text{label-trap} : (\downarrow [\text{Label } n \text{ es } [\text{Trap}]]]) (\text{Label-trap-action}) \rightsquigarrow (\downarrow [\text{Trap}])$
 --- br
 $| \text{br} : [\text{const-list } vs; \text{length } vs = n; L\text{filled } i \text{ lholed } (vs @ [\$(\text{Br } i)]) \text{ LI}] \implies (\downarrow [\text{Label}$
 $n \text{ es } \text{LI}]) (\text{Br-action}) \rightsquigarrow (\downarrow vs @ \text{es})$
 --- br-if
 $| \text{br-if-false} : \text{int-eq } n \ 0 \implies (\downarrow [\$(C (\text{ConstInt32 } \text{sec } n), \$(\text{Br-if } i))]) (\text{Br-if-false-action}$
 $n) \rightsquigarrow (\downarrow [])$
 $| \text{br-if-true} : \text{int-ne } n \ 0 \implies (\downarrow [\$(C (\text{ConstInt32 } \text{sec } n), \$(\text{Br-if } i))]) (\text{Br-if-true-action}$
 $n) \rightsquigarrow (\downarrow [\$(\text{Br } i)])$
 --- br-table
 $| \text{br-table} : [\text{length } is > (\text{nat-of-int } c)] \implies (\downarrow [\$(C (\text{ConstInt32 } \text{sec } c), \$(\text{Br-table } is$
 $i))]) (\text{Br-table-action } c) \rightsquigarrow (\downarrow [\$(\text{Br } (is! (\text{nat-of-int } c)))]])$
 $| \text{br-table-length} : [\text{length } is \leq (\text{nat-of-int } c)] \implies (\downarrow [\$(C (\text{ConstInt32 } \text{sec } c), \$(\text{Br-table}$
 $is \ i))]) (\text{Br-table-length-action } c) \rightsquigarrow (\downarrow [\$(\text{Br } i)])$
 --- local
 $| \text{local-const} : [\text{const-list } es; \text{length } es = n] \implies (\downarrow [\text{Local } n \ i \ vs \ es]) (\text{Local-const-action}) \rightsquigarrow$
 $(\downarrow es)$
 $| \text{local-trap} : (\downarrow [\text{Local } n \ i \ vs \ [\text{Trap}]]]) (\text{Local-trap-action}) \rightsquigarrow (\downarrow [\text{Trap}])$
 --- return
 $| \text{return} : [\text{const-list } vs; \text{length } vs = n; L\text{filled } j \text{ lholed } (vs @ [\$(\text{Return})]) \text{ es}] \implies$
 $(\downarrow [\text{Local } n \ i \ vls \ es]) (\text{Return-action}) \rightsquigarrow (\downarrow vs)$
 --- tee-local
 $| \text{tee-local} : \text{is-const } v \implies (\downarrow [v, \$(\text{Tee-local } i)]) (\text{Tee-local-action}) \rightsquigarrow (\downarrow [v, v, \$(\text{Set-local}$
 $i)])$
 $| \text{trap} : [es \neq [\text{Trap}]; L\text{filled } 0 \text{ lholed } [\text{Trap}] \text{ es}] \implies (\downarrow es) (\text{Trap-action}) \rightsquigarrow (\downarrow [\text{Trap}])$

inductive *reduce* :: $[s, v \text{ list}, e \text{ list}, \text{action}, \text{nat}, s, v \text{ list}, e \text{ list}] \Rightarrow \text{bool } ((\downarrow -; -; -))$
 $\rightsquigarrow' - (\downarrow -; -; -) \ 60$ **where**
 $\text{--- lifting basic reduction}$
 $\text{basic} : (\downarrow e) \ a \rightsquigarrow (\downarrow e') \implies (\downarrow s; vs; e) \ a \rightsquigarrow\text{-i } (\downarrow s; vs; e')$
 --- call
 $| \text{call} : (\downarrow s; vs; [\$(\text{Call } j)]) (\text{Call-action}) \rightsquigarrow\text{-i } (\downarrow s; vs; [\text{Callcl } (\text{sfunc } s \ i \ j)])$
 --- call-indirect
 $| \text{call-indirect-Some} : [\text{stab } s \ i \ (\text{nat-of-int } c) = \text{Some } cl; \text{stypes } s \ i \ j = \text{tf}; \text{cl-type } cl =$
 $\text{tf}] \implies (\downarrow s; vs; [\$(C (\text{ConstInt32 } \text{sec } c), \$(\text{Call-indirect } j))]) (\text{Call-indirect-Some-action})$

$c) \rightsquigarrow -i \langle s; vs; [Callcl\ cl] \rangle$
 $| \text{call-indirect-None}: \llbracket (stab\ s\ i\ (nat\text{-}of\text{-}int\ c) = Some\ cl \wedge stypes\ s\ i\ j \neq cl\text{-}type\ cl) \vee stab\ s\ i\ (nat\text{-}of\text{-}int\ c) = None \rrbracket \implies \langle s; vs; [\$C\ (ConstInt32\ sec\ c), \$ (Call\text{-}indirect\ j)] \rangle \ (Call\text{-}indirect\text{-}None\text{-}action\ c) \rightsquigarrow -i \langle s; vs; [Trap] \rangle$
 --- call
 $| \text{callcl-native}: \llbracket cl = Func\text{-}native\ j\ (tr, (t1s \rightarrow t2s))\ ts\ es; ves = (\$ \$* vcs); length\ vcs = n; length\ ts = k; length\ t1s = n; length\ t2s = m; (n\text{-}zeros\ ts = zs) \rrbracket \implies \langle s; vs; ves @ [Callcl\ cl] \rangle \ (Callcl\text{-}native\text{-}action\ n) \rightsquigarrow -i \langle s; vs; [Local\ m\ j\ (vcs @ zs)\ \$ (Block\ ([] \rightarrow t2s)\ es)] \rangle$
 $| \text{callcl-host-Some}: \llbracket cl = Func\text{-}host\ (tr, (t1s \rightarrow t2s))\ f; ves = (\$ \$* vcs); length\ vcs = n; length\ t1s = n; length\ t2s = m; host\text{-}apply\ s\ (t1s \rightarrow t2s)\ f\ vcs\ hs = Some\ (s', vcs') \rrbracket \implies \langle s; vs; ves @ [Callcl\ cl] \rangle \ (Callcl\text{-}host\text{-}Some\text{-}action\ s\ vcs\ s'\ vcs'\ tr\ (t1s \rightarrow t2s)\ f\ hs) \rightsquigarrow -i \langle s'; vs; (\$ \$* vcs') \rangle$
 $| \text{callcl-host-None}: \llbracket cl = Func\text{-}host\ (tr, (t1s \rightarrow t2s))\ f; ves = (\$ \$* vcs); length\ vcs = n; length\ t1s = n; length\ t2s = m \rrbracket \implies \langle s; vs; ves @ [Callcl\ cl] \rangle \ (Callcl\text{-}host\text{-}None\text{-}action\ s\ vcs\ tr\ (t1s \rightarrow t2s)\ f\ hs) \rightsquigarrow -i \langle s; vs; [Trap] \rangle$
 --- get-local
 $| \text{get-local}: \llbracket length\ vi = j \rrbracket \implies \langle s; (vi @ [v] @ vs); \$ (Get\text{-}local\ j) \rangle \ (Get\text{-}local\text{-}action) \rightsquigarrow -i \langle s; (vi @ [v] @ vs); \$ (C\ v) \rangle$
 --- set-local
 $| \text{set-local}: \llbracket length\ vi = j \rrbracket \implies \langle s; (vi @ [v] @ vs); \$ (C\ v'), \$ (Set\text{-}local\ j) \rangle \ (Set\text{-}local\text{-}action) \rightsquigarrow -i \langle s; (vi @ [v] @ vs); [] \rangle$
 --- get-global
 $| \text{get-global}: \langle s; vs; \$ (Get\text{-}global\ j) \rangle \ (Get\text{-}global\text{-}action) \rightsquigarrow -i \langle s; vs; \$ C\ (sglob\text{-}val\ s\ i\ j) \rangle$
 --- set-global
 $| \text{set-global}: \text{supdate-glob}\ s\ i\ j\ v = s' \implies \langle s; vs; \$ (C\ v), \$ (Set\text{-}global\ j) \rangle \ (Set\text{-}global\text{-}action) \rightsquigarrow -i \langle s'; vs; [] \rangle$
 --- load
 $| \text{load-Some}: \llbracket smem\text{-}ind\ s\ i = Some\ j; ((mem\ s)!j) = (m, sec); load\ m\ (nat\text{-}of\text{-}int\ k)\ off\ (t\text{-}length\ t) = Some\ bs \rrbracket \implies \langle s; vs; \$C\ (ConstInt32\ sec'\ k), \$ (Load\ t\ None\ a\ off) \rangle \ (Load\text{-}Some\text{-}action\ t\ (nat\text{-}of\text{-}int\ k)\ a\ off) \rightsquigarrow -i \langle s; vs; \$C\ (wasm\text{-}deserialise\ bs\ t) \rangle$
 $| \text{load-None}: \llbracket smem\text{-}ind\ s\ i = Some\ j; ((mem\ s)!j) = (m, sec); load\ m\ (nat\text{-}of\text{-}int\ k)\ off\ (t\text{-}length\ t) = None \rrbracket \implies \langle s; vs; \$C\ (ConstInt32\ sec'\ k), \$ (Load\ t\ None\ a\ off) \rangle \ (Load\text{-}None\text{-}action\ t\ (nat\text{-}of\text{-}int\ k)\ a\ off) \rightsquigarrow -i \langle s; vs; [Trap] \rangle$
 --- load packed
 $| \text{load-packed-Some}: \llbracket smem\text{-}ind\ s\ i = Some\ j; ((mem\ s)!j) = (m, sec); load\text{-}packed\ sx\ m\ (nat\text{-}of\text{-}int\ k)\ off\ (tp\text{-}length\ tp)\ (t\text{-}length\ t) = Some\ bs \rrbracket \implies \langle s; vs; \$C\ (ConstInt32\ sec'\ k), \$ (Load\ t\ (Some\ (tp, sx))\ a\ off) \rangle \ (Load\text{-}packed\text{-}Some\text{-}action\ tp\ sx\ (nat\text{-}of\text{-}int\ k)\ a\ off) \rightsquigarrow -i \langle s; vs; \$C\ (wasm\text{-}deserialise\ bs\ t) \rangle$
 $| \text{load-packed-None}: \llbracket smem\text{-}ind\ s\ i = Some\ j; ((mem\ s)!j) = (m, sec); load\text{-}packed\ sx\ m\ (nat\text{-}of\text{-}int\ k)\ off\ (tp\text{-}length\ tp)\ (t\text{-}length\ t) = None \rrbracket \implies \langle s; vs; \$C\ (ConstInt32\ sec'\ k), \$ (Load\ t\ (Some\ (tp, sx))\ a\ off) \rangle \ (Load\text{-}packed\text{-}None\text{-}action\ tp\ sx\ (nat\text{-}of\text{-}int\ k)\ a\ off) \rightsquigarrow -i \langle s; vs; [Trap] \rangle$
 --- store
 $| \text{store-Some}: \llbracket types\text{-}agree\text{-}insecure\ t\ v; smem\text{-}ind\ s\ i = Some\ j; ((mem\ s)!j) = (m, sec); store\ m\ (nat\text{-}of\text{-}int\ k)\ off\ (bits\ v)\ (t\text{-}length\ t) = Some\ mem \rrbracket \implies \langle s; vs; \$C\ (ConstInt32\ sec'\ k), \$ C\ v, \$ (Store\ t\ None\ a\ off) \rangle \ (Store\text{-}Some\text{-}action\ t\ (nat\text{-}of\text{-}int\ k)\ a\ off) \rightsquigarrow -i \langle s; vs; [Trap] \rangle$

$k) \ a \ off) \rightsquigarrow\!\!\sim\! -i \ (\downarrow s \ (\text{mem} := ((\text{mem } s)[j] := (\text{mem}', \text{sec})))) \downarrow vs; [])$
 $\mid \text{store-None} : \llbracket \text{types-agree-insecure } t \ v; \text{smem-ind } s \ i = \text{Some } j; ((\text{mem } s)!j) = (m, \text{sec}); \text{store } m \ (\text{nat-of-int } k) \ \text{off} \ (\text{bits } v) \ (\text{t-length } t) = \text{None} \rrbracket \implies (\downarrow s; vs; [\$C \ (\text{ConstInt32 } \text{sec}' \ k), \$C \ v, \$(\text{Store } t \ \text{None } a \ \text{off})]) \ (\text{Store-None-action } t \ (\text{nat-of-int } k) \ a \ \text{off}) \rightsquigarrow\!\!\sim\! -i \ (\downarrow s; vs; [\text{Trap}])$
 $\quad\text{--- store packed}$
 $\mid \text{store-packed-Some} : \llbracket \text{types-agree-insecure } t \ v; \text{smem-ind } s \ i = \text{Some } j; ((\text{mem } s)!j) = (m, \text{sec}); \text{store-packed } m \ (\text{nat-of-int } k) \ \text{off} \ (\text{bits } v) \ (\text{tp-length } tp) = \text{Some } \text{mem}' \rrbracket \implies (\downarrow s; vs; [\$C \ (\text{ConstInt32 } \text{sec}' \ k), \$C \ v, \$(\text{Store } t \ (\text{Some } tp) \ a \ \text{off})]) \ (\text{Store-packed-Some-action } t \ tp \ (\text{nat-of-int } k) \ a \ \text{off}) \rightsquigarrow\!\!\sim\! -i \ (\downarrow s \ (\text{mem} := ((\text{mem } s)[j] := (\text{mem}', \text{sec})))) \downarrow vs; [])$
 $\mid \text{store-packed-None} : \llbracket \text{types-agree-insecure } t \ v; \text{smem-ind } s \ i = \text{Some } j; ((\text{mem } s)!j) = (m, \text{sec}); \text{store-packed } m \ (\text{nat-of-int } k) \ \text{off} \ (\text{bits } v) \ (\text{tp-length } tp) = \text{None} \rrbracket \implies (\downarrow s; vs; [\$C \ (\text{ConstInt32 } \text{sec}' \ k), \$C \ v, \$(\text{Store } t \ (\text{Some } tp) \ a \ \text{off})]) \ (\text{Store-packed-None-action } t \ tp \ (\text{nat-of-int } k) \ a \ \text{off}) \rightsquigarrow\!\!\sim\! -i \ (\downarrow s; vs; [\text{Trap}])$
 $\quad\text{--- current-memory}$
 $\mid \text{current-memory} : \llbracket \text{smem-ind } s \ i = \text{Some } j; ((\text{mem } s)!j) = (m, \text{sec}); \text{mem-size } m = n \rrbracket \implies (\downarrow s; vs; [\$(\text{Current-memory})]) \ (\text{Current-memory-action } n) \rightsquigarrow\!\!\sim\! -i \ (\downarrow s; vs; [\$C \ (\text{ConstInt32 } \text{Public } (\text{int-of-nat } n))])$
 $\quad\text{--- grow-memory}$
 $\mid \text{grow-memory} : \llbracket \text{smem-ind } s \ i = \text{Some } j; ((\text{mem } s)!j) = (m, \text{sec}); \text{mem-size } m = n; \text{mem-grow } m \ (\text{nat-of-int } c) = \text{mem}' \rrbracket \implies (\downarrow s; vs; [\$C \ (\text{ConstInt32 } \text{sec}' \ c), \$(\text{Grow-memory})]) \ (\text{Grow-memory-Some-action } n \ (\text{nat-of-int } c)) \rightsquigarrow\!\!\sim\! -i \ (\downarrow s \ (\text{mem} := ((\text{mem } s)[j] := (\text{mem}', \text{sec})))) \downarrow vs; [\$C \ (\text{ConstInt32 } \text{Public } (\text{int-of-nat } n))])$
 $\quad\text{--- grow-memory fail}$
 $\mid \text{grow-memory-fail} : \llbracket \text{smem-ind } s \ i = \text{Some } j; ((\text{mem } s)!j) = (m, \text{sec}); \text{mem-size } m = n \rrbracket \implies (\downarrow s; vs; [\$C \ (\text{ConstInt32 } \text{sec}' \ c), \$(\text{Grow-memory})]) \ (\text{Grow-memory-None-action } n \ (\text{nat-of-int } c)) \rightsquigarrow\!\!\sim\! -i \ (\downarrow s; vs; [\$C \ (\text{ConstInt32 } \text{Public } \text{int32-minus-one})])$
 $\quad\text{--- inductive label reduction}$
 $\mid \text{label} : \llbracket (\downarrow s; vs; \text{es}) \rightsquigarrow\!\!\sim\! -i \ (\downarrow s'; vs'; \text{es}') \rrbracket; L\text{filled } k \ \text{lholed } \text{es } \text{les}; L\text{filled } k \ \text{lholed } \text{es}' \ \text{les}' \rrbracket \implies (\downarrow s; vs; \text{les}) \rightsquigarrow\!\!\sim\! -i \ (\downarrow s'; vs'; \text{les}')$
 $\quad\text{--- inductive local reduction}$
 $\mid \text{local} : \llbracket (\downarrow s; vs; \text{es}) \rightsquigarrow\!\!\sim\! -i \ (\downarrow s'; vs'; \text{es}') \rrbracket \implies (\downarrow s; v0s; [\text{Local } n \ i \ \text{vs } \text{es}]) \rightsquigarrow\!\!\sim\! -j \ (\downarrow s'; v0s; [\text{Local } n \ i \ \text{vs}' \ \text{es}'])$

end

4 Host Properties

theory *Wasm-Axioms* **imports** *Wasm* **begin**

lemma *mem-grow-size*:

assumes *mem-grow* $m \ n = m'$

shows $(\text{mem-size } m + (64000 * n)) = \text{mem-size } m'$

<proof>

lemma *load-size*:

(load m n off l = None) = (mem-size m < (off + n + l))
 ⟨proof⟩

lemma load-packed-size:
 (load-packed sx m n off lp l = None) = (mem-size m < (off + n + lp))
 ⟨proof⟩

lemma store-size1:
 (store m n off v l = None) = (mem-size m < (off + n + l))
 ⟨proof⟩

lemma store-size:
 assumes (store m n off v l = Some m')
 shows mem-size m = mem-size m'
 ⟨proof⟩

lemma store-packed-size1:
 (store-packed m n off v l = None) = (mem-size m < (off + n + l))
 ⟨proof⟩

lemma store-packed-size:
 assumes (store-packed m n off v l = Some m')
 shows mem-size m = mem-size m'
 ⟨proof⟩

axiomatization where
 wasm-deserialise-type:typeof (wasm-deserialise bs t) = t

axiomatization where
 host-apply-preserve-store: list-all2 types-agree t1s vs \implies host-apply s (t1s -> t2s) f vs hs = Some (s', vs') \implies store-extension s s'
and host-apply-respect-type: list-all2 types-agree t1s vs \implies host-apply s (t1s -> t2s) f vs hs = Some (s', vs') \implies list-all2 types-agree t2s vs'
and host-trust-security-Some: store-public-agree s s' \implies publics-agree vs vs' \implies host-apply s (t1s -> t2s) f vs hs = Some (s-a, vs-a) \implies
 $\exists s'-a \text{ vs}'-a. \text{ host-apply } s' (t1s -> t2s) f \text{ vs}' \text{ hs}' = \text{Some}$
 $(s'-a, \text{vs}'-a) \wedge$
 $\text{store-public-agree } s-a \text{ s}'-a \wedge$
 $\text{publics-agree } \text{vs}-a \text{ vs}'-a$
and host-trust-security-None: store-public-agree s s' \implies publics-agree vs vs' \implies
 host-apply s (t1s -> t2s) f vs hs = None \implies
 host-apply s' (t1s -> t2s) f vs' hs' = None
end

5 Auxiliary Type System Properties

theory Wasm-Properties-Aux **imports** Wasm-Axioms **begin**

lemma is-float-public-t:

assumes *is-float-t* *t*
shows *is-public-t* *t*
<proof>

lemma *is-secret-int-t*:
assumes *is-secret-t* *t*
shows *is-int-t* *t*
<proof>

lemma *typeof-i32*:
assumes *typeof* *v* = (*T-i32* *sec*)
shows $\exists c. v = \text{ConstInt32 } \text{sec } c$
<proof>

lemma *typeof-i64*:
assumes *typeof* *v* = (*T-i64* *sec*)
shows $\exists c. v = \text{ConstInt64 } \text{sec } c$
<proof>

lemma *typeof-f32*:
assumes *typeof* *v* = *T-f32*
shows $\exists c. v = \text{ConstFloat32 } c$
<proof>

lemma *typeof-f64*:
assumes *typeof* *v* = *T-f64*
shows $\exists c. v = \text{ConstFloat64 } c$
<proof>

lemma *is-int-t-classify-t*:
assumes *is-int-t* *t*
 is-public-t *t*
shows *is-int-t* (*classify-t* *t*)
<proof>

lemma *classify-t-classify-typeof*:
assumes *types-agree-insecure* *t* *v*
shows (*classify-t* *t*) = *typeof* (*classify* *v*)
<proof>

lemma *declassify-t-declassify-typeof*:
assumes *types-agree-insecure* *t* *v*
shows (*declassify-t* *t*) = *typeof* (*declassify* *v*)
<proof>

lemma *exists-v-typeof*: $\exists v v. \text{typeof } v = t$
<proof>

lemma *lfilled-collapse1*:

assumes $Lfilled\ n\ lholed\ (vs@es)\ LI$
 $const\text{-}list\ vs$
 $length\ vs \geq l$
shows $\exists lholed'.\ Lfilled\ n\ lholed'\ ((drop\ (length\ vs - l)\ vs)@es)\ LI$
 $\langle proof \rangle$

lemma *lfilled-collapse2*:
assumes $Lfilled\ n\ lholed\ (es@es')\ LI$
shows $\exists lholed'\ vs'.\ Lfilled\ n\ lholed'\ es\ LI$
 $\langle proof \rangle$

lemma *lfilled-collapse3*:
assumes $Lfilled\ k\ lholed\ [Label\ n\ les\ es]\ LI$
shows $\exists lholed'.\ Lfilled\ (Suc\ k)\ lholed'\ es\ LI$
 $\langle proof \rangle$

lemma *unlift-b-e*: **assumes** $\mathcal{S} \cdot \mathcal{C} \vdash \$*b\text{-}es : tf$ **shows** $\mathcal{C} \vdash b\text{-}es : tf$
 $\langle proof \rangle$

lemma *store-typing-imp-inst-length-eq*:
assumes $store\text{-}typing\ s\ \mathcal{S}$
shows $length\ (inst\ s) = length\ (s\text{-}inst\ \mathcal{S})$
 $\langle proof \rangle$

lemma *store-typing-imp-func-length-eq*:
assumes $store\text{-}typing\ s\ \mathcal{S}$
shows $length\ (funcs\ s) = length\ (s\text{-}funcs\ \mathcal{S})$
 $\langle proof \rangle$

lemma *store-typing-imp-mem-length-eq*:
assumes $store\text{-}typing\ s\ \mathcal{S}$
shows $length\ (s.\text{mem}\ s) = length\ (s\text{-}mem\ \mathcal{S})$
 $\langle proof \rangle$

lemma *store-typing-imp-glob-length-eq*:
assumes $store\text{-}typing\ s\ \mathcal{S}$
shows $length\ (globs\ s) = length\ (s\text{-}globs\ \mathcal{S})$
 $\langle proof \rangle$

lemma *store-typing-imp-inst-typing*:
assumes $store\text{-}typing\ s\ \mathcal{S}$
 $i < length\ (inst\ s)$
shows $inst\text{-}typing\ \mathcal{S}\ ((inst\ s)!i)\ ((s\text{-}inst\ \mathcal{S})!i)$
 $\langle proof \rangle$

lemma *stab-typed-some-imp-member*:
assumes $stab\ s\ i\ c = Some\ cl$
 $store\text{-}typing\ s\ \mathcal{S}$

$i < \text{length } (\text{inst } s)$
shows $\text{Some } cl \in \text{set } (\text{concat } (s.\text{tab } s))$
 $\langle \text{proof} \rangle$

lemma *stab-typed-some-imp-cl-typed*:
assumes $\text{stab } s \ i \ c = \text{Some } cl$
 $\text{store-typing } s \ \mathcal{S}$
 $i < \text{length } (\text{inst } s)$
shows $\exists \text{tf. } \text{cl-typing } \mathcal{S} \ cl \ \text{tf}$
 $\langle \text{proof} \rangle$

lemma *b-e-type-empty1[dest]*: **assumes** $\mathcal{C} \vdash [] : (ts \rightarrow ts')$ **shows** $ts = ts'$
 $\langle \text{proof} \rangle$

lemma *b-e-type-empty*: $(\mathcal{C} \vdash [] : (ts \rightarrow ts')) = (ts = ts')$
 $\langle \text{proof} \rangle$

lemma *b-e-type-value*:
assumes $\mathcal{C} \vdash [e] : (ts \rightarrow ts')$
 $e = C \ v$
shows $ts' = ts \ @ \ [\text{typeof } v]$
 $\langle \text{proof} \rangle$

lemma *b-e-type-load*:
assumes $\mathcal{C} \vdash [e] : (ts \rightarrow ts')$
 $e = \text{Load } t \ \text{tp-sx} \ a \ \text{off}$
shows $\exists \text{ts'' } \text{sec } n. \ ts = \text{ts''} @ [(T\text{-i32 } \text{Public})] \wedge \text{ts}' = \text{ts''} @ [t] \wedge (\text{memory } \mathcal{C}) =$
 $\text{Some } (n, \text{sec}) \wedge t\text{-sec } t = \text{sec}$
 $\text{load-store-t-bounds } a \ (\text{option-projl } \text{tp-sx}) \ t$
 $\langle \text{proof} \rangle$

lemma *b-e-type-store*:
assumes $\mathcal{C} \vdash [e] : (ts \rightarrow ts')$
 $e = \text{Store } t \ \text{tp} \ a \ \text{off}$
shows $ts = \text{ts}' @ [(T\text{-i32 } \text{Public}), t]$
 $\exists \text{sec } n. (\text{memory } \mathcal{C}) = \text{Some } (n, \text{sec}) \wedge t\text{-sec } t = \text{sec}$
 $\text{load-store-t-bounds } a \ \text{tp} \ t$
 $\langle \text{proof} \rangle$

lemma *b-e-type-current-memory*:
assumes $\mathcal{C} \vdash [e] : (ts \rightarrow ts')$
 $e = \text{Current-memory}$
shows $\exists \text{sec } n. \ \text{ts}' = ts \ @ \ [(T\text{-i32 } \text{Public})] \wedge (\text{memory } \mathcal{C}) = \text{Some } (n, \text{sec})$
 $\langle \text{proof} \rangle$

lemma *b-e-type-grow-memory*:
assumes $\mathcal{C} \vdash [e] : (ts \rightarrow ts')$
 $e = \text{Grow-memory}$
shows $\exists \text{ts''}. \ ts = \text{ts''} @ [(T\text{-i32 } \text{Public})] \wedge ts = \text{ts}' \wedge (\exists n. (\text{memory } \mathcal{C}) = \text{Some } (n, \text{sec}))$

$n)$
 $\langle \text{proof} \rangle$

lemma *b-e-type-nop*:
assumes $\mathcal{C} \vdash [e] : (ts \rightarrow ts')$
 $e = \text{Nop}$
shows $ts = ts'$
 $\langle \text{proof} \rangle$

definition *arity-2-result* :: $b-e \Rightarrow t$ **where**
 $\text{arity-2-result } op2 = (\text{case } op2 \text{ of}$
 $\quad \text{Binop-}i \ t \ - \Rightarrow t$
 $\quad | \text{Binop-}f \ t \ - \Rightarrow t$
 $\quad | \text{Relop-}i \ t \ - \Rightarrow (T\text{-i32 } (t\text{-sec } t))$
 $\quad | \text{Relop-}f \ t \ - \Rightarrow (T\text{-i32 } (t\text{-sec } t)))$

lemma *b-e-type-binop-relop*:
assumes $\mathcal{C} \vdash [e] : (ts \rightarrow ts')$
 $e = \text{Binop-}i \ t \ iop \vee e = \text{Binop-}f \ t \ fop \vee e = \text{Relop-}i \ t \ irop \vee e = \text{Relop-}f \ t \ frop$
shows $\exists ts''. ts = ts''@[t, t] \wedge ts' = ts''@[arity\text{-}2\text{-result}(e)]$
 $e = \text{Binop-}i \ t \ iop \Longrightarrow is\text{-secret-}t \ t \Longrightarrow safe\text{-binop-}i \ iop$
 $e = \text{Binop-}f \ t \ fop \Longrightarrow is\text{-float-}t \ t$
 $e = \text{Relop-}f \ t \ frop \Longrightarrow is\text{-float-}t \ t$
 $\langle \text{proof} \rangle$

lemma *b-e-type-testop-drop-cvt0*:
assumes $\mathcal{C} \vdash [e] : (ts \rightarrow ts')$
 $e = \text{Testop } t \ testop \vee e = \text{Drop} \vee e = \text{Cvtop } t1 \ cvtop \ t2 \ sx$
shows $ts \neq []$
 $\langle \text{proof} \rangle$

definition *arity-1-result* :: $b-e \Rightarrow t$ **where**
 $\text{arity-1-result } op1 = (\text{case } op1 \text{ of}$
 $\quad \text{Unop-}i \ t \ - \Rightarrow t$
 $\quad | \text{Unop-}f \ t \ - \Rightarrow t$
 $\quad | \text{Testop } t \ - \Rightarrow (T\text{-i32 } (t\text{-sec } t))$
 $\quad | \text{Cvtop } t1 \ \text{Convert} \ - \Rightarrow t1$
 $\quad | \text{Cvtop } t1 \ \text{Reinterpret} \ - \Rightarrow t1$
 $\quad | \text{Cvtop} \ - \ \text{Classify } t2 \ - \Rightarrow \text{classify-}t \ t2$
 $\quad | \text{Cvtop} \ - \ \text{Declassify } t2 \ - \Rightarrow \text{declassify-}t \ t2)$

lemma *b-e-type-unop-testop*:
assumes $\mathcal{C} \vdash [e] : (ts \rightarrow ts')$
 $e = \text{Unop-}i \ t \ iop \vee e = \text{Unop-}f \ t \ fop \vee e = \text{Testop } t \ testop$
shows $\exists ts''. ts = ts''@[t] \wedge ts' = ts''@[arity\text{-}1\text{-result } e]$
 $e = \text{Unop-}f \ t \ fop \Longrightarrow is\text{-float-}t \ t$
 $\langle \text{proof} \rangle$

lemma *b-e-type-cvtop*:

assumes $\mathcal{C} \vdash [e] : (ts \rightarrow ts')$
 $e = \text{Cvtop } t1 \text{ cvtop } t \text{ } sx$
shows $\exists ts''. ts = ts''@[t] \wedge ts' = ts''@[arity-1\text{-result } e]$
 $cvtop = \text{Convert} \implies (t1 \neq t) \wedge t\text{-sec } t1 = t\text{-sec } t \wedge (sx = \text{None}) =$
 $((is\text{-float-}t \ t1 \wedge is\text{-float-}t \ t) \vee (is\text{-int-}t \ t1 \wedge is\text{-int-}t \ t \wedge (t\text{-length } t1 < t\text{-length } t)))$
 $cvtop = \text{Reinterpret} \implies (t1 \neq t) \wedge t\text{-sec } t1 = t\text{-sec } t \wedge t\text{-length } t1 = t\text{-length } t$
 t
 $cvtop = \text{Classify} \implies is\text{-int-}t \ t \wedge is\text{-public-}t \ t \wedge classify\text{-}t \ t = t1$
 $cvtop = \text{Declassify} \implies (trust\text{-}t \ C) = \text{Trusted} \wedge is\text{-int-}t \ t \wedge is\text{-secret-}t \ t \wedge$
 $declassify\text{-}t \ t = t1$
 $\langle proof \rangle$

lemma *b-e-type-drop*:

assumes $\mathcal{C} \vdash [e] : (ts \rightarrow ts')$
 $e = \text{Drop}$
shows $\exists t. ts = ts'@[t]$
 $\langle proof \rangle$

lemma *b-e-type-select*:

assumes $\mathcal{C} \vdash [e] : (ts \rightarrow ts')$
 $e = \text{Select } sec$
shows $\exists ts'' t. ts = ts''@[t, t, (T\text{-i32 } sec)] \wedge ts' = ts''@[t] \wedge (sec = \text{Secret} \longrightarrow$
 $is\text{-secret-}t \ t)$
 $\langle proof \rangle$

lemma *b-e-type-call*:

assumes $\mathcal{C} \vdash [e] : (ts \rightarrow ts')$
 $e = \text{Call } i$
shows $i < \text{length } (\text{func-}t \ C)$
 $\exists tr \ ts'' \ tf1 \ tf2. trust\text{-compat } (trust\text{-}t \ C) \ tr \wedge ts = ts''@[tf1] \wedge ts' = ts''@[tf2]$
 $\wedge (\text{func-}t \ C)!i = (tr, (tf1 \rightarrow tf2))$
 $\langle proof \rangle$

lemma *b-e-type-call-indirect*:

assumes $\mathcal{C} \vdash [e] : (ts \rightarrow ts')$
 $e = \text{Call-indirect } i$
shows $i < \text{length } (\text{types-}t \ C)$
 $\exists tr \ ts'' \ tf1 \ tf2. trust\text{-compat } (trust\text{-}t \ C) \ tr \wedge ts = ts''@[tf1]@[(T\text{-i32 } \text{Public})]$
 $\wedge ts' = ts''@[tf2] \wedge (\text{types-}t \ C)!i = (tr, (tf1 \rightarrow tf2))$
 $\langle proof \rangle$

lemma *b-e-type-get-local*:

assumes $\mathcal{C} \vdash [e] : (ts \rightarrow ts')$
 $e = \text{Get-local } i$
shows $\exists t. ts' = ts@[t] \wedge (\text{local } C)!i = t \wedge i < \text{length}(\text{local } C)$
 $\langle proof \rangle$

lemma *b-e-type-set-local*:

assumes $\mathcal{C} \vdash [e] : (ts \rightarrow ts')$
 $e = \text{Set-local } i$
shows $\exists t. ts = ts'@[t] \wedge (\text{local } \mathcal{C})!i = t \wedge i < \text{length}(\text{local } \mathcal{C})$
 $\langle \text{proof} \rangle$

lemma *b-e-type-tee-local*:
assumes $\mathcal{C} \vdash [e] : (ts \rightarrow ts')$
 $e = \text{Tee-local } i$
shows $\exists ts'' t. ts = ts''@[t] \wedge ts' = ts''@[t] \wedge (\text{local } \mathcal{C})!i = t \wedge i < \text{length}(\text{local } \mathcal{C})$
 $\langle \text{proof} \rangle$

lemma *b-e-type-get-global*:
assumes $\mathcal{C} \vdash [e] : (ts \rightarrow ts')$
 $e = \text{Get-global } i$
shows $\exists t. ts' = ts@[t] \wedge \text{tg-t}((\text{global } \mathcal{C})!i) = t \wedge i < \text{length}(\text{global } \mathcal{C})$
 $\langle \text{proof} \rangle$

lemma *b-e-type-set-global*:
assumes $\mathcal{C} \vdash [e] : (ts \rightarrow ts')$
 $e = \text{Set-global } i$
shows $\exists t. ts = ts'@[t] \wedge (\text{global } \mathcal{C})!i = \langle \text{tg-mut} = T\text{-mut}, \text{tg-t} = t \rangle \wedge i < \text{length}(\text{global } \mathcal{C})$
 $\langle \text{proof} \rangle$

lemma *b-e-type-block*:
assumes $\mathcal{C} \vdash [e] : (ts \rightarrow ts')$
 $e = \text{Block } tf \text{ es}$
shows $\exists ts'' tfn tfm. tf = (tfn \rightarrow tfm) \wedge (ts = ts''@[tfn] \wedge (ts' = ts''@[tfm] \wedge$
 $(\mathcal{C}(\text{label} := [tfm] @ \text{label } \mathcal{C}) \vdash \text{es} : tf))$
 $\langle \text{proof} \rangle$

lemma *b-e-type-loop*:
assumes $\mathcal{C} \vdash [e] : (ts \rightarrow ts')$
 $e = \text{Loop } tf \text{ es}$
shows $\exists ts'' tfn tfm. tf = (tfn \rightarrow tfm) \wedge (ts = ts''@[tfn] \wedge (ts' = ts''@[tfm] \wedge$
 $(\mathcal{C}(\text{label} := [tfn] @ \text{label } \mathcal{C}) \vdash \text{es} : tf))$
 $\langle \text{proof} \rangle$

lemma *b-e-type-if*:
assumes $\mathcal{C} \vdash [e] : (ts \rightarrow ts')$
 $e = \text{If } tf \text{ es1 es2}$
shows $\exists ts'' tfn tfm. tf = (tfn \rightarrow tfm) \wedge (ts = ts''@[tfn] @ [(T\text{-i32 } \text{Public})]) \wedge$
 $(ts' = ts''@[tfm]) \wedge$
 $(\mathcal{C}(\text{label} := [tfm] @ \text{label } \mathcal{C}) \vdash \text{es1} : tf) \wedge$
 $(\mathcal{C}(\text{label} := [tfm] @ \text{label } \mathcal{C}) \vdash \text{es2} : tf)$
 $\langle \text{proof} \rangle$

lemma *b-e-type-br*:
assumes $\mathcal{C} \vdash [e] : (ts \rightarrow ts')$

$e = Br\ i$
shows $i < length(label\ C)$
 $\exists ts-c\ ts''.\ ts = ts-c @ ts'' \wedge (label\ C)!i = ts''$
 $\langle proof \rangle$

lemma *b-e-type-br-if*:
assumes $C \vdash [e] : (ts \rightarrow ts')$
 $e = Br\text{-}if\ i$
shows $i < length(label\ C)$
 $\exists ts-c\ ts''.\ ts = ts-c @ ts'' @ [(T\text{-}i32\ Public)] \wedge ts' = ts-c @ ts'' \wedge (label\ C)!i = ts''$
 $\langle proof \rangle$

lemma *b-e-type-br-table*:
assumes $C \vdash [e] : (ts \rightarrow ts')$
 $e = Br\text{-}table\ is\ i$
shows $\exists ts-c\ ts''.\ list\text{-}all\ (\lambda i.\ i < length(label\ C) \wedge (label\ C)!i = ts'')\ (is @ [i]) \wedge$
 $ts = ts-c @ ts'' @ [(T\text{-}i32\ Public)]$
 $\langle proof \rangle$

lemma *b-e-type-return*:
assumes $C \vdash [e] : (ts \rightarrow ts')$
 $e = Return$
shows $\exists ts-c\ ts''.\ ts = ts-c @ ts'' \wedge (return\ C) = Some\ ts''$
 $\langle proof \rangle$

lemma *b-e-type-comp*:
assumes $C \vdash es @ [e] : (t1s \rightarrow t4s)$
shows $\exists ts'.\ (C \vdash es : (t1s \rightarrow ts')) \wedge (C \vdash [e] : (ts' \rightarrow t4s))$
 $\langle proof \rangle$

lemma *b-e-type-comp2-unlift*:
assumes $S \cdot C \vdash [\$e1, \$e2] : (t1s \rightarrow t2s)$
shows $\exists ts'.\ (C \vdash [e1] : (t1s \rightarrow ts')) \wedge (C \vdash [e2] : (ts' \rightarrow t2s))$
 $\langle proof \rangle$

lemma *b-e-type-comp2-relift*:
assumes $C \vdash [e1] : (t1s \rightarrow ts')\ C \vdash [e2] : (ts' \rightarrow t2s)$
shows $S \cdot C \vdash [\$e1, \$e2] : (ts @ t1s \rightarrow ts @ t2s)$
 $\langle proof \rangle$

lemma *b-e-type-value2*:
assumes $C \vdash [C\ v1, C\ v2] : (t1s \rightarrow t2s)$
shows $t2s = t1s @ [typeof\ v1, typeof\ v2]$
 $\langle proof \rangle$

lemma *e-type-comp*:

assumes $\mathcal{S}\cdot\mathcal{C} \vdash es@[e] : (t1s \rightarrow t3s)$
shows $\exists ts'. (\mathcal{S}\cdot\mathcal{C} \vdash es : (t1s \rightarrow ts')) \wedge (\mathcal{S}\cdot\mathcal{C} \vdash [e] : (ts' \rightarrow t3s))$
 $\langle proof \rangle$

lemma *e-type-comp-conc*:
assumes $\mathcal{S}\cdot\mathcal{C} \vdash es : (t1s \rightarrow t2s)$
 $\mathcal{S}\cdot\mathcal{C} \vdash es' : (t2s \rightarrow t3s)$
shows $\mathcal{S}\cdot\mathcal{C} \vdash es@es' : (t1s \rightarrow t3s)$
 $\langle proof \rangle$

lemma *b-e-type-comp-conc*:
assumes $\mathcal{C} \vdash es : (t1s \rightarrow t2s)$
 $\mathcal{C} \vdash es' : (t2s \rightarrow t3s)$
shows $\mathcal{C} \vdash es@es' : (t1s \rightarrow t3s)$
 $\langle proof \rangle$

lemma *e-type-comp-conc1*:
assumes $\mathcal{S}\cdot\mathcal{C} \vdash es@es' : (ts \rightarrow ts')$
shows $\exists ts''. (\mathcal{S}\cdot\mathcal{C} \vdash es : (ts \rightarrow ts'')) \wedge (\mathcal{S}\cdot\mathcal{C} \vdash es' : (ts'' \rightarrow ts'))$
 $\langle proof \rangle$

lemma *e-type-comp-conc2*:
assumes $\mathcal{S}\cdot\mathcal{C} \vdash es@es'@es'' : (t1s \rightarrow t2s)$
shows $\exists ts' ts''. (\mathcal{S}\cdot\mathcal{C} \vdash es : (t1s \rightarrow ts'))$
 $\wedge (\mathcal{S}\cdot\mathcal{C} \vdash es' : (ts' \rightarrow ts''))$
 $\wedge (\mathcal{S}\cdot\mathcal{C} \vdash es'' : (ts'' \rightarrow t2s))$
 $\langle proof \rangle$

lemma *b-e-type-value-list*:
assumes $(\mathcal{C} \vdash es@[C\ v] : (ts \rightarrow ts'@[t]))$
shows $(\mathcal{C} \vdash es : (ts \rightarrow ts'))$
 $(typeof\ v = t)$
 $\langle proof \rangle$

lemma *e-type-label*:
assumes $\mathcal{S}\cdot\mathcal{C} \vdash [Label\ n\ es0\ es] : (ts \rightarrow ts')$
shows $\exists t1s\ t2s. (ts' = (ts@t2s))$
 $\wedge length\ t1s = n$
 $\wedge (\mathcal{S}\cdot\mathcal{C} \vdash es0 : (t1s \rightarrow t2s))$
 $\wedge (\mathcal{S}\cdot\mathcal{C} \llbracket label := [t1s] @ (label\ \mathcal{C}) \rrbracket \vdash es : ([] \rightarrow t2s))$
 $\langle proof \rangle$

lemma *e-type-callcl-native*:
assumes $\mathcal{S}\cdot\mathcal{C} \vdash [Callcl\ cl] : (t1s' \rightarrow t2s')$
 $cl = Func-native\ i\ (tr,tf)\ ts\ es$
shows $\exists t1s\ t2s\ ts-c. trust-compat\ (trust-t\ \mathcal{C})\ tr$
 $\wedge (t1s' = ts-c @ t1s)$
 $\wedge (t2s' = ts-c @ t2s)$

$\wedge tf = (t1s \rightarrow t2s)$
 $\wedge i < \text{length } (s\text{-inst } \mathcal{S})$
 $\wedge (((s\text{-inst } \mathcal{S})!i) \langle trust\text{-}t := tr, local := (local ((s\text{-inst } \mathcal{S})!i)) \rangle @$
 $t1s @ ts, label := ([t2s] @ (label ((s\text{-inst } \mathcal{S})!i))), return := Some\ t2s \rangle \vdash es : ([$
 $\rightarrow t2s))$
 $\langle proof \rangle$

lemma *e-type-callcl-host*:
assumes $\mathcal{S} \cdot \mathcal{C} \vdash [Callcl\ cl] : (t1s' \rightarrow t2s')$
 $cl = Func\text{-}host\ tf\ f$
shows $\exists tr\ t1s\ t2s\ ts\text{-}c. trust\text{-}compat\ (trust\text{-}t\ \mathcal{C})\ tr$
 $\wedge (t1s' = ts\text{-}c @ t1s)$
 $\wedge (t2s' = ts\text{-}c @ t2s)$
 $\wedge tf = (tr, (t1s \rightarrow t2s))$
 $\langle proof \rangle$

lemma *e-type-callcl*:
assumes $\mathcal{S} \cdot \mathcal{C} \vdash [Callcl\ cl] : (t1s' \rightarrow t2s')$
shows $\exists tr\ t1s\ t2s\ ts\text{-}c. trust\text{-}compat\ (trust\text{-}t\ \mathcal{C})\ tr$
 $\wedge (t1s' = ts\text{-}c @ t1s)$
 $\wedge (t2s' = ts\text{-}c @ t2s)$
 $\wedge cl\text{-}type\ cl = (tr, (t1s \rightarrow t2s))$
 $\langle proof \rangle$

lemma *s-type-unfold*:
assumes $\mathcal{S} \cdot tr \cdot rs \Vdash\text{-}i\ vs; es : ts$
shows $i < \text{length } (s\text{-inst } \mathcal{S})$
 $(rs = Some\ ts) \vee rs = None$
 $(\mathcal{S} \cdot ((s\text{-inst } \mathcal{S})!i) \langle trust\text{-}t := tr, local := (local ((s\text{-inst } \mathcal{S})!i)) \rangle @ (map\ typeof$
 $vs), return := rs \rangle \vdash es : ([\rightarrow ts))$
 $\langle proof \rangle$

lemma *e-type-local*:
assumes $\mathcal{S} \cdot \mathcal{C} \vdash [Local\ n\ i\ vs\ es] : (ts \rightarrow ts')$
shows $\exists t1s. i < \text{length } (s\text{-inst } \mathcal{S})$
 $\wedge \text{length } t1s = n$
 $\wedge (\mathcal{S} \cdot ((s\text{-inst } \mathcal{S})!i) \langle trust\text{-}t := (trust\text{-}t\ \mathcal{C}), local := (local ((s\text{-inst } \mathcal{S})!i)) \rangle @$
 $(map\ typeof\ vs), return := Some\ t1s \rangle \vdash es : ([\rightarrow t1s))$
 $\wedge ts' = ts @ t1s$
 $\langle proof \rangle$

lemma *e-type-local-shallow*:
assumes $\mathcal{S} \cdot \mathcal{C} \vdash [Local\ n\ i\ vs\ es] : (ts \rightarrow ts')$
shows $\exists t1s. \text{length } t1s = n \wedge ts' = ts @ t1s \wedge (\mathcal{S} \cdot (trust\text{-}t\ \mathcal{C}) \cdot (Some\ t1s) \Vdash\text{-}i\ vs; es$
 $: t1s)$
 $\langle proof \rangle$

lemma *e-type-const-unwrap*:

assumes *is-const e*
shows $\exists v. e = \$C\ v$
 $\langle proof \rangle$

lemma *is-const-list1*:
assumes $ves = \text{map } (Basic \circ EConst) \ vs$
shows *const-list ves*
 $\langle proof \rangle$

lemma *is-const-list*:
assumes $ves = \$\$* \ vs$
shows *const-list ves*
 $\langle proof \rangle$

lemma *const-list-cons-last*:
assumes *const-list (es@[e])*
shows *const-list es*
is-const e
 $\langle proof \rangle$

lemma *e-type-const1*:
assumes *is-const e*
shows $\exists t. (\mathcal{S} \cdot \mathcal{C} \vdash [e] : (ts \rightarrow ts@[t]))$
 $\langle proof \rangle$

lemma *e-type-const*:
assumes *is-const e*
 $\mathcal{S} \cdot \mathcal{C} \vdash [e] : (ts \rightarrow ts')$
shows $\exists t. (ts' = ts@[t]) \wedge (\mathcal{S} \cdot \mathcal{C}' \vdash [e] : ([\] \rightarrow [t]))$
 $\langle proof \rangle$

lemma *const-typeof*:
assumes $\mathcal{S} \cdot \mathcal{C} \vdash [\$C\ v] : ([\] \rightarrow [t])$
shows *typeof v = t*
 $\langle proof \rangle$

lemma *e-type-const-list*:
assumes *const-list vs*
 $\mathcal{S} \cdot \mathcal{C} \vdash vs : (ts \rightarrow ts')$
shows $\exists tvs. ts' = ts @ tvs \wedge \text{length } vs = \text{length } tvs \wedge (\mathcal{S} \cdot \mathcal{C}' \vdash vs : ([\] \rightarrow tvs))$
 $\langle proof \rangle$

lemma *e-type-const-list-snoc*:
assumes *const-list vs*
 $\mathcal{S} \cdot \mathcal{C} \vdash vs : ([\] \rightarrow ts@[t])$
shows $\exists vs1\ v2. (\mathcal{S} \cdot \mathcal{C} \vdash vs1 : ([\] \rightarrow ts))$
 $\wedge (\mathcal{S} \cdot \mathcal{C} \vdash [v2] : (ts \rightarrow ts@[t]))$
 $\wedge (vs = vs1@[v2])$
 $\wedge \text{const-list } vs1$

$\wedge \text{is-const } v2$
 $\langle \text{proof} \rangle$

lemma *e-type-const-list-cons*:
assumes *const-list vs*
 $\mathcal{S} \cdot \mathcal{C} \vdash vs : (\Box \rightarrow (ts1 @ ts2))$
shows $\exists vs1\ vs2. (\mathcal{S} \cdot \mathcal{C} \vdash vs1 : (\Box \rightarrow ts1))$
 $\wedge (\mathcal{S} \cdot \mathcal{C} \vdash vs2 : (ts1 \rightarrow (ts1 @ ts2)))$
 $\wedge vs = vs1 @ vs2$
 $\wedge \text{const-list } vs1$
 $\wedge \text{const-list } vs2$
 $\langle \text{proof} \rangle$

lemma *e-type-const-conv-vs*:
assumes *const-list ves*
shows $\exists vs. ves = \$\$* vs$
 $\langle \text{proof} \rangle$

lemma *types-exist-lfilled*:
assumes *Lfilled k lholed es lfilled*
 $\mathcal{S} \cdot \mathcal{C} \vdash \text{lfilled} : (ts \rightarrow ts')$
shows $\exists t1s\ t2s\ C' \text{arb-label}. (\mathcal{S} \cdot \mathcal{C} \langle \text{label} := \text{arb-label} @ (\text{label } C) \rangle \vdash es : (t1s \rightarrow t2s))$
 $\langle \text{proof} \rangle$

lemma *types-exist-lfilled-weak*:
assumes *Lfilled k lholed es lfilled*
 $\mathcal{S} \cdot \mathcal{C} \vdash \text{lfilled} : (ts \rightarrow ts')$
shows $\exists t1s\ t2s\ C' \text{arb-label arb-return}. (\mathcal{S} \cdot \mathcal{C} \langle \text{label} := \text{arb-label}, \text{return} := \text{arb-return} \rangle \vdash es : (t1s \rightarrow t2s))$
 $\langle \text{proof} \rangle$

lemma *store-typing-imp-func-agree*:
assumes *store-typing s S*
 $i < \text{length } (s\text{-inst } S)$
 $j < \text{length } (\text{func-t } ((s\text{-inst } S)!i))$
shows $(s\text{func-ind } s\ i\ j) < \text{length } (s\text{-funcs } S)$
 $\text{cl-typing } S\ (s\text{func } s\ i\ j) ((s\text{-funcs } S)!(s\text{func-ind } s\ i\ j))$
 $((s\text{-funcs } S)!(s\text{func-ind } s\ i\ j)) = (\text{func-t } ((s\text{-inst } S)!i))!j$
 $\langle \text{proof} \rangle$

lemma *store-typing-imp-glob-agree*:
assumes *store-typing s S*
 $i < \text{length } (s\text{-inst } S)$
 $j < \text{length } (\text{global } ((s\text{-inst } S)!i))$
shows $(s\text{glob-ind } s\ i\ j) < \text{length } (s\text{-globs } S)$
 $\text{glob-agree } (s\text{glob } s\ i\ j) ((s\text{-globs } S)!(s\text{glob-ind } s\ i\ j))$
 $((s\text{-globs } S)!(s\text{glob-ind } s\ i\ j)) = (\text{global } ((s\text{-inst } S)!i))!j$
 $\langle \text{proof} \rangle$

lemma *store-typing-imp-mem-agree-Some*:

assumes *store-typing* $s \mathcal{S}$

$i < \text{length } (s\text{-inst } \mathcal{S})$

$\text{smem-ind } s \ i = \text{Some } j$

shows $j < \text{length } (s\text{-mem } \mathcal{S})$

$\text{mem-agree } ((\text{mem } s)!j) ((s\text{-mem } \mathcal{S})!j)$

$\exists x. ((s\text{-mem } \mathcal{S})!j) = x \wedge (\text{memory } ((s\text{-inst } \mathcal{S})!i)) = \text{Some } x$

<proof>

lemma *store-typing-imp-mem-agree-None*:

assumes *store-typing* $s \mathcal{S}$

$i < \text{length } (s\text{-inst } \mathcal{S})$

$\text{smem-ind } s \ i = \text{None}$

shows $(\text{memory } ((s\text{-inst } \mathcal{S})!i)) = \text{None}$

<proof>

lemma *store-typing-imp-mem-agree-inst*:

assumes *store-typing* $s \mathcal{S}$

$i < \text{length } (s\text{-inst } \mathcal{S})$

shows $\text{option-projr } (\text{memory } ((s\text{-inst } \mathcal{S})!i)) = \text{map-option } (\lambda j. \text{snd } ((\text{mem } s)!j))$
 $(\text{smem-ind } s \ i)$

<proof>

lemma *store-preserved-mem*:

assumes *store-typing* $s \mathcal{S}$

$s' = s[s.\text{mem} := (s.\text{mem } s)[i := (\text{mem}', \text{sec})]]$

$\text{mem-size } \text{mem}' \geq \text{mem-size } \text{orig-mem}$

$((s.\text{mem } s)!i) = (\text{orig-mem}, \text{sec})$

shows *store-typing* $s' \mathcal{S}$

<proof>

lemma *types-agree-imp-e-typing*:

assumes *types-agree* $t \ v$

shows $\mathcal{S} \cdot \mathcal{C} \vdash [\$C \ v] : ([\] \rightarrow [t])$

<proof>

lemma *list-types-agree-imp-e-typing*:

assumes *list-all2 types-agree* $ts \ vs$

shows $\mathcal{S} \cdot \mathcal{C} \vdash \$\$* \ vs : ([\] \rightarrow ts)$

<proof>

lemma *b-e-typing-imp-list-types-agree*:

assumes $\mathcal{C} \vdash (\text{map } (\lambda v. \ C \ v) \ vs) : (ts' \rightarrow ts'@ts)$

shows *list-all2 types-agree* $ts \ vs$

<proof>

lemma *e-typing-imp-list-types-agree*:

assumes $\mathcal{S} \cdot \mathcal{C} \vdash (\$\$* \ vs) : (ts' \rightarrow ts'@ts)$

shows *list-all2 types-agree ts vs*
 $\langle proof \rangle$

lemma *store-extension-imp-store-typing*:
assumes *store-extension* $s\ s'$
store-typing $s\ \mathcal{S}$
shows *store-typing* $s'\ \mathcal{S}$
<proof>

lemma *lfilled-deterministic:*
assumes L filled k $lfilled$ es les
 L filled k $lfilled$ es les'
shows $les = les'$
 $\langle proof \rangle$

lemma *b-e-typing-trust-compat*:
assumes $\mathcal{C} \vdash es : tf$
 $trust_compat\ tr\ (trust\text{-}t\ C)$
shows $\mathcal{C}(\text{trust}\text{-}t := tr) \vdash es : tf$
<proof>

lemma *e-typing-s-typing-trust-compat*:

$$\begin{aligned} \mathcal{S}\mathcal{C} \vdash es : tf &\implies trust\text{-}compat\ tr\ (trust\text{-}t\ \mathcal{C}) \implies \mathcal{S}\mathcal{C}(\lfloor trust\text{-}t := tr \rfloor) \vdash es : tf \\ \mathcal{S}\cdot tr' \cdot r \Vdash\text{-}i\ vs; es : ts &\implies trust\text{-}compat\ tr\ tr' \implies \mathcal{S}\cdot tr \cdot r \Vdash\text{-}i\ vs; es : ts \end{aligned}$$

⟨proof⟩

end

6 Lemmas for Soundness Proof

```
theory Wasm-Properties imports Wasm-Properties-Aux begin
```

6.1 Preservation

lemma t -cvt: assumes $cvt\ t\ sx\ v = Some\ v'$ shows $t = typeof\ v'$
 $\langle proof \rangle$

$$\begin{array}{l}
\textbf{lemma } \textit{store-preserved1}: \\
\textbf{assumes } \langle s; vs; es \rangle \text{ a} \rightsquigarrow \text{-i } \langle s'; vs'; es' \rangle \\
\textit{store-typing } s \ \mathcal{S} \\
\mathcal{S} \cdot \mathcal{C} \vdash es : (ts \rightarrow ts') \\
\mathcal{C} = ((s\text{-inst } \mathcal{S})!i) (\textit{trust-t} := tr, \textit{local} := local \ ((s\text{-inst } \mathcal{S})!i) \ @ \ (\textit{map typeof} \\
vs), \textit{label} := \textit{arb-label}, \textit{return} := \textit{arb-return}) \\
i < \textit{length } (s\text{-inst } \mathcal{S}) \\
\textbf{shows } \textit{store-typing } s' \ \mathcal{S} \\
\langle \textit{proof} \rangle
\end{array}$$

lemma *store-preserved*:
assumes $(s; vs; es) \quad a \rightsquigarrow -i \quad (s'; vs'; es')$

$store\text{-}typing\ s\ \mathcal{S}$
 $\mathcal{S}.tr.None \Vdash\text{-}i\ us;es : ts$
shows $store\text{-}typing\ s'\ \mathcal{S}$
 $\langle proof \rangle$

lemma $typeof\text{-}unop\text{-}testop$:
assumes $\mathcal{S}.C \vdash [\$C\ v, \$e] : (ts \rightarrow ts')$
 $(e = (Unop\text{-}i\ t\ iop)) \vee (e = (Unop\text{-}f\ t\ fop)) \vee (e = (Testop\ t\ testop))$
shows $(typeof\ v) = t$
 $e = (Unop\text{-}f\ t\ fop) \implies is\text{-}float\text{-}t\ t$
 $\langle proof \rangle$

lemma $typeof\text{-}cvtop$:
assumes $\mathcal{S}.C \vdash [\$C\ v, \$e] : (ts \rightarrow ts')$
 $e = Cvtop\ t1\ cvtop\ t\ sx$
shows $(typeof\ v) = t$
 $cvtop = Convert \implies (t1 \neq t) \wedge (t\text{-}sec\ t1 = t\text{-}sec\ t) \wedge ((sx = None) =$
 $((is\text{-}float\text{-}t\ t1 \wedge is\text{-}float\text{-}t\ t) \vee (is\text{-}int\text{-}t\ t1 \wedge is\text{-}int\text{-}t\ t \wedge (t\text{-}length\ t1 < t\text{-}length\ t))))$
 $cvtop = Reinterpret \implies (t1 \neq t) \wedge t\text{-}sec\ t1 = t\text{-}sec\ t \wedge t\text{-}length\ t1 = t\text{-}length$
 t
 $cvtop = Classify \implies is\text{-}int\text{-}t\ t \wedge is\text{-}public\text{-}t\ t \wedge classify\text{-}t\ t = t1$
 $cvtop = Declassify \implies (trust\text{-}t\ C) = Trusted \wedge is\text{-}int\text{-}t\ t \wedge is\text{-}secret\text{-}t\ t \wedge$
 $declassify\text{-}t\ t = t1$
 $\langle proof \rangle$

lemma $typeof\text{-}callcl\text{-}host$:
assumes $\mathcal{S}.C \vdash (\$ \$* us) @ [e] : (ts \rightarrow ts')$
 $e = Callcl\ cl$
 $cl = Func\text{-}host\ (tr,tf)\ f$
shows $trust\text{-}compat\ (trust\text{-}t\ C)\ tr$
 $\langle proof \rangle$

lemma $types\text{-}preserved\text{-}unop\text{-}testop\text{-}cvtop$:
assumes $\llbracket \$C\ v, \$e \rrbracket a \rightsquigarrow \llbracket \$C\ v' \rrbracket$
 $\mathcal{S}.C \vdash [\$C\ v, \$e] : (ts \rightarrow ts')$
 $(e = (Unop\text{-}i\ t\ iop)) \vee (e = (Unop\text{-}f\ t\ fop)) \vee (e = (Testop\ t\ testop)) \vee$
 $(e = (Cvtop\ t2\ cvtop\ t\ sx))$
shows $\mathcal{S}.C \vdash [\$C\ v'] : (ts \rightarrow ts')$
 $\langle proof \rangle$

lemma $typeof\text{-}binop\text{-}relop$:
assumes $\mathcal{S}.C \vdash [\$C\ v1, \$C\ v2, \$e] : (ts \rightarrow ts')$
 $e = Binop\text{-}i\ t\ iop \vee e = Binop\text{-}f\ t\ fop \vee e = Relop\text{-}i\ t\ irop \vee e = Relop\text{-}f$
 $t\ frop$
shows $typeof\ v1 = t$
 $typeof\ v2 = t$
 $e = Binop\text{-}i\ t\ iop \implies is\text{-}secret\text{-}t\ t \implies safe\text{-}binop\text{-}i\ iop$
 $e = Binop\text{-}f\ t\ fop \implies is\text{-}float\text{-}t\ t$
 $e = Relop\text{-}f\ t\ frop \implies is\text{-}float\text{-}t\ t$

$\langle \text{proof} \rangle$

lemma *types-preserved-binop-relop*:

assumes $\langle [\$C\ v1, \$C\ v2, \$e] \rangle a \rightsquigarrow \langle [\$C\ v'] \rangle$

$\mathcal{S} \cdot \mathcal{C} \vdash [\$C\ v1, \$C\ v2, \$e] : (ts \rightarrow ts')$

$e = \text{Binop-}i\ t\ iop \vee e = \text{Binop-}f\ t\ fop \vee e = \text{Relop-}i\ t\ irop \vee e = \text{Relop-}f$

$t\ frop$

shows $\mathcal{S} \cdot \mathcal{C} \vdash [\$C\ v'] : (ts \rightarrow ts')$

$\langle \text{proof} \rangle$

lemma *types-preserved-drop*:

assumes $\langle [\$C\ v, \$e] \rangle a \rightsquigarrow \langle [] \rangle$

$\mathcal{S} \cdot \mathcal{C} \vdash [\$C\ v, \$e] : (ts \rightarrow ts')$

$(e = (\text{Drop}))$

shows $\mathcal{S} \cdot \mathcal{C} \vdash [] : (ts \rightarrow ts')$

$\langle \text{proof} \rangle$

lemma *typeof-select*:

assumes $\mathcal{S} \cdot \mathcal{C} \vdash [\$C\ v1, \$C\ v2, \$C\ vn, \$e] : (ts \rightarrow ts')$

$(e = \text{Select}\ sec)$

shows $t\text{-}sec\ (typeof\ vn) = sec$

$typeof\ v1 = typeof\ v2$

$sec = \text{Secret} \longrightarrow is\text{-}secret\text{-}t\ (typeof\ v1)$

$sec = \text{Secret} \longrightarrow is\text{-}secret\text{-}t\ (typeof\ v2)$

$\langle \text{proof} \rangle$

lemma *types-preserved-select*:

assumes $\langle [\$C\ v1, \$C\ v2, \$C\ vn, \$e] \rangle a \rightsquigarrow \langle [\$C\ v3] \rangle$

$\mathcal{S} \cdot \mathcal{C} \vdash [\$C\ v1, \$C\ v2, \$C\ vn, \$e] : (ts \rightarrow ts')$

$(e = \text{Select}\ sec)$

shows $\mathcal{S} \cdot \mathcal{C} \vdash [\$C\ v3] : (ts \rightarrow ts')$

$\langle \text{proof} \rangle$

lemma *types-preserved-block*:

assumes $\langle [vs\ @\ [\$Block\ (tn \rightarrow tm)\ es]] \rangle a \rightsquigarrow \langle [Label\ m\ []\ (vs\ @\ (\$*\ es))] \rangle$

$\mathcal{S} \cdot \mathcal{C} \vdash vs\ @\ [\$Block\ (tn \rightarrow tm)\ es] : (ts \rightarrow ts')$

$const\text{-}list\ vs$

$length\ vs = n$

$length\ tn = n$

$length\ tm = m$

shows $\mathcal{S} \cdot \mathcal{C} \vdash [Label\ m\ []\ (vs\ @\ (\$*\ es))] : (ts \rightarrow ts')$

$\langle \text{proof} \rangle$

lemma *typeof-if*:

assumes $\mathcal{S} \cdot \mathcal{C} \vdash [\$C\ \text{ConstInt32}\ sec\ n, \$If\ tf\ e1s\ e2s] : (ts \rightarrow ts')$

shows $sec = \text{Public}$

$\langle \text{proof} \rangle$

lemma *types-preserved-if*:

assumes $\langle [\$C \text{ ConstInt32 } \text{sec } n, \$If \text{ tf } e1s \text{ } e2s] \rangle a \rightsquigarrow \langle [\$Block \text{ tf } es] \rangle$
 $\mathcal{S}\cdot\mathcal{C} \vdash [\$C \text{ ConstInt32 } \text{sec } n, \$If \text{ tf } e1s \text{ } e2s] : (ts \rightarrow ts')$
shows $\mathcal{S}\cdot\mathcal{C} \vdash [\$Block \text{ tf } es] : (ts \rightarrow ts')$
 $\langle proof \rangle$

lemma types-preserved-tee-local:
assumes $\langle [v, \$Tee\text{-local } i] \rangle a \rightsquigarrow \langle [v, v, \$Set\text{-local } i] \rangle$
 $\mathcal{S}\cdot\mathcal{C} \vdash [v, \$Tee\text{-local } i] : (ts \rightarrow ts')$
 $is\text{-const } v$
shows $\mathcal{S}\cdot\mathcal{C} \vdash [v, v, \$Set\text{-local } i] : (ts \rightarrow ts')$
 $\langle proof \rangle$

lemma types-preserved-loop:
assumes $\langle [vs @ [\$Loop (t1s \rightarrow t2s) \text{ es}]] \rangle a \rightsquigarrow \langle [Label \text{ } n \text{ } [\$Loop (t1s \rightarrow t2s) \text{ es}]] \rangle$
 $(vs @ (\$* \text{ es}))$
 $\mathcal{S}\cdot\mathcal{C} \vdash vs @ [\$Loop (t1s \rightarrow t2s) \text{ es}] : (ts \rightarrow ts')$
 $const\text{-list } vs$
 $length \text{ } vs = n$
 $length \text{ } t1s = n$
 $length \text{ } t2s = m$
shows $\mathcal{S}\cdot\mathcal{C} \vdash [Label \text{ } n \text{ } [\$Loop (t1s \rightarrow t2s) \text{ es}]] (vs @ (\$* \text{ es})) : (ts \rightarrow ts')$
 $\langle proof \rangle$

lemma types-preserved-label-value:
assumes $\langle [Label \text{ } n \text{ } es0 \text{ } vs] \rangle a \rightsquigarrow \langle [vs] \rangle$
 $\mathcal{S}\cdot\mathcal{C} \vdash [Label \text{ } n \text{ } es0 \text{ } vs] : (ts \rightarrow ts')$
 $const\text{-list } vs$
shows $\mathcal{S}\cdot\mathcal{C} \vdash vs : (ts \rightarrow ts')$
 $\langle proof \rangle$

lemma typeof-br-if:
assumes $\mathcal{S}\cdot\mathcal{C} \vdash [\$C \text{ ConstInt32 } \text{sec } n, \$Br\text{-if } i] : (ts \rightarrow ts')$
shows $sec = Public$
 $\langle proof \rangle$

lemma types-preserved-br-if:
assumes $\langle [\$C \text{ ConstInt32 } \text{sec } n, \$Br\text{-if } i] \rangle a \rightsquigarrow \langle [e] \rangle$
 $\mathcal{S}\cdot\mathcal{C} \vdash [\$C \text{ ConstInt32 } \text{sec } n, \$Br\text{-if } i] : (ts \rightarrow ts')$
 $e = [\$Br \text{ } i] \vee e = []$
shows $\mathcal{S}\cdot\mathcal{C} \vdash e : (ts \rightarrow ts')$
 $\langle proof \rangle$

lemma typeof-br-table:
assumes $\mathcal{S}\cdot\mathcal{C} \vdash [\$C \text{ ConstInt32 } \text{sec } c, \$Br\text{-table is } i] : (ts \rightarrow ts')$
shows $sec = Public$
 $\langle proof \rangle$

lemma types-preserved-br-table:
assumes $\langle [\$C \text{ ConstInt32 } \text{sec } c, \$Br\text{-table is } i] \rangle a \rightsquigarrow \langle [\$Br \text{ } i] \rangle$

$\mathcal{S} \cdot \mathcal{C} \vdash [\$C \text{ ConstInt32 } \text{sec } c, \$Br\text{-table } is \ i] : (ts \rightarrow ts')$
 $(i' = (is \ ! \ \text{nat-of-int } c) \wedge \text{length } is > \text{nat-of-int } c) \vee i' = i$
shows $\mathcal{S} \cdot \mathcal{C} \vdash [\$Br \ i'] : (ts \rightarrow ts')$
 $\langle \text{proof} \rangle$

lemma *types-preserved-local-const*:
assumes $\langle [Local \ n \ i \ \text{vs } es] \rangle a \rightsquigarrow \langle es \rangle$
 $\mathcal{S} \cdot \mathcal{C} \vdash [Local \ n \ i \ \text{vs } es] : (ts \rightarrow ts')$
 $\text{const-list } es$
shows $\mathcal{S} \cdot \mathcal{C} \vdash es : (ts \rightarrow ts')$
 $\langle \text{proof} \rangle$

lemma *typing-map-typeof*:
assumes $ves = \$\$* \ \text{vs}$
 $\mathcal{S} \cdot \mathcal{C} \vdash ves : (\ [] \rightarrow tvs)$
shows $tvs = \text{map } \text{typeof } \text{vs}$
 $\langle \text{proof} \rangle$

lemma *types-preserved-call-indirect-Some*:
assumes $\mathcal{S} \cdot \mathcal{C} \vdash [\$C \text{ ConstInt32 } \text{sec } c, \$Call\text{-indirect } j] : (ts \rightarrow ts')$
 $\text{stab } s \ i' \ (\text{nat-of-int } c) = \text{Some } cl$
 $\text{stypes } s \ i' \ j = (tr', tf)$
 $cl\text{-type } cl = (tr', tf)$
 $\text{store-typing } s \ \mathcal{S}$
 $i' < \text{length } (\text{inst } s)$
 $C = (s\text{-inst } \mathcal{S} \ ! \ i') \ (\text{trust-}t := tr, \text{local} := \text{local } (s\text{-inst } \mathcal{S} \ ! \ i') \ @ \ tvs, \text{label}$
 $:= \text{arb-labs}, \text{return} := \text{arb-return})$
shows $\mathcal{S} \cdot \mathcal{C} \vdash [Callcl \ cl] : (ts \rightarrow ts')$
 $\text{sec} = \text{Public}$
 $\langle \text{proof} \rangle$

lemma *types-preserved-call-indirect-None*:
assumes $\mathcal{S} \cdot \mathcal{C} \vdash [\$C \text{ ConstInt32 } \text{sec } c, \$Call\text{-indirect } j] : (ts \rightarrow ts')$
shows $\mathcal{S} \cdot \mathcal{C} \vdash [Trap] : (ts \rightarrow ts')$
 $\text{sec} = \text{Public}$
 $\langle \text{proof} \rangle$

lemma *types-preserved-callcl-native*:
assumes $\mathcal{S} \cdot \mathcal{C} \vdash ves \ @ \ [Callcl \ cl] : (ts \rightarrow ts')$
 $cl = \text{Func-native } i \ (tr, (t1s \rightarrow t2s)) \ tfs \ es$
 $ves = \$\$* \ \text{vs}$
 $\text{length } \text{vs} = n$
 $\text{length } tfs = k$
 $\text{length } t1s = n$
 $\text{length } t2s = m$
 $n\text{-zeros } tfs = zs$
 $\text{store-typing } s \ \mathcal{S}$
shows $\mathcal{S} \cdot \mathcal{C} \vdash [Local \ m \ i \ (vs \ @ \ zs) \ [\$Block \ (\ [] \rightarrow t2s) \ es]] : (ts \rightarrow ts')$
 $\langle \text{proof} \rangle$

lemma *types-preserved-callcl-host-some*:

assumes $\mathcal{S}\cdot\mathcal{C} \vdash ves @ [Callcl\ cl] : (ts \rightarrow ts')$
 $cl = Func-host\ (tr, (t1s \rightarrow t2s))\ f$
 $ves = \$\$* vcs$
 $length\ vcs = n$
 $length\ t1s = n$
 $length\ t2s = m$
 $host-apply\ s\ (t1s \rightarrow t2s)\ f\ vcs\ hs = Some\ (s', vcs')$
 $store-typing\ s\ \mathcal{S}$
shows $\mathcal{S}\cdot\mathcal{C} \vdash \$\$* vcs' : (ts \rightarrow ts')$
 $\langle proof \rangle$

lemma *types-imp-concat*:

assumes $\mathcal{S}\cdot\mathcal{C} \vdash es @ [e] @ es' : (ts \rightarrow ts')$
 $\bigwedge tes\ tes'. ((\mathcal{S}\cdot\mathcal{C} \vdash [e] : (tes \rightarrow tes')) \implies (\mathcal{S}\cdot\mathcal{C} \vdash [e'] : (tes \rightarrow tes')))$
shows $\mathcal{S}\cdot\mathcal{C} \vdash es @ [e'] @ es' : (ts \rightarrow ts')$
 $\langle proof \rangle$

lemma *type-const-return*:

assumes $Lfilled\ i\ lhole d\ (vs @ [Return])\ LI$
 $(return\ \mathcal{C}) = Some\ tcs$
 $length\ tcs = length\ vs$
 $\mathcal{S}\cdot\mathcal{C} \vdash LI : (ts \rightarrow ts')$
 $const-list\ vs$
shows $\mathcal{S}\cdot\mathcal{C}' \vdash vs : ([] \rightarrow tcs)$
 $\langle proof \rangle$

lemma *types-preserved-return*:

assumes $([Local\ n\ i\ vls\ LI])\ a \rightsquigarrow ([ves])$
 $\mathcal{S}\cdot\mathcal{C} \vdash [Local\ n\ i\ vls\ LI] : (ts \rightarrow ts')$
 $const-list\ ves$
 $length\ ves = n$
 $Lfilled\ j\ lhole d\ (ves @ [Return])\ LI$
shows $\mathcal{S}\cdot\mathcal{C} \vdash ves : (ts \rightarrow ts')$
 $\langle proof \rangle$

lemma *type-const-br*:

assumes $Lfilled\ i\ lhole d\ (vs @ [Br\ (i+k)])\ LI$
 $length\ (label\ \mathcal{C}) > k$
 $(label\ \mathcal{C})!k = tcs$
 $length\ tcs = length\ vs$
 $\mathcal{S}\cdot\mathcal{C} \vdash LI : (ts \rightarrow ts')$
 $const-list\ vs$
shows $\mathcal{S}\cdot\mathcal{C}' \vdash vs : ([] \rightarrow tcs)$
 $\langle proof \rangle$

lemma *types-preserved-br*:

assumes $([Label\ n\ es0\ LI])\ a \rightsquigarrow ([vs @ es0])$

$\mathcal{S} \cdot \mathcal{C} \vdash [\text{Label } n \text{ es0 } LI] : (ts \rightarrow ts')$
 $\text{const-list } vs$
 $\text{length } vs = n$
 $L\text{filled } i \text{ lholed } (vs @ [\text{\$Br } i]) \text{ } LI$
shows $\mathcal{S} \cdot \mathcal{C} \vdash (vs @ \text{es0}) : (ts \rightarrow ts')$
 $\langle \text{proof} \rangle$

lemma *store-local-label-empty*:
assumes $i < \text{length } (s\text{-inst } \mathcal{S})$
 $\text{store-typing } s \ \mathcal{S}$
shows $\text{label } ((s\text{-inst } \mathcal{S})!i) = [] \text{ local } ((s\text{-inst } \mathcal{S})!i) = []$
 $\langle \text{proof} \rangle$

lemma *types-preserved-b-e1*:
assumes $\langle \text{es} \rangle \rightsquigarrow \langle \text{es}' \rangle$
 $\text{store-typing } s \ \mathcal{S}$
 $\mathcal{S} \cdot \mathcal{C} \vdash \text{es} : (ts \rightarrow ts')$
shows $\mathcal{S} \cdot \mathcal{C} \vdash \text{es}' : (ts \rightarrow ts')$
 $\langle \text{proof} \rangle$

lemma *types-preserved-b-e*:
assumes $\langle \text{es} \rangle \rightsquigarrow \langle \text{es}' \rangle$
 $\text{store-typing } s \ \mathcal{S}$
 $\mathcal{S} \cdot \text{tr} \cdot \text{None} \Vdash\text{-i } vs; \text{es} : ts$
shows $\mathcal{S} \cdot \text{tr} \cdot \text{None} \Vdash\text{-i } vs; \text{es}' : ts$
 $\langle \text{proof} \rangle$

lemma *types-preserved-store*:
assumes $\mathcal{S} \cdot \mathcal{C} \vdash [\text{\$C ConstInt32 sec } k, \text{\$C } v, \text{\$Store } t \text{ tp } a \text{ off}] : (ts \rightarrow ts')$
shows $\mathcal{S} \cdot \mathcal{C} \vdash [] : (ts \rightarrow ts')$
 $\text{sec} = \text{Public}$
 $\text{types-agree } t \ v$
 $\langle \text{proof} \rangle$

lemma *types-preserved-current-memory*:
assumes $\mathcal{S} \cdot \mathcal{C} \vdash [\text{\$Current-memory}] : (ts \rightarrow ts')$
shows $\mathcal{S} \cdot \mathcal{C} \vdash [\text{\$C ConstInt32 Public } c] : (ts \rightarrow ts')$
 $\langle \text{proof} \rangle$

lemma *types-preserved-grow-memory*:
assumes $\mathcal{S} \cdot \mathcal{C} \vdash [\text{\$C ConstInt32 sec } c, \text{\$Grow-memory}] : (ts \rightarrow ts')$
shows $\mathcal{S} \cdot \mathcal{C} \vdash [\text{\$C ConstInt32 sec } c'] : (ts \rightarrow ts')$
 $\text{sec} = \text{Public}$
 $\langle \text{proof} \rangle$

lemma *types-preserved-set-global*:
assumes $\mathcal{S} \cdot \mathcal{C} \vdash [\text{\$C } v, \text{\$Set-global } j] : (ts \rightarrow ts')$
shows $\mathcal{S} \cdot \mathcal{C} \vdash [] : (ts \rightarrow ts')$
 $\text{tg-t } (\text{global } \mathcal{C} ! j) = \text{typeof } v$

$\langle proof \rangle$

lemma *types-preserved-load*:

assumes $\mathcal{S}\cdot\mathcal{C} \vdash [\$C \text{ ConstInt32 } sec \ k, \$Load \ t \ tp \ a \ off] : (ts \rightarrow ts')$
 $typeof \ v = t$
shows $\mathcal{S}\cdot\mathcal{C} \vdash [\$C \ v] : (ts \rightarrow ts')$
 $sec = Public$

$\langle proof \rangle$

lemma *types-preserved-get-local*:

assumes $\mathcal{S}\cdot\mathcal{C} \vdash [\$Get-local \ i] : (ts \rightarrow ts')$
 $length \ vi = i$
 $(local \ C) = map \ typeof \ (vi \ @ \ [v] \ @ \ vs)$
shows $\mathcal{S}\cdot\mathcal{C} \vdash [\$C \ v] : (ts \rightarrow ts')$

$\langle proof \rangle$

lemma *types-preserved-set-local*:

assumes $\mathcal{S}\cdot\mathcal{C} \vdash [\$C \ v', \$Set-local \ i] : (ts \rightarrow ts')$
 $length \ vi = i$
 $(local \ C) = map \ typeof \ (vi \ @ \ [v] \ @ \ vs)$
shows $(\mathcal{S}\cdot\mathcal{C} \vdash [] : (ts \rightarrow ts')) \wedge map \ typeof \ (vi \ @ \ [v] \ @ \ vs) = map \ typeof \ (vi \ @ \ [v'] \ @ \ vs)$
 $\langle proof \rangle$

lemma *types-preserved-get-global*:

assumes $typeof \ (sglob-val \ s \ i \ j) = tg-t \ (global \ C \ ! \ j)$
 $\mathcal{S}\cdot\mathcal{C} \vdash [\$Get-global \ j] : (ts \rightarrow ts')$
shows $\mathcal{S}\cdot\mathcal{C} \vdash [\$C \ sglob-val \ s \ i \ j] : (ts \rightarrow ts')$
 $\langle proof \rangle$

lemma *lholed-same-type*:

assumes $Lfilled \ k \ lholed \ es \ les$
 $Lfilled \ k \ lholed \ es' \ les'$
 $\mathcal{S}\cdot\mathcal{C} \vdash les : (ts \rightarrow ts')$
 $\bigwedge arb-labs \ ts \ ts'.$
 $\mathcal{S}\cdot(\mathcal{C}(\lfloor label := arb-labs @ (label \ C) \rfloor)) \vdash es : (ts \rightarrow ts')$
 $\implies \mathcal{S}\cdot(\mathcal{C}(\lfloor label := arb-labs @ (label \ C) \rfloor)) \vdash es' : (ts \rightarrow ts')$
shows $(\mathcal{S}\cdot\mathcal{C} \vdash les' : (ts \rightarrow ts'))$
 $\langle proof \rangle$

lemma *types-preserved-e1*:

assumes $(\lfloor s; vs; es \rfloor) a \rightsquigarrow -i (\lfloor s'; vs'; es' \rfloor)$
 $store-typing \ s \ \mathcal{S}$
 $tvs = map \ typeof \ vs$
 $i < length \ (inst \ s)$
 $C = ((s-inst \ \mathcal{S})!i)(\lfloor trust-t := tr, local := (local \ ((s-inst \ \mathcal{S})!i) \ @ \ tvs), label$
 $:= arb-labs, return := arb-return \rfloor)$
 $\mathcal{S}\cdot\mathcal{C} \vdash es : (ts \rightarrow ts')$
shows $(\mathcal{S}\cdot\mathcal{C} \vdash es' : (ts \rightarrow ts')) \wedge (map \ typeof \ vs = map \ typeof \ vs')$

$\langle \text{proof} \rangle$

lemma *types-preserved-e*:

assumes $\langle s; vs; es \rangle \rightsquigarrow\!\!\sim i \langle s'; vs'; es' \rangle$
store-typing $s \ \mathcal{S}$
 $\mathcal{S} \cdot \text{tr} \cdot \text{None} \Vdash\!\!\sim i \text{ vs; es} : ts$
shows $\mathcal{S} \cdot \text{tr} \cdot \text{None} \Vdash\!\!\sim i \text{ vs'; es'} : ts$
 $\langle \text{proof} \rangle$

6.2 Progress

lemma *const-list-no-progress*:

assumes *const-list* es
shows $\neg \langle s; vs; es \rangle \rightsquigarrow\!\!\sim i \langle s'; vs'; es' \rangle$
 $\langle \text{proof} \rangle$

lemma *empty-no-progress*:

assumes $es = []$
shows $\neg \langle s; vs; es \rangle \rightsquigarrow\!\!\sim i \langle s'; vs'; es' \rangle$
 $\langle \text{proof} \rangle$

lemma *trap-no-progress*:

assumes $es = [\text{Trap}]$
shows $\neg \langle s; vs; es \rangle \rightsquigarrow\!\!\sim i \langle s'; vs'; es' \rangle$
 $\langle \text{proof} \rangle$

lemma *terminal-no-progress*:

assumes *const-list* $es \vee es = [\text{Trap}]$
shows $\neg \langle s; vs; es \rangle \rightsquigarrow\!\!\sim i \langle s'; vs'; es' \rangle$
 $\langle \text{proof} \rangle$

lemma *progress-L0*:

assumes $\langle s; vs; es \rangle \rightsquigarrow\!\!\sim i \langle s'; vs'; es' \rangle$
const-list cs
shows $\langle s; vs; cs @ es @ es - c \rangle \rightsquigarrow\!\!\sim i \langle s'; vs'; cs @ es' @ es - c \rangle$
 $\langle \text{proof} \rangle$

lemma *progress-L0-left*:

assumes $\langle s; vs; es \rangle \rightsquigarrow\!\!\sim i \langle s'; vs'; es' \rangle$
const-list cs
shows $\langle s; vs; cs @ es \rangle \rightsquigarrow\!\!\sim i \langle s'; vs'; cs @ es' \rangle$
 $\langle \text{proof} \rangle$

lemma *progress-L0-trap*:

assumes *const-list* cs
 $cs \neq [] \vee es \neq []$
shows $\exists a. \langle s; vs; cs @ [\text{Trap}] @ es \rangle \rightsquigarrow\!\!\sim i \langle s; vs; [\text{Trap}] \rangle$
 $\langle \text{proof} \rangle$

lemma *progress-LN*:
assumes $(Lfilled\ j\ lhole\ [\$Br\ (j+k)]\ es)$
 $\mathcal{S}\cdot\mathcal{C} \vdash es : ([\] \rightarrow ts)$
 $(label\ \mathcal{C})!k = tvs$
shows $\exists\ lhole'\ vs\ \mathcal{C}'. (Lfilled\ j\ lhole'\ (vs@[\$Br\ (j+k)])\ es)$
 $\wedge (\mathcal{S}\cdot\mathcal{C}' \vdash vs : ([\] \rightarrow tvs))$
 $\wedge\ const\text{-}list\ vs$
 $\langle proof \rangle$

lemma *progress-LN-return*:
assumes $(Lfilled\ j\ lhole\ [\$Return]\ es)$
 $\mathcal{S}\cdot\mathcal{C} \vdash es : ([\] \rightarrow ts)$
 $(return\ \mathcal{C}) = Some\ tvs$
shows $\exists\ lhole'\ vs\ \mathcal{C}'. (Lfilled\ j\ lhole'\ (vs@[\$Return])\ es)$
 $\wedge (\mathcal{S}\cdot\mathcal{C}' \vdash vs : ([\] \rightarrow tvs))$
 $\wedge\ const\text{-}list\ vs$
 $\langle proof \rangle$

lemma *progress-LN1*:
assumes $(Lfilled\ j\ lhole\ [\$Br\ (j+k)]\ es)$
 $\mathcal{S}\cdot\mathcal{C} \vdash es : (ts \rightarrow ts')$
shows $length\ (label\ \mathcal{C}) > k$
 $\langle proof \rangle$

lemma *progress-LN2*:
assumes $(Lfilled\ j\ lhole\ e1s\ lfilled)$
shows $\exists\ lfilled'. (Lfilled\ j\ lhole\ e2s\ lfilled')$
 $\langle proof \rangle$

lemma *const-of-const-list*:
assumes $length\ cs = 1$
 $const\text{-}list\ cs$
shows $\exists\ v. cs = [\$C\ v]$
 $\langle proof \rangle$

lemma *const-of-i32*:
assumes $const\text{-}list\ cs$
 $\mathcal{S}\cdot\mathcal{C} \vdash cs : ([\] \rightarrow [(T\text{-}i32\ sec)])$
shows $\exists\ c. cs = [\$C\ ConstInt32\ sec\ c]$
 $\langle proof \rangle$

lemma *const-of-i64*:
assumes $const\text{-}list\ cs$
 $\mathcal{S}\cdot\mathcal{C} \vdash cs : ([\] \rightarrow [(T\text{-}i64\ sec)])$
shows $\exists\ c. cs = [\$C\ ConstInt64\ sec\ c]$
 $\langle proof \rangle$

lemma *const-of-f32*:
assumes $const\text{-}list\ cs$

$\mathcal{S}\cdot\mathcal{C} \vdash cs : ([\] \rightarrow [T\text{-}f32])$
shows $\exists c. cs = [\$C \text{ ConstFloat32 } c]$
 $\langle proof \rangle$

lemma *const-of-f64*:
assumes *const-list cs*
 $\mathcal{S}\cdot\mathcal{C} \vdash cs : ([\] \rightarrow [T\text{-}f64])$
shows $\exists c. cs = [\$C \text{ ConstFloat64 } c]$
 $\langle proof \rangle$

lemma *progress-unop-testop-i*:
assumes $\mathcal{S}\cdot\mathcal{C} \vdash cs : ([\] \rightarrow [t])$
is-int-t t
const-list cs
 $e = \text{Unop-}i \ t \ iop \vee e = \text{Testop } t \ testop$
shows $\exists a \ s' \ vs' \ es'. ([s;vs;cs@([\$e]]) \ a \rightsquigarrow\text{-}i ([s';vs';es']))$
 $\langle proof \rangle$

lemma *progress-unop-f*:
assumes $\mathcal{S}\cdot\mathcal{C} \vdash cs : ([\] \rightarrow [t])$
is-float-t t
const-list cs
 $e = \text{Unop-}f \ t \ iop$
shows $\exists a \ s' \ vs' \ es'. ([s;vs;cs@([\$e]]) \ a \rightsquigarrow\text{-}i ([s';vs';es']))$
 $\langle proof \rangle$

lemma *const-list-split-2*:
assumes *const-list cs*
 $\mathcal{S}\cdot\mathcal{C} \vdash cs : ([\] \rightarrow [t1, t2])$
shows $\exists c1 \ c2. (\mathcal{S}\cdot\mathcal{C} \vdash [c1] : ([\] \rightarrow [t1]))$
 $\wedge (\mathcal{S}\cdot\mathcal{C} \vdash [c2] : ([\] \rightarrow [t2]))$
 $\wedge cs = [c1, c2]$
 $\wedge \text{const-list } [c1]$
 $\wedge \text{const-list } [c2]$
 $\langle proof \rangle$

lemma *const-list-split-3*:
assumes *const-list cs*
 $\mathcal{S}\cdot\mathcal{C} \vdash cs : ([\] \rightarrow [t1, t2, t3])$
shows $\exists c1 \ c2 \ c3. (\mathcal{S}\cdot\mathcal{C} \vdash [c1] : ([\] \rightarrow [t1]))$
 $\wedge (\mathcal{S}\cdot\mathcal{C} \vdash [c2] : ([\] \rightarrow [t2]))$
 $\wedge (\mathcal{S}\cdot\mathcal{C} \vdash [c3] : ([\] \rightarrow [t3]))$
 $\wedge cs = [c1, c2, c3]$
 $\langle proof \rangle$

lemma *progress-binop-relop-i*:
assumes $\mathcal{S}\cdot\mathcal{C} \vdash cs : ([\] \rightarrow [t, t])$
is-int-t t
const-list cs

$e = \text{Binop-}i\ t\ iop \vee e = \text{Relop-}i\ t\ irop$
shows $\exists a\ s'\ vs'\ es'. \langle s; vs; cs@([\$e]) \rangle\ a \rightsquigarrow\text{-}i\ \langle s'; vs'; es' \rangle$
 $\langle \text{proof} \rangle$

lemma *progress-binop-relop-f*:
assumes $\mathcal{S}\cdot\mathcal{C} \vdash cs : ([\] \rightarrow [t, t])$
 $is\text{-}float\text{-}t\ t$
 $const\text{-}list\ cs$
 $e = \text{Binop-}f\ t\ fop \vee e = \text{Relop-}f\ t\ frop$
shows $\exists a\ s'\ vs'\ es'. \langle s; vs; cs@([\$e]) \rangle\ a \rightsquigarrow\text{-}i\ \langle s'; vs'; es' \rangle$
 $\langle \text{proof} \rangle$

lemma *progress-b-e*:
assumes $\mathcal{C} \vdash b\text{-}es : (ts \rightarrow ts')$
 $\mathcal{S}\cdot\mathcal{C} \vdash cs : ([\] \rightarrow ts)$
 $(\bigwedge l\text{holed}. \neg(Lfilled\ 0\ l\text{holed}\ [\$Return]\ (cs@(\$*b\text{-}es))))$
 $\bigwedge i\ l\text{holed}. \neg(Lfilled\ 0\ l\text{holed}\ [\$Br\ (i)]\ (cs@(\$*b\text{-}es)))$
 $const\text{-}list\ cs$
 $\neg\ const\text{-}list\ (\$*b\text{-}es)$
 $i < length\ (s\text{-}inst\ \mathcal{S})$
 $length\ (local\ \mathcal{C}) = length\ (vs)$
 $option\text{-}projr\ (memory\ \mathcal{C}) = map\text{-}option\ (\lambda j. snd\ ((mem\ s)!j))\ (smem\text{-}ind$
 $s\ i)$
shows $\exists a\ s'\ vs'\ es'. \langle s; vs; cs@(\$*b\text{-}es) \rangle\ a \rightsquigarrow\text{-}i\ \langle s'; vs'; es' \rangle$
 $\langle \text{proof} \rangle$

lemma *progress-e*:
assumes $\mathcal{S}\cdot tr\cdot None \Vdash\text{-}i\ vs; cs\text{-}es : ts'$
 $\bigwedge k\ l\text{holed}. \neg(Lfilled\ k\ l\text{holed}\ [\$Return]\ cs\text{-}es)$
 $\bigwedge i\ k\ l\text{holed}. (Lfilled\ k\ l\text{holed}\ [\$Br\ (i)]\ cs\text{-}es) \implies i < k$
 $cs\text{-}es \neq [Trap]$
 $\neg\ const\text{-}list\ (cs\text{-}es)$
 $store\text{-}typing\ s\ \mathcal{S}$
shows $\exists a\ s'\ vs'\ es'. \langle s; vs; cs\text{-}es \rangle\ a \rightsquigarrow\text{-}i\ \langle s'; vs'; es' \rangle$
 $\langle \text{proof} \rangle$

lemma *progress-e1*:
assumes $\mathcal{S}\cdot tr\cdot None \Vdash\text{-}i\ vs; es : ts$
shows $\neg(Lfilled\ k\ l\text{holed}\ [\$Return]\ es)$
 $\langle \text{proof} \rangle$

lemma *progress-e2*:
assumes $\mathcal{S}\cdot tr\cdot None \Vdash\text{-}i\ vs; es : ts$
 $store\text{-}typing\ s\ \mathcal{S}$
shows $(Lfilled\ k\ l\text{holed}\ [\$Br\ (j)]\ es) \implies j < k$
 $\langle \text{proof} \rangle$

lemma *progress-e3*:
assumes $\mathcal{S}\cdot tr\cdot None \Vdash\text{-}i\ vs; cs\text{-}es : ts'$

```

      cs-es ≠ [Trap]
      ¬ const-list (cs-es)
      store-typing s S
shows ∃ a s' vs' es'. (s;vs;cs-es) a↗-i (s';vs';es')
    <proof>

end

```

7 Soundness Theorems

theory *Wasm-Soundness* **imports** *Main Wasm-Properties* **begin**

theorem *preservation*:

```

assumes ⊢-i s;vs;es : (tr,ts)
          (s;vs;es) a↗-i (s';vs';es')
shows ⊢-i s';vs';es' : (tr,ts)
    <proof>

```

theorem *progress*:

```

assumes ⊢-i s;vs;es : (tr,ts)
shows const-list es ∨ es = [Trap] ∨ (∃ a s' vs' es'. (s;vs;es) a↗-i (s';vs';es'))
    <proof>

```

end

8 Augmented Type Syntax for Concrete Checker

theory *Wasm-Checker-Types* **imports** *Wasm HOL-Library.Sublist* **begin**

datatype *ct* =

```

  TAny
| TSecret
| TSome t

```

datatype *checker-type* =

```

  TopType ct list
| Type t list
| Bot

```

definition *to-ct-list* :: *t list* ⇒ *ct list* **where**

```

  to-ct-list ts = map TSome ts

```

fun *can-secret-ct* :: *ct* ⇒ *bool* **where**

```

  can-secret-ct (TSome t) = is-secret-t t
| can-secret-ct - = True

```

fun *sec-ct* :: *sec* ⇒ *ct* **where**

```

  sec-ct Public = TAny

```

| *sec-ct Private* = *TSecret*

fun *ct-eq* :: *ct* \Rightarrow *ct* \Rightarrow *bool* **where**
 ct-eq (*TSome* *t*) (*TSome* *t'*) = (*t* = *t'*)
 | *ct-eq* *TSecret* *ct* = *can-secret-ct* *ct*
 | *ct-eq* *ct* *TSecret* = *can-secret-ct* *ct*
 | *ct-eq* *TAny* - = *True*
 | *ct-eq* - *TAny* = *True*

definition *ct-list-eq* :: *ct list* \Rightarrow *ct list* \Rightarrow *bool* **where**
 ct-list-eq *ct1s* *ct2s* = *list-all2* *ct-eq* *ct1s* *ct2s*

definition *ct-prefix* :: *ct list* \Rightarrow *ct list* \Rightarrow *bool* **where**
 ct-prefix *xs* *ys* = (\exists *as* *bs*. *ys* = *as@bs* \wedge *ct-list-eq* *as* *xs*)

definition *ct-suffix* :: *ct list* \Rightarrow *ct list* \Rightarrow *bool* **where**
 ct-suffix *xs* *ys* = (\exists *as* *bs*. *ys* = *as@bs* \wedge *ct-list-eq* *bs* *xs*)

lemma *ct-eq-commute*:
 assumes *ct-eq* *x* *y*
 shows *ct-eq* *y* *x*
 \langle *proof* \rangle

lemma *ct-eq-flip*: *ct-eq*⁻¹⁻¹ = *ct-eq*
 \langle *proof* \rangle

lemma *exists-secret*: \exists *t*. *is-secret-t* *t*
 \langle *proof* \rangle

lemma *ct-eq-common-tsome*: *ct-eq* *x* *y* = (\exists *t*. *ct-eq* *x* (*TSome* *t*) \wedge *ct-eq* (*TSome* *t*) *y*)
 \langle *proof* \rangle

lemma *ct-list-eq-commute*:
 assumes *ct-list-eq* *xs* *ys*
 shows *ct-list-eq* *ys* *xs*
 \langle *proof* \rangle

lemma *ct-list-eq-refl*: *ct-list-eq* *xs* *xs*
 \langle *proof* \rangle

lemma *ct-list-eq-length*:
 assumes *ct-list-eq* *xs* *ys*
 shows *length* *xs* = *length* *ys*
 \langle *proof* \rangle

lemma *ct-list-eq-concat*:
 assumes *ct-list-eq* *xs* *ys*
 ct-list-eq *xs'* *ys'*

shows $ct\text{-list-eq } (xs @ xs') (ys @ ys')$
 $\langle proof \rangle$

lemma $ct\text{-list-eq-ts-conv-eq}$:
 $ct\text{-list-eq } (to\text{-ct-list } ts) (to\text{-ct-list } ts') = (ts = ts')$
 $\langle proof \rangle$

lemma $ct\text{-list-eq-exists}$: $\exists ys. ct\text{-list-eq } xs (to\text{-ct-list } ys)$
 $\langle proof \rangle$

lemma $ct\text{-list-eq-common-tsome-list}$:
 $ct\text{-list-eq } xs ys = (\exists zs. ct\text{-list-eq } xs (to\text{-ct-list } zs) \wedge ct\text{-list-eq } (to\text{-ct-list } zs) ys)$
 $\langle proof \rangle$

lemma $ct\text{-list-eq-cons-ct-list}$:
assumes $ct\text{-list-eq } (to\text{-ct-list } as) (xs @ ys)$
shows $\exists bs \ bs'. as = bs @ bs' \wedge ct\text{-list-eq } (to\text{-ct-list } bs) xs \wedge ct\text{-list-eq } (to\text{-ct-list } bs') ys$
 $\langle proof \rangle$

lemma $ct\text{-list-eq-cons-ct-list1}$:
assumes $ct\text{-list-eq } (to\text{-ct-list } as) (xs @ (to\text{-ct-list } ys))$
shows $\exists bs. as = bs @ ys \wedge ct\text{-list-eq } (to\text{-ct-list } bs) xs$
 $\langle proof \rangle$

lemma $ct\text{-list-eq-shared}$:
assumes $ct\text{-list-eq } xs (to\text{-ct-list } as)$
 $ct\text{-list-eq } ys (to\text{-ct-list } as)$
shows $ct\text{-list-eq } xs ys$
 $\langle proof \rangle$

lemma $ct\text{-list-eq-take}$:
assumes $ct\text{-list-eq } xs ys$
shows $ct\text{-list-eq } (take \ n \ xs) (take \ n \ ys)$
 $\langle proof \rangle$

lemma $ct\text{-prefixI}$ $[intro?]$:
assumes $ys = as @ zs$
 $ct\text{-list-eq } as \ xs$
shows $ct\text{-prefix } xs \ ys$
 $\langle proof \rangle$

lemma $ct\text{-prefixE}$ $[elim?]$:
assumes $ct\text{-prefix } xs \ ys$
obtains $as \ zs$ **where** $ys = as @ zs$ $ct\text{-list-eq } as \ xs$
 $\langle proof \rangle$

lemma $ct\text{-prefix-snoc}$ $[simp]$: $ct\text{-prefix } xs \ (ys @ [y]) = (ct\text{-list-eq } xs \ (ys @ [y]) \vee ct\text{-prefix } xs \ ys)$

$\langle proof \rangle$

lemma *ct-prefix-nil:ct-prefix* [] *xs*
 $\neg ct_prefix\ (x \# xs)\ []$
 $\langle proof \rangle$

lemma *Cons-ct-prefix-Cons[simp]*: *ct-prefix* (*x* # *xs*) (*y* # *ys*) = ((*ct-eq* *x y*) \wedge *ct-prefix* *xs ys*)
 $\langle proof \rangle$

lemma *ct-prefix-code* [*code*]:
 ct-prefix [] *xs* = *True*
 ct-prefix (*x* # *xs*) [] = *False*
 ct-prefix (*x* # *xs*) (*y* # *ys*) = ((*ct-eq* *x y*) \wedge *ct-prefix* *xs ys*)
 $\langle proof \rangle$

lemma *ct-suffix-to-ct-prefix* [*code*]: *ct-suffix* *xs ys* = *ct-prefix* (*rev xs*) (*rev ys*)
 $\langle proof \rangle$

lemma *inj-TSome*: *inj* *TSome*
 $\langle proof \rangle$

lemma *to-ct-list-append*:
 assumes *to-ct-list* *ts* = *as*@*bs*
 shows $\exists as'.\ to_ct_list\ as' = as$
 $\exists bs'.\ to_ct_list\ bs' = bs$
 $\langle proof \rangle$

lemma *ct-suffixI* [*intro?*]:
 assumes *ys* = *as* @ *zs*
 ct-list-eq *zs xs*
 shows *ct-suffix* *xs ys*
 $\langle proof \rangle$

lemma *ct-suffixE* [*elim?*]:
 assumes *ct-suffix* *xs ys*
 obtains *as zs* **where** *ys* = *as* @ *zs* *ct-list-eq* *zs xs*
 $\langle proof \rangle$

lemma *ct-suffix-nil*: *ct-suffix* [] *ts*
 $\langle proof \rangle$

lemma *ct-suffix-refl*: *ct-suffix* *ts ts*
 $\langle proof \rangle$

lemma *ct-suffix-length*:
 assumes *ct-suffix* *ts ts'*
 shows *length* *ts* \leq *length* *ts'*
 $\langle proof \rangle$

lemma *ct-suffix-take*:

assumes *ct-suffix* *ts ts'*

shows *ct-suffix* ((take (length *ts* - *n*) *ts*)) ((take (length *ts'* - *n*) *ts'*))

⟨*proof*⟩

lemma *ct-suffix-ts-conv-suffix*:

ct-suffix (to-ct-list *ts*) (to-ct-list *ts'*) = *suffix* *ts ts'*

⟨*proof*⟩

lemma *ct-suffix-exists*: \exists *ts-c*. *ct-suffix* *x1* (to-ct-list *ts-c*)

⟨*proof*⟩

lemma *ct-suffix-ct-list-eq-exists*:

assumes *ct-suffix* *x1 x2*

shows \exists *ts-c*. *ct-suffix* *x1* (to-ct-list *ts-c*) \wedge *ct-list-eq* (to-ct-list *ts-c*) *x2*

⟨*proof*⟩

lemma *ct-suffix-cons-ct-list*:

assumes *ct-suffix* (*xs@ys*) (to-ct-list *zs*)

shows \exists *as bs*. *zs* = *as@bs* \wedge *ct-list-eq* *ys* (to-ct-list *bs*) \wedge *ct-suffix* *xs* (to-ct-list *as*)

⟨*proof*⟩

lemma *ct-suffix-cons-ct-list-single*:

assumes *ct-suffix* (*xs@[y]*) (to-ct-list *zs*)

shows \exists *as b*. *zs* = *as@[b]* \wedge *ct-eq* *y* (TSome *b*) \wedge *ct-suffix* *xs* (to-ct-list *as*)

⟨*proof*⟩

lemma *ct-suffix-cons-ct-list1*:

assumes *ct-suffix* (*xs@(to-ct-list ys)*) (to-ct-list *zs*)

shows \exists *as*. *zs* = *as@ys* \wedge *ct-suffix* *xs* (to-ct-list *as*)

⟨*proof*⟩

lemma *ct-suffix-cons2*:

assumes *ct-suffix* (*xs*) (*ys@zs*)

length xs = *length zs*

shows *ct-list-eq xs zs*

⟨*proof*⟩

lemma *ct-suffix-imp-ct-list-eq*:

assumes *ct-suffix xs ys*

shows *ct-list-eq* (drop (length *ys* - length *xs*) *ys*) *xs*

⟨*proof*⟩

lemma *ct-suffix-extend-ct-list-eq*:

assumes *ct-suffix xs ys*

ct-list-eq xs' ys'

shows *ct-suffix* (*xs@xs'*) (*ys@ys'*)

<proof>

lemma *ct-suffix-extend-any1*:
assumes *ct-suffix xs ys*
 $\text{length } xs < \text{length } ys$
shows *ct-suffix (TAny#xs) ys*
<proof>

lemma *ct-suffix-singleton-any*: *ct-suffix [TAny] [t]*
<proof>

lemma *ct-suffix-cons-it*: *ct-suffix xs (xs'@xs)*
<proof>

lemma *ct-suffix-singleton*:
assumes $\text{length } cts > 0$
shows *ct-suffix [TAny] cts*
<proof>

lemma *ct-suffix-less*:
assumes *ct-suffix (xs@xs') ys*
shows *ct-suffix xs' ys*
<proof>

lemma *ct-suffix-unfold-one*: *ct-suffix (xs@[x]) (ys@[y]) = ((ct-eq x y) \wedge ct-suffix xs ys)*
<proof>

lemma *ct-suffix-shared*:
assumes *ct-suffix cts (to-ct-list ts)*
 $\text{ct-suffix } cts' (\text{to-ct-list } ts)$
shows $\text{ct-suffix } cts \ cts' \vee \text{ct-suffix } cts' \ cts$
<proof>

fun *checker-type-suffix::checker-type \Rightarrow checker-type \Rightarrow bool* **where**
 $\text{checker-type-suffix } (\text{Type } ts) (\text{Type } ts') = \text{suffix } ts \ ts'$
 $\mid \text{checker-type-suffix } (\text{Type } ts) (\text{TopType } cts) = \text{ct-suffix } (\text{to-ct-list } ts) \ cts$
 $\mid \text{checker-type-suffix } (\text{TopType } cts) (\text{Type } ts) = \text{ct-suffix } cts (\text{to-ct-list } ts)$
 $\mid \text{checker-type-suffix } - = \text{False}$

fun *consume :: checker-type \Rightarrow ct list \Rightarrow checker-type* **where**
 $\text{consume } (\text{Type } ts) \ cons = (\text{if } \text{ct-suffix } cons (\text{to-ct-list } ts)$
 $\quad \text{then } \text{Type } (\text{take } (\text{length } ts - \text{length } cons) \ ts)$
 $\quad \text{else } \text{Bot})$
 $\mid \text{consume } (\text{TopType } cts) \ cons = (\text{if } \text{ct-suffix } cons \ cts$
 $\quad \text{then } \text{TopType } (\text{take } (\text{length } cts - \text{length } cons) \ cts)$
 $\quad \text{else } (\text{if } \text{ct-suffix } cts \ cons$
 $\quad \quad \text{then } \text{TopType } []$
 $\quad \quad \text{else } \text{Bot}))$

$$| \text{consume} - - = Bot$$

```

fun produce :: checker-type  $\Rightarrow$  checker-type  $\Rightarrow$  checker-type where
  produce (TopType ts) (Type ts') = TopType (ts@(to-ct-list ts'))
| produce (Type ts) (Type ts') = Type (ts@ts')
| produce (Type ts') (TopType ts) = TopType ts
| produce (TopType ts') (TopType ts) = TopType ts
| produce - = Bot

```

$$\text{fun type-update} :: \text{checker-type} \Rightarrow \text{ct list} \Rightarrow \text{checker-type} \Rightarrow \text{checker-type} \text{ where}$$

$$\text{type-update curr-type cons prods} = \text{produce (consume curr-type cons) prods}$$
$$\begin{array}{l} \text{fun } \textit{ens-sec-ct} :: \textit{sec} \Rightarrow \textit{ct} \Rightarrow \textit{ct} \text{ where} \\ \quad \textit{ens-sec-ct} \textit{ Secret } T\textit{Any} = T\textit{Secret} \\ \quad | \textit{ens-sec-ct} \textit{ - ct} = \textit{ct} \end{array}$$
$$\begin{aligned}
\text{fun } \text{select-return-top} :: \text{sec} \Rightarrow \text{ct list} \Rightarrow \text{ct} \Rightarrow \text{ct} \Rightarrow \text{checker-type } \mathbf{where} \\
\text{select-return-top sec ts ct1 TAny} = & (\text{if } (\text{sec} = \text{Secret} \longrightarrow \text{can-secret-ct ct1}) \\
& \text{then TopType } ((\text{take } (\text{length ts} - 3) \text{ ts}) @ \\
& [\text{ens-sec-ct sec ct1}]) \\
& \text{else Bot}) \\
\text{select-return-top sec ts TAny ct2} = & (\text{if } (\text{sec} = \text{Secret} \longrightarrow \text{can-secret-ct ct2}) \\
& \text{then TopType } ((\text{take } (\text{length ts} - 3) \text{ ts}) @ \\
& [\text{ens-sec-ct sec ct2}]) \\
& \text{else Bot}) \\
\text{select-return-top sec ts ct1 TSecret} = & (\text{if } (\text{can-secret-ct ct1}) \\
& \text{then TopType } ((\text{take } (\text{length ts} - 3) \text{ ts}) @ \\
& [\text{ens-sec-ct sec ct1}]) \\
& \text{else Bot}) \\
\text{select-return-top sec ts TSecret ct2} = & (\text{if } (\text{can-secret-ct ct2}) \\
& \text{then TopType } ((\text{take } (\text{length ts} - 3) \text{ ts}) @ [\text{ct2}]) \\
& \text{else Bot}) \\
\text{select-return-top sec ts (TSome t1) (TSome t2)} = & (\text{if } (\text{t1} = \text{t2} \wedge (\text{sec} = \text{Secret} \\
& \longrightarrow \text{is-secret-t t1})) \\
& \text{then } (\text{TopType } ((\text{take } (\text{length ts} - 3) \text{ ts}) \\
& @ [\text{TSome t1}])) \\
& \text{else Bot})
\end{aligned}$$

lemma *select-return-top-ens-sec-ct*:

assumes *select-return-top sec ts ct1 ct2 = ct'*
shows $ct' = Bot \vee ct' = TopType ((take (length ts - 3) ts) @ [ens-sec-ct sec ct1]) \vee ct' = TopType ((take (length ts - 3) ts) @ [ens-sec-ct sec ct2])$
<proof>

lemma *select-return-top-ens-sec-ct-not-bot*:

assumes *select-return-top sec ts ct1 ct2 = ct'*
ct' ≠ Bot
shows *ct' = TopType ((take (length ts - 3) ts) @ [ens-sec-ct sec ct1]) ∨ ct' = TopType ((take (length ts - 3) ts) @ [ens-sec-ct sec ct2])*

$\langle \text{proof} \rangle$

fun *type-update-select* :: *sec* \Rightarrow *checker-type* \Rightarrow *checker-type* **where**
 type-update-select *sec* (*Type* *ts*) = (if (*length* *ts* \geq 3 \wedge (*ts*!(*length* *ts*-2)) =
 (*ts*!(*length* *ts*-3)) \wedge (*sec* = *Secret* \longrightarrow *is-secret-t* (*ts*!(*length* *ts*-2))))
 then *consume* (*Type* *ts*) [*TAny*, *TSome* (*T-i32* *sec*)]
 else *Bot*)
 | *type-update-select* *sec* (*TopType* *ts*) = (case *length* *ts* of
 0 \Rightarrow *TopType* [*sec-ct* *sec*]
 | *Suc* 0 \Rightarrow *type-update* (*TopType* *ts*) [*TSome*
 (*T-i32* *sec*)] (*TopType* [*sec-ct* *sec*])
 | *Suc* (*Suc* 0) \Rightarrow *type-update* (*TopType* *ts*) [*sec-ct*
 sec, *TSome* (*T-i32* *sec*)] (*TopType* [*ens-sec-ct* *sec* (*ts*!(*length* *ts*-2))])
 | - \Rightarrow *type-update* (*TopType* *ts*) [*sec-ct* *sec*, *sec-ct*
 sec, *TSome* (*T-i32* *sec*)]
 (*select-return-top* *sec* *ts* (*ts*!(*length*
 ts-2)) (*ts*!(*length* *ts*-3))))
 | *type-update-select* - = *Bot*

fun *c-types-agree* :: *checker-type* \Rightarrow *t list* \Rightarrow *bool* **where**
 c-types-agree (*Type* *ts*) *ts'* = (*ts* = *ts'*)
 | *c-types-agree* (*TopType* *ts*) *ts'* = *ct-suffix* *ts* (*to-ct-list* *ts'*)
 | *c-types-agree* *Bot* - = *False*

lemma *select-return-top-sec*:

assumes *select-return-top* *sec* *ts* *ct1* *ct2* = *ts'*
 ts' \neq *Bot*
 shows (*sec* = *Secret* \longrightarrow *can-secret-ct* *ct1*) \wedge (*sec* = *Secret* \longrightarrow *can-secret-ct*
 ct2)
 $\langle \text{proof} \rangle$

lemma *produce-not-bot*:

assumes *produce* *a* *b* = *c*
 c \neq *Bot*
 shows *a* \neq *Bot* *b* \neq *Bot*
 $\langle \text{proof} \rangle$

lemma *consume-type*:

assumes *consume* (*Type* *ts*) *ts'* = *c-t*
 c-t \neq *Bot*
 shows \exists *ts''*. *ct-list-eq* (*to-ct-list* *ts*) ((*to-ct-list* *ts''*)@*ts'*) \wedge *c-t* = *Type* *ts''*
 $\langle \text{proof} \rangle$

lemma *consume-top-geq*:

assumes *consume* (*TopType* *ts*) *ts'* = *c-t*
 length *ts* \geq *length* *ts'*
 c-t \neq *Bot*
 shows (\exists *as* *bs*. *ts* = *as*@*bs* \wedge *ct-list-eq* *bs* *ts'* \wedge *c-t* = *TopType* *as*)
 $\langle \text{proof} \rangle$

lemma *consume-top-leq*:
assumes *consume* (*TopType* *ts*) *ts'* = *c-t*
 $\text{length } ts \leq \text{length } ts'$
 $c-t \neq \text{Bot}$
shows $c-t = \text{TopType } []$
 $\langle \text{proof} \rangle$

lemma *consume-type-type*:
assumes *consume* *xs* *cons* = (*Type* *t-int*)
shows $\exists tn. xs = \text{Type } tn$
 $\langle \text{proof} \rangle$

lemma *produce-type-type*:
assumes *produce* *xs* *cons* = (*Type* *tm*)
shows $\exists tn. xs = \text{Type } tn$
 $\langle \text{proof} \rangle$

lemma *consume-weaken-type*:
assumes *consume* (*Type* *tn*) *cons* = (*Type* *t-int*)
shows *consume* (*Type* (*ts@tn*)) *cons* = (*Type* (*ts@t-int*))
 $\langle \text{proof} \rangle$

lemma *produce-weaken-type*:
assumes *produce* (*Type* *tn*) *cons* = (*Type* *tm*)
shows *produce* (*Type* (*ts@tn*)) *cons* = (*Type* (*ts@tm*))
 $\langle \text{proof} \rangle$

lemma *produce-nil*: *produce* *ts* (*Type* $[]$) = *ts*
 $\langle \text{proof} \rangle$

lemma *c-types-agree-id*: *c-types-agree* (*Type* *ts*) *ts*
 $\langle \text{proof} \rangle$

lemma *c-types-agree-top1*: *c-types-agree* (*TopType* $[]$) *ts*
 $\langle \text{proof} \rangle$

lemma *c-types-agree-top2*:
assumes *ct-list-eq* *ts* (*to-ct-list* *ts''*)
shows *c-types-agree* (*TopType* *ts*) (*ts'@ts''*)
 $\langle \text{proof} \rangle$

lemma *c-types-agree-imp-ct-list-eq*:
assumes *c-types-agree* (*TopType* *cts*) *ts*
shows $\exists ts' ts''. (ts = ts'@ts'') \wedge \text{ct-list-eq } cts (\text{to-ct-list } ts'')$
 $\langle \text{proof} \rangle$

lemma *c-types-agree-not-bot-exists*:
assumes $ts \neq \text{Bot}$

shows $\exists ts-c. c\text{-types-agree } ts \ ts-c$
 $\langle proof \rangle$

lemma *consume-c-types-agree*:
assumes $consume \ (Type \ ts) \ cts = (Type \ ts')$
 $c\text{-types-agree } ctn \ ts$
shows $\exists c-t'. consume \ ctn \ cts = c-t' \wedge c\text{-types-agree } c-t' \ ts'$
 $\langle proof \rangle$

lemma *type-update-type*:
assumes $type\text{-update} \ (Type \ ts) \ (to\text{-ct-list } cons) \ prods = ts'$
 $ts' \neq Bot$
shows $(ts' = prods \wedge (\exists ts-c. prods = (TopType \ ts-c)))$
 $\vee (\exists ts-a \ ts-b. prods = Type \ ts-a \wedge ts = ts-b@cons \wedge ts' = Type$
 $(ts-b@ts-a))$
 $\langle proof \rangle$

lemma *type-update-empty*: $type\text{-update } ts \ cons \ (Type \ []) = consume \ ts \ cons$
 $\langle proof \rangle$

lemma *type-update-top-top*:
assumes $type\text{-update} \ (TopType \ ts) \ (to\text{-ct-list } cons) \ (Type \ prods) = (TopType \ ts')$
 $c\text{-types-agree } (TopType \ ts') \ t-ag$
shows $ct\text{-suffix} \ (to\text{-ct-list } prods) \ ts'$
 $\exists t-ag'. t-ag = t-ag'@prods \wedge c\text{-types-agree } (TopType \ ts) \ (t-ag'@cons)$
 $\langle proof \rangle$

lemma *type-update-select-length0*:
assumes $type\text{-update-select } sec \ (TopType \ cts) = tm$
 $length \ cts = 0$
 $tm \neq Bot$
shows $tm = TopType \ [sec\text{-ct } sec]$
 $\langle proof \rangle$

lemma *type-update-select-length1*:
assumes $type\text{-update-select } sec \ (TopType \ cts) = tm$
 $length \ cts = 1$
 $tm \neq Bot$
shows $ct\text{-list-eq } cts \ [TSome \ (T-i32 \ sec)]$
 $tm = TopType \ [sec\text{-ct } sec]$
 $\langle proof \rangle$

lemma *type-update-select-length2*:
assumes $type\text{-update-select } sec \ (TopType \ cts) = tm$
 $length \ cts = 2$
 $tm \neq Bot$
shows $\exists t1 \ t2. cts = [t1, t2] \wedge ct\text{-eq } t2 \ (TSome \ (T-i32 \ sec)) \wedge ct\text{-eq } t1 \ (sec\text{-ct})$

$sec) \wedge tm = TopType [ens\text{-}sec\text{-}ct\ sec\ t1]$
 $\langle proof \rangle$

lemma *type-update-select-length3*:

assumes *type-update-select* *sec* (*TopType* *cts*) = (*TopType* *ctm*)
 $length\ cts \geq 3$
shows $\exists cts' ct1\ ct2\ ct3. cts = cts'@[ct1, ct2, ct3] \wedge ct\text{-}eq\ ct3\ (TSome\ (T\text{-}i32\ sec))$
 $\wedge ct\text{-}eq\ ct1\ (sec\text{-}ct\ sec) \wedge ct\text{-}eq\ ct2\ (sec\text{-}ct\ sec)$
 $\langle proof \rangle$

lemma *type-update-select-type-length3*:

assumes *type-update-select* *sec* (*Type* *tn*) = (*Type* *tm*)
shows $\exists t\ ts'. tn = ts'@[t, t, (T\text{-}i32\ sec)]$
 $\langle proof \rangle$

lemma *select-return-top-exists*:

assumes *select-return-top* *sec* *cts* *c1* *c2* = *ctm*
 $ctm \neq Bot$
shows $\exists xs. ctm = TopType\ xs$
 $\langle proof \rangle$

lemma *type-update-select-top-exists*:

assumes *type-update-select* *sec* *xs* = (*TopType* *tm*)
shows $\exists tn. xs = TopType\ tn$
 $\langle proof \rangle$

lemma *select-return-top-ct-eq*:

assumes *select-return-top* *sec* *cts* *c1* *c2* = *TopType* *ctm*
 $length\ cts \geq 3$
 $c\text{-types-agree}\ (TopType\ ctm)\ cm$
shows $\exists c'\ cm'. cm = cm'@[c']$
 $\wedge ct\text{-}suffix\ (take\ (length\ cts - 3)\ cts)\ (to\text{-}ct\text{-}list\ cm')$
 $\wedge ct\text{-}eq\ c1\ (TSome\ c')$
 $\wedge ct\text{-}eq\ c2\ (TSome\ c')$
 $\langle proof \rangle$

lemma *ens-sec-ct-imp-ct-eq*:

assumes *ct-eq* (*ens-sec-ct* *sec* *ct*) *ct'*
shows *ct-eq* *ct* *ct'*
 $\langle proof \rangle$

lemma *ens-sec-ct-imp-ct-eq-sec*:

assumes *ct-eq* *ct* *ct'*
 $sec = Secret \longrightarrow can\text{-}secret\text{-}ct\ ct'$
shows *ct-eq* (*ens-sec-ct* *sec* *ct*) *ct'*
 $\langle proof \rangle$

lemma *ct-eq-TSecret-imp-is-secret-t*:
assumes *ct-eq ct1 TSecret*
ct-eq (ens-sec-ct Secret ct1) (TSome t'')
shows *is-secret-t t''*
<proof>

lemma *ct-eq-TSome-imp-ct-eq-TSecret*:
assumes *ct-eq ct (TSome t)*
(sec = Secret \longrightarrow is-secret-t t)
shows *ct-eq ct (sec-ct sec)*
<proof>

lemma *select-return-top-secret*:
assumes *select-return-top Secret ts ct1 ct2 = ct3*
ct3 \neq Bot
c-types-agree ct3 t3
shows *is-secret-t (last t3)*
<proof>

end

9 Executable Type Checker

theory *Wasm-Checker* **imports** *Wasm-Checker-Types* **begin**

fun *convert-cond* :: *t \Rightarrow t \Rightarrow sx option \Rightarrow bool* **where**
convert-cond t1 t2 sx = ((t1 \neq t2) \wedge (t-sec t1 = t-sec t2) \wedge (sx = None) =
((is-float-t t1 \wedge is-float-t t2)
 \vee (is-int-t t1 \wedge is-int-t t2 \wedge (t-length
t1 < t-length t2))))

fun *same-lab-h* :: *nat list \Rightarrow (t list) list \Rightarrow t list \Rightarrow (t list) option* **where**
same-lab-h [] - ts = Some ts
| same-lab-h (i#is) lab-c ts = (if i \geq length lab-c
then None
else (if lab-c!i = ts
then same-lab-h is lab-c (lab-c!i)
else None))

fun *same-lab* :: *nat list \Rightarrow (t list) list \Rightarrow (t list) option* **where**
same-lab [] lab-c = None
| same-lab (i#is) lab-c = (if i \geq length lab-c
then None
else same-lab-h is lab-c (lab-c!i))

lemma *same-lab-h-conv-list-all*:
assumes *same-lab-h ils ls ts' = Some ts*
shows *list-all ($\lambda i. i < \text{length } ls \wedge ls!i = ts$) ils \wedge ts' = ts*

<proof>

lemma *same-lab-conv-list-all*:

assumes *same-lab* *ils* *ls* = *Some* *ts*

shows *list-all* ($\lambda i. i < \text{length } ls \wedge ls[i] = ts$) *ils*

<proof>

lemma *list-all-conv-same-lab-h*:

assumes *list-all* ($\lambda i. i < \text{length } ls \wedge ls[i] = ts$) *ils*

shows *same-lab-h* *ils* *ls* *ts* = *Some* *ts*

<proof>

lemma *list-all-conv-same-lab*:

assumes *list-all* ($\lambda i. i < \text{length } ls \wedge ls[i] = ts$) (*is@*[*i*])

shows *same-lab* (*is@*[*i*]) *ls* = *Some* *ts*

<proof>

fun *b-e-type-checker* :: *t-context* \Rightarrow *b-e list* \Rightarrow *tf* \Rightarrow *bool*

and *check* :: *t-context* \Rightarrow *b-e list* \Rightarrow *checker-type* \Rightarrow *checker-type*

and *check-single* :: *t-context* \Rightarrow *b-e* \Rightarrow *checker-type* \Rightarrow *checker-type* **where**

b-e-type-checker *C* *es* (*tn* \rightarrow *tm*) = *c-types-agree* (*check* *C* *es* (*Type* *tn*)) *tm*

| *check* *C* *es* *ts* = (case *es* of

 [] \Rightarrow *ts*

 | (*e#es*) \Rightarrow (case *ts* of

Bot \Rightarrow *Bot*

 | - \Rightarrow *check* *C* *es* (*check-single* *C* *e* *ts*)))

| *check-single* *C* (*C* *v*) *ts* = *type-update* *ts* [] (*Type* [*typeof* *v*])

| *check-single* *C* (*Unop-i* *t* -) *ts* = (if *is-int-t* *t*

 then *type-update* *ts* [*TSome* *t*] (*Type* [*t*])

 else *Bot*)

| *check-single* *C* (*Unop-f* *t* -) *ts* = (if *is-float-t* *t*

 then *type-update* *ts* [*TSome* *t*] (*Type* [*t*])

 else *Bot*)

| *check-single* *C* (*Binop-i* *t* *iop*) *ts* = (if *is-int-t* *t* \wedge (*is-secret-t* *t* \longrightarrow *safe-binop-i* *iop*)

 then *type-update* *ts* [*TSome* *t*, *TSome* *t*] (*Type* [*t*])

 else *Bot*)

| *check-single* *C* (*Binop-f* *t* -) *ts* = (if *is-float-t* *t*

 then *type-update* *ts* [*TSome* *t*, *TSome* *t*] (*Type* [*t*])

 else *Bot*)

| *check-single* *C* (*Testop* *t* -) *ts* = (if *is-int-t* *t*

 then *type-update* *ts* [*TSome* *t*] (*Type* [*T-i32* (*t-sec* *t*)])

 else *Bot*)

| *check-single* *C* (*Relop-i* *t* -) *ts* = (if *is-int-t* *t*

 then *type-update* *ts* [*TSome* *t*, *TSome* *t*] (*Type*

[*T-i32* (*t-sec* *t*)])

 else *Bot*)

$$| \text{check-single } \mathcal{C} \text{ (Relop-} f \text{ } t \text{ -)} \text{ } ts = \begin{aligned} & \text{(if is-float-} t \text{ } t \\ & \quad \text{then type-update } ts \text{ [TSome } t, \text{ TSome } t] \text{ (Type} \\ & \text{[T-i32 (t-sec } t\text{)])} \\ & \quad \text{else Bot)} \end{aligned}$$

$$| \text{check-single } \mathcal{C} \text{ (Cvtop } t1 \text{ Convert } t2 \text{ } sx) \text{ } ts = \begin{aligned} & \text{(if (convert-cond } t1 \text{ } t2 \text{ } sx) \\ & \quad \text{then type-update } ts \text{ [TSome } t2] \text{ (Type [} t1\text{])} \\ & \quad \text{else Bot)} \end{aligned}$$

$$| \text{check-single } \mathcal{C} \text{ (Cvtop } t1 \text{ Reinterpret } t2 \text{ } sx) \text{ } ts = \begin{aligned} & \text{(if ((} t1 \neq t2 \text{) } \wedge \text{ (t-sec } t1 = \text{t-sec} \\ & \text{ } t2 \text{) } \wedge \text{ t-length } t1 = \text{t-length } t2 \wedge sx = \text{None)} \\ & \quad \text{then type-update } ts \text{ [TSome } t2] \text{ (Type} \\ & \text{[} t1\text{])} \\ & \quad \text{else Bot)} \end{aligned}$$

$$| \text{check-single } \mathcal{C} \text{ (Cvtop } t1 \text{ Classify } t2 \text{ } sx) \text{ } ts = \begin{aligned} & \text{(if (is-int-} t \text{ } t2 \wedge \text{ is-public-} t \text{ } t2 \wedge \\ & \text{classify-} t \text{ } t2 = } t1 \wedge sx = \text{None)} \\ & \quad \text{then type-update } ts \text{ [TSome } t2] \text{ (Type [} t1\text{])} \\ & \quad \text{else Bot)} \end{aligned}$$

$$| \text{check-single } \mathcal{C} \text{ (Cvtop } t1 \text{ Declassify } t2 \text{ } sx) \text{ } ts = \begin{aligned} & \text{(if ((trust-} t \text{ } \mathcal{C}) = \text{Trusted } \wedge \text{ is-int-} t \\ & \text{ } t2 \wedge \text{ is-secret-} t \text{ } t2 \wedge \text{ declassify-} t \text{ } t2 = } t1 \wedge sx = \text{None)} \\ & \quad \text{then type-update } ts \text{ [TSome } t2] \text{ (Type [} t1\text{])} \\ & \quad \text{else Bot)} \end{aligned}$$

$$| \text{check-single } \mathcal{C} \text{ (Unreachable) } ts = \text{type-update } ts \text{ [] (TopType [])}$$

$$| \text{check-single } \mathcal{C} \text{ (Nop) } ts = ts$$

$$| \text{check-single } \mathcal{C} \text{ (Drop) } ts = \text{type-update } ts \text{ [TAny] (Type [])}$$

$$| \text{check-single } \mathcal{C} \text{ (Select } sec) \text{ } ts = \text{type-update-select } sec \text{ } ts$$

$$| \text{check-single } \mathcal{C} \text{ (Block (tn -> tm) es) } ts = \begin{aligned} & \text{(if (b-e-type-checker (C[label := ([tm]} \\ & \text{ @ (label C))})) es (tn -> tm))} \\ & \quad \text{then type-update } ts \text{ (to-ct-list tn) (Type tm)} \\ & \quad \text{else Bot)} \end{aligned}$$

$$| \text{check-single } \mathcal{C} \text{ (Loop (tn -> tm) es) } ts = \begin{aligned} & \text{(if (b-e-type-checker (C[label := ([tn]} \\ & \text{ @ (label C))})) es (tn -> tm))} \\ & \quad \text{then type-update } ts \text{ (to-ct-list tn) (Type tm)} \\ & \quad \text{else Bot)} \end{aligned}$$

$$| \text{check-single } \mathcal{C} \text{ (If (tn -> tm) es1 es2) } ts = \begin{aligned} & \text{(if (b-e-type-checker (C[label := ([tm]} \\ & \text{ @ (label C))})) es1 (tn -> tm)} \\ & \quad \wedge \text{ b-e-type-checker (C[label := ([tm]} @ \\ & \text{ (label C))})) es2 (tn -> tm))} \\ & \quad \text{then type-update } ts \text{ (to-ct-list (tn@[T-i32} \\ & \text{Public])) (Type tm)} \\ & \quad \text{else Bot)} \end{aligned}$$

$$| \text{check-single } \mathcal{C} \text{ (Br } i) \text{ } ts = \text{(if } i < \text{length (label } \mathcal{C})$$

$$\begin{aligned} & \text{then type-update } ts \text{ (to-ct-list } ((\text{label } \mathcal{C})!i)) \text{ (TopType } [])) \\ & \text{else Bot} \end{aligned}$$

$$\begin{aligned} | \text{ check-single } \mathcal{C} \text{ (Br-if } i) \text{ } ts = & (\text{if } i < \text{length } (\text{label } \mathcal{C}) \\ & \text{then type-update } ts \text{ (to-ct-list } ((\text{label } \mathcal{C})!i \text{ @ } [T-i32 \\ \text{Public}])) \text{ (Type } ((\text{label } \mathcal{C})!i)) \\ & \text{else Bot}) \end{aligned}$$

$$\begin{aligned} | \text{ check-single } \mathcal{C} \text{ (Br-table is } i) \text{ } ts = & (\text{case } (\text{same-lab } (is@[i]) \text{ (label } \mathcal{C})) \text{ of} \\ & \text{None} \Rightarrow \text{Bot} \\ | \text{Some } tls \Rightarrow & \text{type-update } ts \text{ (to-ct-list } (tls \text{ @ } [T-i32 \\ \text{Public}])) \text{ (TopType } [])) \end{aligned}$$

$$\begin{aligned} | \text{ check-single } \mathcal{C} \text{ (Return) } ts = & (\text{case } (\text{return } \mathcal{C}) \text{ of} \\ & \text{None} \Rightarrow \text{Bot} \\ | \text{Some } tls \Rightarrow & \text{type-update } ts \text{ (to-ct-list } tls) \text{ (TopType } [])) \end{aligned}$$

$$\begin{aligned} | \text{ check-single } \mathcal{C} \text{ (Call } i) \text{ } ts = & (\text{if } i < \text{length } (\text{func-t } \mathcal{C}) \\ & \text{then } (\text{case } ((\text{func-t } \mathcal{C})!i) \text{ of} \\ & (tr, (tn \rightarrow tm)) \Rightarrow \text{if } (\text{trust-compat } (\text{trust-t } \mathcal{C}) \text{ } tr) \\ & \text{then type-update } ts \text{ (to-ct-list} \\ & tn) \text{ (Type } tm) \\ & \text{else Bot}) \\ & \text{else Bot}) \end{aligned}$$

$$\begin{aligned} | \text{ check-single } \mathcal{C} \text{ (Call-indirect } i) \text{ } ts = & (\text{if } (\text{table } \mathcal{C}) \neq \text{None} \wedge i < \text{length } (\text{types-t } \mathcal{C}) \\ & \text{then } (\text{case } ((\text{types-t } \mathcal{C})!i) \text{ of} \\ & (tr, (tn \rightarrow tm)) \Rightarrow \text{if } (\text{trust-compat} \\ & (\text{trust-t } \mathcal{C}) \text{ } tr) \\ & \text{then type-update } ts \\ & \text{(to-ct-list } (tn@[T-i32 \text{ Public}])) \text{ (Type } tm) \\ & \text{else Bot}) \\ & \text{else Bot}) \end{aligned}$$

$$\begin{aligned} | \text{ check-single } \mathcal{C} \text{ (Get-local } i) \text{ } ts = & (\text{if } i < \text{length } (\text{local } \mathcal{C}) \\ & \text{then type-update } ts \text{ [] (Type } [(\text{local } \mathcal{C})!i]) \\ & \text{else Bot}) \end{aligned}$$

$$\begin{aligned} | \text{ check-single } \mathcal{C} \text{ (Set-local } i) \text{ } ts = & (\text{if } i < \text{length } (\text{local } \mathcal{C}) \\ & \text{then type-update } ts \text{ [TSome } ((\text{local } \mathcal{C})!i)] \text{ (Type } []) \\ & \text{else Bot}) \end{aligned}$$

$$\begin{aligned} | \text{ check-single } \mathcal{C} \text{ (Tee-local } i) \text{ } ts = & (\text{if } i < \text{length } (\text{local } \mathcal{C}) \\ & \text{then type-update } ts \text{ [TSome } ((\text{local } \mathcal{C})!i)] \text{ (Type } [(\text{local } \mathcal{C})!i]) \\ & \text{else Bot}) \end{aligned}$$

$$\begin{aligned} | \text{ check-single } \mathcal{C} \text{ (Get-global } i) \text{ } ts = & (\text{if } i < \text{length } (\text{global } \mathcal{C}) \end{aligned}$$

```

      then type-update ts [] (Type [tg-t ((global C)!i)])
      else Bot)

| check-single C (Set-global i) ts = (if i < length (global C) ∧ is-mut (global C ! i)
      then type-update ts [TSome (tg-t ((global C)!i))])
(Type [])
      else Bot)

| check-single C (Load t tp-sx a off) ts =
      (case (memory C) of
        Some (m, sec) ⇒
          if t-sec t = sec ∧ load-store-t-bounds a (option-proj
tp-sx) t
            then type-update ts [TSome (T-i32 Public)] (Type [t])
            else Bot
        | None ⇒ Bot)

| check-single C (Store t tp a off) ts =
      (case (memory C) of
        Some (m, sec) ⇒
          if t-sec t = sec ∧ load-store-t-bounds a tp t
            then type-update ts [TSome (T-i32 Public), TSome t]
(Type [])
            else Bot
        | None ⇒ Bot)

| check-single C Current-memory ts = (if (memory C) ≠ None
      then type-update ts [] (Type [T-i32 Public])
      else Bot)

| check-single C Grow-memory ts = (if (memory C) ≠ None
      then type-update ts [TSome (T-i32 Public)] (Type
[T-i32 Public])
      else Bot)

end

```

10 Correctness of Type Checker

theory *Wasm-Checker-Properties* **imports** *Wasm-Checker* *Wasm-Properties* **begin**

10.1 Soundness

lemma *b-e-check-single-type-sound*:

```

assumes type-update (Type x1) (to-ct-list t-in) (Type t-out) = Type x2
          c-types-agree (Type x2) tm
          C ⊢ [e] : (t-in -> t-out)
shows ∃ tn. c-types-agree (Type x1) tn ∧ C ⊢ [e] : (tn -> tm)

```

$\langle \text{proof} \rangle$

lemma *b-e-check-single-top-sound*:

assumes $\text{type-update } (\text{TopType } x1) (\text{to-ct-list } t\text{-in}) (\text{Type } t\text{-out}) = \text{TopType } x2$
 $c\text{-types-agree } (\text{TopType } x2) \text{ } tm$
 $\mathcal{C} \vdash [e] : (t\text{-in} \rightarrow t\text{-out})$
shows $\exists tn. c\text{-types-agree } (\text{TopType } x1) \text{ } tn \wedge \mathcal{C} \vdash [e] : (tn \rightarrow tm)$
 $\langle \text{proof} \rangle$

lemma *b-e-check-single-top-not-bot-sound*:

assumes $\text{type-update } ts (\text{to-ct-list } t\text{-in}) (\text{TopType } []) = ts'$
 $ts \neq \text{Bot}$
 $ts' \neq \text{Bot}$
shows $\exists tn. c\text{-types-agree } ts \text{ } tn \wedge \text{suffix } t\text{-in } tn$
 $\langle \text{proof} \rangle$

lemma *b-e-check-single-type-not-bot-sound*:

assumes $\text{type-update } ts (\text{to-ct-list } t\text{-in}) (\text{Type } t\text{-out}) = ts'$
 $ts \neq \text{Bot}$
 $ts' \neq \text{Bot}$
 $c\text{-types-agree } ts' \text{ } tm$
 $\mathcal{C} \vdash [e] : (t\text{-in} \rightarrow t\text{-out})$
shows $\exists tn. c\text{-types-agree } ts \text{ } tn \wedge \mathcal{C} \vdash [e] : (tn \rightarrow tm)$
 $\langle \text{proof} \rangle$

lemma *b-e-check-single-sound-unop-testop-cvtop*:

assumes $\text{check-single } \mathcal{C} \text{ } e \text{ } tn' = tm'$
 $((e = (\text{Unop-i } t \text{ } uv) \vee e = (\text{Testop } t \text{ } uv)) \wedge \text{is-int-t } t)$
 $\vee (e = (\text{Unop-f } t \text{ } uv) \wedge \text{is-float-t } t)$
 $\vee (e = (\text{Cvtop } t1 \text{ } \text{Convert } t \text{ } sx) \wedge \text{convert-cond } t1 \text{ } t \text{ } sx)$
 $\vee (e = (\text{Cvtop } t1 \text{ } \text{Reinterpret } t \text{ } sx) \wedge ((t1 \neq t) \wedge (t\text{-sec } t1 = t\text{-sec } t) \wedge$
 $t\text{-length } t1 = t\text{-length } t \wedge sx = \text{None}))$
 $\vee (e = (\text{Cvtop } t1 \text{ } \text{Classify } t \text{ } sx) \wedge (\text{is-int-t } t \wedge \text{is-public-t } t \wedge \text{classify-t } t$
 $= t1 \wedge sx = \text{None}))$
 $\vee (e = (\text{Cvtop } t1 \text{ } \text{Declassify } t \text{ } sx) \wedge ((\text{trust-t } \mathcal{C}) = \text{Trusted} \wedge \text{is-int-t } t \wedge$
 $\text{is-secret-t } t \wedge \text{declassify-t } t = t1 \wedge sx = \text{None}))$
 $c\text{-types-agree } tm' \text{ } tm$
 $tn' \neq \text{Bot}$
 $tm' \neq \text{Bot}$
shows $\exists tn. c\text{-types-agree } tn' \text{ } tn \wedge \mathcal{C} \vdash [e] : (tn \rightarrow tm)$
 $\langle \text{proof} \rangle$

lemma *b-e-check-single-sound-binop-relop*:

assumes $\text{check-single } \mathcal{C} \text{ } e \text{ } tn' = tm'$
 $((e = \text{Binop-i } t \text{ } iop \wedge \text{is-int-t } t \wedge (\text{is-secret-t } t \rightarrow \text{safe-binop-i } iop))$
 $\vee (e = \text{Binop-f } t \text{ } fop \wedge \text{is-float-t } t)$
 $\vee (e = \text{Relop-i } t \text{ } irop \wedge \text{is-int-t } t)$
 $\vee (e = \text{Relop-f } t \text{ } frop \wedge \text{is-float-t } t))$

$c\text{-types-agree } tm' \text{ } tm$
 $tn' \neq Bot$
 $tm' \neq Bot$
shows $\exists tn. c\text{-types-agree } tn' \text{ } tn \wedge \mathcal{C} \vdash [e] : (tn \rightarrow tm)$
 $\langle proof \rangle$

lemma *b-e-type-checker-sound*:
assumes *b-e-type-checker* \mathcal{C} es $(tn \rightarrow tm)$
shows $\mathcal{C} \vdash es : (tn \rightarrow tm)$
 $\langle proof \rangle$

10.2 Completeness

lemma *check-single-imp*:
assumes *check-single* \mathcal{C} e $ctn = ctm$
 $ctm \neq Bot$
shows *check-single* \mathcal{C} $e = id$
 $\vee (\exists sec. \text{check-single } \mathcal{C} \text{ } e = (\lambda ctn. \text{type-update-select } sec \text{ } ctn))$
 $\vee (\exists cons \text{ } prods. (\text{check-single } \mathcal{C} \text{ } e = (\lambda ctn. \text{type-update } ctn \text{ } cons \text{ } prods)))$
 $\langle proof \rangle$

lemma *check-equiv-fold*:
 $\text{check } \mathcal{C} \text{ } es \text{ } ts = \text{foldl } (\lambda ts \text{ } e. (\text{case } ts \text{ of } Bot \Rightarrow Bot \mid - \Rightarrow \text{check-single } \mathcal{C} \text{ } e \text{ } ts))$
 $ts \text{ } es$
 $\langle proof \rangle$

lemma *check-neq-bot-snoc*:
assumes *check* \mathcal{C} $(es@[e]) \text{ } ts \neq Bot$
shows *check* \mathcal{C} $es \text{ } ts \neq Bot$
 $\langle proof \rangle$

lemma *check-unfold-snoc*:
assumes *check* \mathcal{C} $es \text{ } ts \neq Bot$
shows *check* \mathcal{C} $(es@[e]) \text{ } ts = \text{check-single } \mathcal{C} \text{ } e (\text{check } \mathcal{C} \text{ } es \text{ } ts)$
 $\langle proof \rangle$

lemma *check-single-imp-weakening*:
assumes *check-single* \mathcal{C} e $(\text{Type } t1s) = ctm$
 $ctm \neq Bot$
 $c\text{-types-agree } ctn \text{ } t1s$
 $c\text{-types-agree } ctm \text{ } t2s$
shows $\exists ctm'. \text{check-single } \mathcal{C} \text{ } e \text{ } ctn = ctm' \wedge c\text{-types-agree } ctm' \text{ } t2s$
 $\langle proof \rangle$

lemma *b-e-type-checker-compose*:
assumes *b-e-type-checker* \mathcal{C} es $(t1s \rightarrow t2s)$
 $b\text{-e-type-checker } \mathcal{C} [e] (t2s \rightarrow t3s)$
shows *b-e-type-checker* \mathcal{C} $(es @ [e]) (t1s \rightarrow t3s)$
 $\langle proof \rangle$

lemma *b-e-check-single-type-type*:
 assumes *check-single* \mathcal{C} e $xs = (\text{Type } tm)$
 shows $\exists tn. xs = (\text{Type } tn)$
 $\langle proof \rangle$

lemma *b-e-check-single-weaken-type*:
 assumes *check-single* \mathcal{C} e $(\text{Type } tn) = (\text{Type } tm)$
 shows *check-single* \mathcal{C} e $(\text{Type } (ts@tn)) = \text{Type } (ts@tm)$
 $\langle proof \rangle$

lemma *b-e-check-single-weaken-top*:
 assumes *check-single* \mathcal{C} e $(\text{Type } tn) = \text{TopType } tm$
 shows *check-single* \mathcal{C} e $(\text{Type } (ts@tn)) = \text{TopType } tm$
 $\langle proof \rangle$

lemma *b-e-check-weaken-type*:
 assumes *check* \mathcal{C} es $(\text{Type } tn) = (\text{Type } tm)$
 shows *check* \mathcal{C} es $(\text{Type } (ts@tn)) = (\text{Type } (ts@tm))$
 $\langle proof \rangle$

lemma *check-bot*: *check* \mathcal{C} es $Bot = Bot$
 $\langle proof \rangle$

lemma *b-e-check-weaken-top*:
 assumes *check* \mathcal{C} es $(\text{Type } tn) = (\text{TopType } tm)$
 shows *check* \mathcal{C} es $(\text{Type } (ts@tn)) = (\text{TopType } tm)$
 $\langle proof \rangle$

lemma *b-e-type-checker-weaken*:
 assumes *b-e-type-checker* \mathcal{C} es $(t1s \rightarrow t2s)$
 shows *b-e-type-checker* \mathcal{C} es $(ts@t1s \rightarrow ts@t2s)$
 $\langle proof \rangle$

lemma *b-e-type-checker-complete*:
 assumes $\mathcal{C} \vdash es : (tn \rightarrow tm)$
 shows *b-e-type-checker* \mathcal{C} es $(tn \rightarrow tm)$
 $\langle proof \rangle$

theorem *b-e-typing-equiv-b-e-type-checker*:
 shows $(\mathcal{C} \vdash es : (tn \rightarrow tm)) = (\text{b-e-type-checker } \mathcal{C} \text{ } es \text{ } (tn \rightarrow tm))$
 $\langle proof \rangle$

end

11 Auxiliary Security Properties

theory *Wasm-Secret-Aux* **imports** *Wasm-Soundness HOL-Eisbach.Eisbach-Tools*
begin

lemma *memory-public-agree-imp-eq-length*:
assumes *memory-public-agree m m'*
shows $\text{mem-size } (\text{fst } m) = \text{mem-size } (\text{fst } m')$
 $\langle \text{proof} \rangle$

lemma *store-public-agree-smem-ind-eq*:
assumes *store-public-agree s s'*
shows $(\text{smem-ind } s \ i) = (\text{smem-ind } s' \ i)$
 $\langle \text{proof} \rangle$

lemma *store-public-agree-sfunc-eq*:
assumes *store-public-agree s s'*
shows $(\text{sfunc } s \ i \ j) = (\text{sfunc } s' \ i \ j)$
 $\langle \text{proof} \rangle$

lemma *store-public-agree-stab-eq*:
assumes *store-public-agree s s'*
shows $(\text{stab } s \ i \ j) = (\text{stab } s' \ i \ j)$
 $\langle \text{proof} \rangle$

lemma *store-public-agree-sglob-ind-eq*:
assumes *store-public-agree s s'*
 $(\text{sglob-ind } s \ i \ j) < \text{length } (\text{globs } s)$
shows $(\text{sglob-ind } s \ i \ j) = (\text{sglob-ind } s' \ i \ j)$
 $\langle \text{proof} \rangle$

lemma *store-public-agree-sglob-val-agree*:
assumes *store-public-agree s s'*
 $(\text{sglob-ind } s \ i \ j) < \text{length } (\text{globs } s)$
shows $\text{public-agree } (\text{sglob-val } s \ i \ j) (\text{sglob-val } s' \ i \ j)$
 $\langle \text{proof} \rangle$

lemma *store-public-agree-stypes-eq*:
assumes *store-public-agree s s'*
shows $(\text{stypes } s \ i \ j) = (\text{stypes } s' \ i \ j)$
 $\langle \text{proof} \rangle$

lemma *store-agree-imp-callcl-cond*:
assumes *store-public-agree s s'*
 $(\text{stab } s \ i \ (\text{nat-of-int } c) = \text{Some } cl \wedge \text{stypes } s \ i \ j \neq \text{cl-type } cl) \vee \text{stab } s \ i$
 $(\text{nat-of-int } c) = \text{None}$
shows $(\text{stab } s' \ i \ (\text{nat-of-int } c) = \text{Some } cl \wedge \text{stypes } s' \ i \ j \neq \text{cl-type } cl) \vee \text{stab } s' \ i$
 $(\text{nat-of-int } c) = \text{None}$
 $\langle \text{proof} \rangle$

lemma *public-agree-imp-typeof*:
assumes *public-agree v v'*
shows $\text{typeof } v = \text{typeof } v'$

<proof>

lemma *not-typeof-imp-no-public-agree:*

assumes *typeof v ≠ typeof v'*

shows $\neg \text{public-agree } v \ v'$

<proof>

lemma *publics-agree-imp-typeof:*

assumes *publics-agree vs vs'*

shows *map typeof vs = map typeof vs'*

<proof>

lemma *public-agree-imp-types-agree-insecure:*

assumes *types-agree-insecure t v*

public-agree v v'

shows *types-agree-insecure t v'*

<proof>

lemma *public-agree-imp-types-agree:*

assumes *types-agree t v*

public-agree v v'

shows *types-agree t v'*

<proof>

lemma *publics-agree-nil1:*

assumes *publics-agree [] vs*

shows *vs = []*

<proof>

lemma *publics-agree-nil2:*

assumes *publics-agree vs []*

shows *vs = []*

<proof>

lemma *public-agree-refl: public-agree v v*

<proof>

lemma *public-agree-public-i32:*

assumes *public-agree (ConstInt32 sec c) v*

shows $\exists c. v = (\text{ConstInt32 } \text{sec } c)$

<proof>

lemma *public-agree-public-i64:*

assumes *public-agree (ConstInt64 sec c) v*

shows $\exists c. v = (\text{ConstInt64 } \text{sec } c)$

<proof>

lemma *public-agree-public-f32:*

assumes *public-agree (ConstFloat32 c) v*

shows $\exists c. v = (\text{ConstFloat32 } c)$
 $\langle \text{proof} \rangle$

lemma *public-agree-public-f64*:
assumes *public-agree* $(\text{ConstFloat64 } c) v$
shows $\exists c. v = (\text{ConstFloat64 } c)$
 $\langle \text{proof} \rangle$

lemma *publics-agree-refl*: *publics-agree vs vs*
 $\langle \text{proof} \rangle$

lemma *publics-agree1*:
assumes *publics-agree* $[v] \text{ es}'$
shows $\exists v'. \text{es}' = [v']$
 $\langle \text{proof} \rangle$

lemma *publics-agree-secret1*:
assumes *publics-agree* $[v] \text{ es}'$
 $t\text{-sec } (\text{typeof } v) = \text{Public}$
shows $\text{es}' = [v]$
 $\langle \text{proof} \rangle$

lemma *publics-agree-public1*:
assumes *publics-agree* $[v] \text{ es}'$
 $t\text{-sec } (\text{typeof } v) = \text{Public}$
shows $\exists v'. \text{es}' = [v'] \wedge \text{public-agree } v v'$
 $\langle \text{proof} \rangle$

lemma *memories-public-agree-refl*: *memories-public-agree ms ms*
 $\langle \text{proof} \rangle$

lemma *globals-public-agree-refl*: *globals-public-agree gs gs*
 $\langle \text{proof} \rangle$

lemmas *expr-public-agree-refl* = *expr-public-agree.intros(1)*

lemma *exprs-public-agree-refl*: *exprs-public-agree es es*
 $\langle \text{proof} \rangle$

lemma *list-all2-symm*:
assumes *list-all2* $P \text{ xs ys}$
 $(\bigwedge x y. P \ x \ y \implies P \ y \ x)$
shows *list-all2* $P \text{ ys xs}$
 $\langle \text{proof} \rangle$

lemma *public-agree-symm*:
assumes *public-agree* $v v'$
shows *public-agree* $v' v$
 $\langle \text{proof} \rangle$

lemma *public-agree-trans*:
 assumes *public-agree v v'*
 public-agree v' v''
 shows *public-agree v v''*
 ⟨*proof*⟩

lemma *equivp-public-agree:equivp public-agree*
 ⟨*proof*⟩

lemma *publics-agree-trans*:
 assumes *publics-agree vs vs'*
 publics-agree vs' vs''
 shows *publics-agree vs vs''*
 ⟨*proof*⟩

lemma *memories-public-agree-symm*:
 assumes *memories-public-agree ms ms'*
 shows *memories-public-agree ms' ms*
 ⟨*proof*⟩

lemma *globals-public-agree-symm*:
 assumes *globals-public-agree gs gs'*
 shows *globals-public-agree gs' gs*
 ⟨*proof*⟩

lemma *transp-memory-public-agree:transp memory-public-agree*
 ⟨*proof*⟩

lemma *memories-public-agree-trans*:
 assumes *memories-public-agree ms ms'*
 memories-public-agree ms' ms''
 shows *memories-public-agree ms ms''*
 ⟨*proof*⟩

lemma *transp-global-public-agree:transp global-public-agree*
 ⟨*proof*⟩

lemma *globals-public-agree-trans*:
 assumes *globals-public-agree gs gs'*
 globals-public-agree gs' gs''
 shows *globals-public-agree gs gs''*
 ⟨*proof*⟩

lemma *equivp-memories-public-agree: equivp memories-public-agree*
 ⟨*proof*⟩

lemma *equivp-globals-public-agree: equivp globals-public-agree*
 ⟨*proof*⟩

lemma *list-all2-flip-args*:
assumes *list-all2* $(\lambda x y. P x y)$ *xs ys*
shows *list-all2* $(\lambda y x. P x y)$ *ys xs*
 $\langle proof \rangle$

lemma *publics-agree-symm*:
assumes *publics-agree* *vs vs'*
shows *publics-agree* *vs' vs*
 $\langle proof \rangle$

lemma *expr-public-agree-symm*:
assumes *expr-public-agree* *e e'*
shows *expr-public-agree* *e' e*
 $\langle proof \rangle$

lemma *exprs-public-agree-symm*:
assumes *exprs-public-agree* *es es'*
shows *exprs-public-agree* *es' es*
 $\langle proof \rangle$

lemma *store-public-agree-refl*: *store-public-agree s s*
 $\langle proof \rangle$

lemma *store-public-agree-symm*:
assumes *store-public-agree* *s s'*
shows *store-public-agree* *s' s*
 $\langle proof \rangle$

lemma *store-public-agree-trans*:
assumes *store-public-agree* *s s'*
store-public-agree s' s''
shows *store-public-agree* *s s''*
 $\langle proof \rangle$

lemma *expr-public-agree-imp-public-agree*:
assumes *expr-public-agree* $(\$C v)$ *e*
shows $\exists v'. e = (\$C v') \wedge$ *public-agree* *v v'*
 $\langle proof \rangle$

lemma *expr-public-agree-block*:
assumes *expr-public-agree* $(\$Block\ tf\ es)$ *les*
shows $\exists es'. les = (\$Block\ tf\ es') \wedge$ *exprs-public-agree* $(\$*es) (\$*es')$
 $\langle proof \rangle$

lemma *expr-public-agree-loop*:
assumes *expr-public-agree* $(\$Loop\ tf\ es)$ *les*
shows $\exists es'. les = (\$Loop\ tf\ es') \wedge$ *exprs-public-agree* $(\$*es) (\$*es')$
 $\langle proof \rangle$

lemma *expr-public-agree-if*:

assumes *expr-public-agree* ($\$If\ tf\ es1\ es2$) *les*
shows $\exists es1'\ es2'.\ les = (\$If\ tf\ es1'\ es2') \wedge exprs-public-agree\ (\$*es1)\ (\$*es1')$
 $\wedge exprs-public-agree\ (\$*es2)\ (\$*es2')$
 $\langle proof \rangle$

lemma *expr-public-agree-local*:

assumes *expr-public-agree* (*Local* *n i vs es*) *les*
shows $\exists vs'\ es'.\ les = (Local\ n\ i\ vs'\ es') \wedge publics-agree\ vs\ vs' \wedge exprs-public-agree\ es\ es'$
 $\langle proof \rangle$

lemma *expr-public-agree-label*:

assumes *expr-public-agree* (*Label* *n les es*) *e'*
shows $\exists les'\ es''.\ e' = (Label\ n\ les'\ es'') \wedge exprs-public-agree\ les\ les' \wedge exprs-public-agree\ es\ es''$
 $\langle proof \rangle$

lemmas *expr-public-agree-imp-expr-publics-agree* = *list.rel-intros*(2)[*OF* - *list.rel-intros*(1),
of expr-public-agree]

lemma *expr-public-agree-basic*:

assumes *expr-public-agree* ($\$b-e1$) *e2*
shows $\exists b-e2.\ e2 = \$b-e2$
 $\langle proof \rangle$

lemma *exprs-public-agree-imp-expr-public-agree*:

assumes *exprs-public-agree* [*e1*] [*e2*]
shows *expr-public-agree* *e1 e2*
 $\langle proof \rangle$

lemmas *public-agree-imp-expr-public-agree* = *expr-public-agree.intros*(2)

lemma *exprs-public-agree-imp-publics-agree-cons*:

assumes *exprs-public-agree* ($(\$C\ v)\#es$) *es'*
shows $\exists v'\ es''.\ es' = ((\$C\ v')\#es'') \wedge public-agree\ v\ v' \wedge exprs-public-agree\ es\ es''$
 $\langle proof \rangle$

lemma *exprs-public-agree-imp-publics-agree*:

assumes *exprs-public-agree* ($(\$*ves)\@[es]$) *es'*
shows $\exists ves'\ es''.\ es' = ((\$*ves')\@[es'']) \wedge publics-agree\ ves\ ves' \wedge exprs-public-agree\ es\ es''$
 $\langle proof \rangle$

lemma *exprs-public-agree-imp-publics-agree1*:

assumes *exprs-public-agree* ($(\$*ves)\@[e]$) *es'*
shows $\exists ves'\ e'.\ es' = ((\$*ves')\@[e]) \wedge publics-agree\ ves\ ves' \wedge expr-public-agree$

$e \ e'$
 $\langle \text{proof} \rangle$

lemma *exprs-public-agree-imp-publics-agree1-const0*:
assumes *exprs-public-agree* $[e] \ es'$
shows $\exists e'. \ es' = [e] \wedge \text{expr-public-agree } e \ e'$
 $\langle \text{proof} \rangle$

lemma *b-e-exprs-public-agree-imp-publics-agree1-const0*:
assumes *exprs-public-agree* $(\$*[b-e]) \ es'$
shows $\exists b-e'. \ es' = [\$b-e] \wedge \text{expr-public-agree } (\$b-e) \ (\$b-e')$
 $\langle \text{proof} \rangle$

lemma *exprs-public-agree-trap-imp-is-trap*:
assumes *exprs-public-agree* $[Trap] \ es$
shows $es = [Trap]$
 $\langle \text{proof} \rangle$

lemma *exprs-public-agree-imp-publics-agree1-const1*:
assumes *exprs-public-agree* $[(\$C \ v), e] \ es'$
shows $\exists v' \ e'. \ es' = [(\$C \ v'), e] \wedge \text{public-agree } v \ v' \wedge \text{expr-public-agree } e \ e'$
 $\langle \text{proof} \rangle$

lemma *exprs-public-agree-imp-publics-agree1-const2*:
assumes *exprs-public-agree* $[(\$C \ v1), (\$C \ v2), e] \ es'$
shows $\exists v1' \ v2' \ e'. \ es' = [(\$C \ v1'), (\$C \ v2'), e] \wedge$
 $\text{public-agree } v1 \ v1' \wedge$
 $\text{public-agree } v2 \ v2' \wedge$
 $\text{expr-public-agree } e \ e'$
 $\langle \text{proof} \rangle$

lemma *exprs-public-agree-imp-publics-agree1-const3*:
assumes *exprs-public-agree* $[(\$C \ v1), (\$C \ v2), (\$C \ v3), e] \ es'$
shows $\exists v1' \ v2' \ v3' \ e'. \ es' = [(\$C \ v1'), (\$C \ v2'), (\$C \ v3'), e] \wedge$
 $\text{public-agree } v1 \ v1' \wedge$
 $\text{public-agree } v2 \ v2' \wedge$
 $\text{public-agree } v3 \ v3' \wedge$
 $\text{expr-public-agree } e \ e'$
 $\langle \text{proof} \rangle$

lemma *publics-agree-imp-exprs-public-agree-cons*:
assumes *public-agree* $v \ v'$
 $\text{exprs-public-agree } es \ es'$
shows *exprs-public-agree* $((\$C \ v) \# es) \ ((\$C \ v') \# es')$
 $\langle \text{proof} \rangle$

lemma *publics-agree-imp-exprs-public-agree*:
assumes *publics-agree* $ves \ ves'$
 $\text{exprs-public-agree } es \ es'$

shows *exprs-public-agree* $((\$* \text{ ves})@es) ((\$* \text{ ves}')@es')$
 $\langle \text{proof} \rangle$

lemma *expr-public-agree-const*:
assumes *expr-public-agree* $e e'$
 $\text{is-const } e$
shows *is-const* e'
 $\langle \text{proof} \rangle$

lemma *exprs-public-agree-const-list*:
assumes *exprs-public-agree* $es es'$
 $\text{const-list } es$
shows *const-list* es'
 $\langle \text{proof} \rangle$

lemma *exprs-public-agree-basic*:
assumes *exprs-public-agree* $(\$* \text{ ves}) es'$
shows $\exists \text{ ves}'. es' = (\$* \text{ ves}')$
 $\langle \text{proof} \rangle$

lemma *exprs-public-agree-app3*:
assumes *exprs-public-agree* $(vs @ es @ es') les$
shows $\exists \text{ vs-a es-a es'-a. } les = \text{vs-a} @ \text{es-a} @ \text{es'-a} \wedge$
 $\text{exprs-public-agree } vs \text{ vs-a} \wedge$
 $\text{exprs-public-agree } es \text{ es-a} \wedge$
 $\text{exprs-public-agree } es' \text{ es'-a}$
 $\langle \text{proof} \rangle$

lemma *store-public-agree-imp-store-typing*:
assumes *store-typing* $s \mathcal{S}$
 $\text{store-public-agree } s s'$
shows *store-typing* $s' \mathcal{S}$
 $\langle \text{proof} \rangle$

lemma *exprs-public-agree-imp-lholed-public-agree*:
assumes $L\text{filled } k \text{ lholed } es \text{ les}$
 $\text{exprs-public-agree } les \text{ les'}$
shows $\exists \text{ lholed' es'}. \text{lholed-public-agree } \text{lholed } \text{lholed'} \wedge$
 $\text{exprs-public-agree } es \text{ es'} \wedge$
 $L\text{filled } k \text{ lholed' es' les'}$
 $\langle \text{proof} \rangle$

lemma *lholed-public-agree-imp-exprs-public-agree*:
assumes $\text{lholed-public-agree } \text{lholed } \text{lholed'}$
 $L\text{filled } k \text{ lholed } es \text{ les}$
 $\text{exprs-public-agree } es \text{ es'}$
shows $\exists \text{ les'}. L\text{filled } k \text{ lholed' es' les'} \wedge \text{exprs-public-agree } les \text{ les'}$
 $\langle \text{proof} \rangle$

method *solve-exprs-public-agree-imp-b-e-typing-trivial* =
 (match **premises in** *A:exprs-public-agree* (\$* [b-e]) (\$* bes')
 and *B:C ⊢ [b-e] : tf*
 for *b-e bes' C tf ⇒*
 ⟨*solves* ⟨*insert b-e-exprs-public-agree-imp-publics-agree1-const0*[OF A] B;
 fastforce simp add: expr-public-agree.simps⟩⟩)

lemma *exprs-public-agree-imp-b-e-typing*:
assumes *C ⊢ bes : tf*
 exprs-public-agree (\$*bes) (\$*bes')
shows *C ⊢ bes' : tf*
 ⟨*proof*⟩

lemma *exprs-public-agree-imp-e-typing-s-typing*:
S.C ⊢ es : (ts -> ts') ⇒ exprs-public-agree es es' ⇒ S.C ⊢ es' : (ts -> ts')
S.tr.rs ⊢-i vs;es : ts' ⇒ publics-agree vs vs' ⇒ exprs-public-agree es es' ⇒
S.tr.rs ⊢-i vs';es' : ts'
 ⟨*proof*⟩

lemma *exprs-public-agree-imp-config-typing*:
assumes *⊢-i s;vs;es : ts*
 store-public-agree s s'
 publics-agree vs vs'
 exprs-public-agree es es'
shows *⊢-i s';vs';es' : ts*
 ⟨*proof*⟩

fun *config-indistinguishable* :: (*s × v list × e list*) ⇒ (*s × v list × e list*) ⇒ *bool*
 (- ~'-c - 60) **where**
 ((*s,vs,es*) ~-c (*s',vs',es'*)) = (*store-public-agree s s' ∧ publics-agree vs vs' ∧*
exprs-public-agree es es')

lemma *config-indistinguishable-imp-config-typing*:
assumes *⊢-i s;vs;es : ts*
 (*s,vs,es*) ~-c (*s',vs',es'*)
shows *⊢-i s';vs';es' : ts*
 ⟨*proof*⟩

lemma *expr-public-agree-trans*:
assumes *expr-public-agree a b*
 expr-public-agree b c
shows *expr-public-agree a c*
 ⟨*proof*⟩

lemma *equivp-expr-public-agree:equivp expr-public-agree*
 ⟨*proof*⟩

lemma *equivp-exprs-public-agree:equivp exprs-public-agree*
 ⟨*proof*⟩

lemma *exprs-public-agree-trans*:
assumes *exprs-public-agree es es'*
exprs-public-agree es' es''
shows *exprs-public-agree es es''*
 $\langle \text{proof} \rangle$

lemma *equivp-store-public-agree:equivp store-public-agree*
 $\langle \text{proof} \rangle$

lemma *config-indistinguishable-refl:config-indistinguishable c c*
 $\langle \text{proof} \rangle$

lemma *config-indistinguishable-symm*:
assumes $c \sim\text{-}c \ c'$
shows $c' \sim\text{-}c \ c$
 $\langle \text{proof} \rangle$

lemma *config-indistinguishable-trans*:
assumes $c \sim\text{-}c \ c'$
 $c' \sim\text{-}c \ c''$
shows $c \sim\text{-}c \ c''$
 $\langle \text{proof} \rangle$

lemma *equivp-config-indistinguishable:equivp config-indistinguishable*
 $\langle \text{proof} \rangle$

definition *config-untrusted-equiv* :: $((s \times v \text{ list} \times e \text{ list}) \times \text{nat}) \Rightarrow ((s \times v \text{ list} \times e \text{ list}) \times \text{nat}) \Rightarrow \text{bool} \ (- \sim'\text{-}cp \text{ - } 60)$ **where**
 $\text{config-untrusted-equiv} \equiv$
 $(\lambda((s, vs, es), i) ((s', vs', es'), i')). ((s, vs, es) \sim\text{-}c (s', vs', es')) \wedge$
 $(\exists ts. \vdash\text{-}i \ s; vs; es : (\text{Untrusted}, ts)) \wedge$
 $i = i')$

lemma *ex-config-untrusted-equiv-refl*: $\exists s \ vs \ es \ i. (((s, vs, es), i) \sim\text{-}cp ((s, vs, es), i))$
 $\langle \text{proof} \rangle$

lemma *config-untrusted-equiv-symm*:
assumes $((s, vs, es), i) \sim\text{-}cp ((s', vs', es'), i')$
shows $((s', vs', es'), i') \sim\text{-}cp ((s, vs, es), i)$
 $\langle \text{proof} \rangle$

lemma *config-untrusted-equiv-trans*:
assumes $((s, vs, es), i) \sim\text{-}cp ((s'', vs'', es''), i'')$
 $((s'', vs'', es''), i'') \sim\text{-}cp ((s', vs', es'), i')$
shows $((s, vs, es), i) \sim\text{-}cp ((s', vs', es'), i')$
 $\langle \text{proof} \rangle$

lemma *part-equivp-config-untrusted-equiv:part-equivp config-untrusted-equiv*

$\langle \text{proof} \rangle$

definition *config-inst-length* :: $(s \times v \text{ list} \times e \text{ list}) \Rightarrow \text{nat}$ **where**
config-inst-length *c* = *length* (*inst* (*fst* *c*))

quotient-type *config-untrusted-quot* = $((s \times v \text{ list} \times e \text{ list}) \times \text{nat}) / \text{partial:config-untrusted-equiv}$
 $\langle \text{proof} \rangle$

lift-definition *config-untrusted-quot-inst-length* :: *config-untrusted-quot* \Rightarrow *nat* **is**
 $(\lambda(c,i). \text{length} (\text{inst} (\text{fst } c)))$
 $\langle \text{proof} \rangle$

lift-definition *config-untrusted-quot-store-typing* :: *config-untrusted-quot* \Rightarrow *s-context*
 \Rightarrow *bool* **is** $(\lambda(c,i) \mathcal{S}. \text{store-typing} (\text{fst } c) \mathcal{S})$
 $\langle \text{proof} \rangle$

lift-definition *config-untrusted-quot-e-typing* :: [*s-context*, *t-context*, *config-untrusted-quot*,
tf] \Rightarrow *bool* **is** $(\lambda \mathcal{S} \mathcal{C} (c,i) \text{tf}. (\mathcal{S} \cdot \mathcal{C} \vdash (\text{snd} (\text{snd } c)) : \text{tf}))$
 $\langle \text{proof} \rangle$

lift-definition *config-untrusted-quot-s-typing* :: [*s-context*, *trust*, (*t list*) *option*,
config-untrusted-quot, *t list*] \Rightarrow *bool* **is** $(\lambda \mathcal{S} \text{tr } rs (c,i) \text{ts}. (\mathcal{S} \cdot \text{tr} \cdot rs \Vdash\text{-i} (\text{fst} (\text{snd } c)); (\text{snd} (\text{snd } c)) : \text{ts}))$
 $\langle \text{proof} \rangle$

lift-definition *config-untrusted-quot-config-typing* :: [*config-untrusted-quot*, *trust*
 \times *t list*] \Rightarrow *bool* **is** $(\lambda((s,vs,es),i) \text{ts}. (\vdash\text{-i } s;vs;es : \text{ts}))$
 $\langle \text{proof} \rangle$

end

12 Security Proofs

theory *Wasm-Secret* **imports** *Wasm-Secret-Aux* *AFP/Coinductive/Coinductive*
HOL-Library.BNF-Corec **begin**

inductive *action-indistinguishable* :: *action* \Rightarrow *action* \Rightarrow *bool* ($\sim\text{'-a - 60}$) **where**

refl: *a* $\sim\text{'-a a}$
 $| \text{binop-32:safe-binop-i } iop \Longrightarrow (\text{Binop-i32-Some-action } iop \text{ c1 c2}) \sim\text{'-a } (\text{Binop-i32-Some-action } iop \text{ c1' c2'})$
 $| \text{binop-64:safe-binop-i } iop \Longrightarrow (\text{Binop-i64-Some-action } iop \text{ c1 c2}) \sim\text{'-a } (\text{Binop-i64-Some-action } iop \text{ c1' c2'})$
 $| \text{select:}(\text{Select-action Secret } c1) \sim\text{'-a } (\text{Select-action Secret } c2)$
 $| \text{host-Some:}[\text{store-public-agree } s \text{ s'}; \text{publics-agree } vcs \text{ vcs'}; \text{store-public-agree } s\text{'-o } s'\text{'-o}; \text{publics-agree } vcs\text{'-o } vcs'\text{'-o}] \Longrightarrow (\text{Callcl-host-Some-action } s \text{ vcs } s\text{'-o } vcs\text{'-o } \text{Untrusted } \text{tf } f \text{ hs}) \sim\text{'-a } (\text{Callcl-host-Some-action } s' \text{ vcs' } s'\text{'-o } vcs'\text{'-o } \text{Untrusted } \text{tf } f \text{ hs'})$
 $| \text{host-None:}[\text{store-public-agree } s \text{ s'}; \text{publics-agree } vcs \text{ vcs'}] \Longrightarrow (\text{Callcl-host-None-action } s \text{ vcs } \text{Untrusted } \text{tf } f \text{ hs}) \sim\text{'-a } (\text{Callcl-host-None-action } s' \text{ vcs' } \text{Untrusted } \text{tf } f \text{ hs'})$
 $| \text{convert-Some:}[\text{is-int-t } t1; \text{is-int-t } t2; \text{types-agree } t1 \text{ v}; \text{public-agree } v \text{ v'}] \Longrightarrow$

$(\text{Convert-Some-action } t1 \ t2 \ v) \sim\text{-}a \ (\text{Convert-Some-action } t1 \ t2 \ v')$
 $| \text{convert-None} : \llbracket \text{is-int-t } t1; \text{is-int-t } t2; \text{types-agree } t1 \ v; \text{public-agree } v \ v \rrbracket \implies$
 $(\text{Convert-None-action } t1 \ t2 \ v) \sim\text{-}a \ (\text{Convert-None-action } t1 \ t2 \ v')$

lemma *action-indistinguishable-symm:*

assumes $a \sim\text{-}a \ b$

shows $b \sim\text{-}a \ a$

<proof>

lemma *action-indistinguishable-trans:*

assumes $a \sim\text{-}a \ b$

$b \sim\text{-}a \ c$

shows $a \sim\text{-}a \ c$

<proof>

lemma *equivp-action-indistinguishable: equivp action-indistinguishable*

<proof>

lemma *equivp-obs: equivp (list-all2 action-indistinguishable)*

<proof>

quotient-type (overloaded) *observation = action list / list-all2 action-indistinguishable*

<proof>

abbreviation *abs-obs :: action list \Rightarrow observation (\$A - 60) where*

abs-obs a \equiv abs-observation a

inductive *reduction-actions :: [s, v list, e list, nat, action list] \Rightarrow bool (r'-actions*

(|-;-|-) - - 60) where

$\llbracket \text{const-list } es \vee es = [\text{Trap}] \rrbracket \implies \text{r-actions } (|s;vs;es|) \ i \ []$

$| \llbracket (|s;vs;es|) \ a \rightsquigarrow\text{-}i \ (|s';vs';es'|); \text{r-actions } (|s';vs';es'|) \ i \ as \rrbracket \implies \text{r-actions } (|s;vs;es|) \ i \ (a \# as)$

inductive *reduce-weight :: [s, v list, e list, nat, nat, s, v list, e list] \Rightarrow bool ((|-;-|-)*

| -| \rightsquigarrow ' - - (|-;-|-) 60) where

$(|s;vs;es|) \ a \rightsquigarrow\text{-}i \ (|s';vs';es'|) \implies (|s;vs;es|) \ |(weight \ a)| \rightsquigarrow\text{-}i \ (|s';vs';es'|)$

inductive *reduction-weight :: [s, v list, e list, nat, nat] \Rightarrow bool (r'-weight (|-;-|-) -*

- 60) where

$\llbracket \text{const-list } es \vee es = [\text{Trap}] \rrbracket \implies \text{r-weight } (|s;vs;es|) \ i \ 0$

$| \llbracket (|s;vs;es|) \ |w| \rightsquigarrow\text{-}i \ (|s';vs';es'|); \text{r-weight } (|s';vs';es'|) \ i \ w \rrbracket \implies \text{r-weight } (|s;vs;es|) \ i \ (w + w')$

lemma *r-actions-imp-r-weight:*

assumes $\text{r-actions } (|s;vs;es|) \ i \ as$

shows $\text{r-weight } (|s;vs;es|) \ i \ (\text{sum-list } (\text{map weight } as))$

<proof>

lemma *memories-public-agree-helper*:

assumes $\text{smem-ind } s \ i = \text{Some } j$
 $\text{store-public-agree } s \ s'$
 $\text{store-typing } s \ \mathcal{S}$
 $i < \text{length } (\text{inst } s)$
 $s.\text{mem } s \ ! \ j = (m, \text{sec})$
shows $\text{smem-ind } s' \ i = \text{Some } j$
 $j < \text{length } (s.\text{mem } s')$
 $\text{memories-public-agree } (s.\text{mem } s) \ (s.\text{mem } s')$
 $\text{memory-public-agree } ((s.\text{mem } s)!\ j) \ ((s.\text{mem } s')!\ j)$
 $\exists m'. s.\text{mem } s' \ ! \ j = (m', \text{sec})$
 $\langle \text{proof} \rangle$

lemma *load-helper*:

assumes $\text{smem-ind } s \ i = \text{Some } j$
 $s.\text{mem } s \ ! \ j = (m, \text{sec})$
 $\text{store-typing } s \ \mathcal{S}$
 $i < \text{length } (\text{inst } s)$
 $C = (s\text{-inst } \mathcal{S} \ ! \ i) (\text{trust-}t := tr, \text{local} := \text{local } (s\text{-inst } \mathcal{S} \ ! \ i) \ @ \ tvs, \text{label} :=$
 $\text{arb-labs}, \text{return} := \text{arb-return})$
 $S\text{-}C \vdash [\$C \ \text{ConstInt32 } \text{sec}' \ k, \$\text{Load } t \ tp \ a \ \text{off}] : (ts \rightarrow ts')$
shows $t\text{-sec } t = \text{sec}$
 $\langle \text{proof} \rangle$

lemma *store-helper*:

assumes $\text{smem-ind } s \ i = \text{Some } j$
 $s.\text{mem } s \ ! \ j = (m, \text{sec})$
 $\text{exprs-public-agree } [\$C \ \text{ConstInt32 } \text{sec}' \ k, \$C \ v, \$\text{Store } t \ tp \ a \ \text{off}] \ es'$
 $\text{store-public-agree } s \ s'$
 $\text{store-typing } s \ \mathcal{S}$
 $i < \text{length } (\text{inst } s)$
 $C = (s\text{-inst } \mathcal{S} \ ! \ i) (\text{trust-}t := tr, \text{local} := \text{local } (s\text{-inst } \mathcal{S} \ ! \ i) \ @ \ tvs, \text{label} :=$
 $\text{arb-labs}, \text{return} := \text{arb-return})$
 $S\text{-}C \vdash [\$C \ \text{ConstInt32 } \text{sec}' \ k, \$C \ v, \$\text{Store } t \ tp \ a \ \text{off}] : (ts \rightarrow ts')$
shows $t\text{-sec } t = \text{sec}$
 $\text{sec}' = \text{Public}$
 $\text{types-agree } t \ v$
 $\exists v' \ v''. es' = [\$C \ v', \$C \ v'', \$\text{Store } t \ tp \ a \ \text{off}] \wedge$
 $v' = (\text{ConstInt32 } \text{sec}' \ k) \wedge$
 $\text{public-agree } v \ v''$
 $\text{smem-ind } s' \ i = \text{Some } j$
 $j < \text{length } (s.\text{mem } s')$
 $\text{memories-public-agree } (s.\text{mem } s) \ (s.\text{mem } s')$
 $\text{memory-public-agree } ((s.\text{mem } s)!\ j) \ ((s.\text{mem } s')!\ j)$
 $\exists m'. s.\text{mem } s' \ ! \ j = (m', \text{sec})$
 $\langle \text{proof} \rangle$

lemma *load-m-imp-load-m'*:

assumes $\text{memory-public-agree } ((s.\text{mem } s)!\ j) \ ((s.\text{mem } s')!\ j)$

$s.mem\ s!\ j = (m, sec)$
 $s.mem\ s'!\ j = (m', sec)$
 $load\ m\ n\ off\ l = Some\ bs$
shows $\exists bs'. load\ m'\ n\ off\ l = Some\ bs'$
 $\langle proof \rangle$

lemma *load-packed-m-imp-load-packed-m'*:
assumes *memory-public-agree* $((s.mem\ s)!\ j)\ ((s.mem\ s')!\ j)$
 $s.mem\ s!\ j = (m, sec)$
 $s.mem\ s'!\ j = (m', sec)$
 $load-packed\ sx\ m\ n\ off\ lp\ l = Some\ bs$
shows $\exists bs'. load-packed\ sx\ m'\ n\ off\ lp\ l = Some\ bs'$
 $\langle proof \rangle$

lemma *store-m-imp-store-m'*:
assumes $t-sec\ t = sec$
 $types-agree\ t\ v$
 $public-agree\ v\ v''$
 $memory-public-agree\ ((s.mem\ s)!\ j)\ ((s.mem\ s')!\ j)$
 $s.mem\ s!\ j = (m, sec)$
 $s.mem\ s'!\ j = (m', sec)$
 $store\ m\ (nat-of-int\ k)\ off\ (bits\ v)\ (t-length\ t) = Some\ mem'$
shows $\exists mem''. store\ m'\ (nat-of-int\ k)\ off\ (bits\ v'')\ (t-length\ t) = Some\ mem''$
 \wedge
 $memory-public-agree\ (mem', sec)\ (mem'', sec)$
 $\langle proof \rangle$

lemma *store-packed-m-imp-store-packed-m'*:
assumes $t-sec\ t = sec$
 $types-agree\ t\ v$
 $public-agree\ v\ v''$
 $memory-public-agree\ ((s.mem\ s)!\ j)\ ((s.mem\ s')!\ j)$
 $s.mem\ s!\ j = (m, sec)$
 $s.mem\ s'!\ j = (m', sec)$
 $store-packed\ m\ (nat-of-int\ k)\ off\ (bits\ v)\ (tp-length\ tp) = Some\ mem'$
shows $\exists mem''. store-packed\ m'\ (nat-of-int\ k)\ off\ (bits\ v'')\ (tp-length\ tp) = Some\ mem'' \wedge$
 $memory-public-agree\ (mem', sec)\ (mem'', sec)$
 $\langle proof \rangle$

lemma *binop-i-secret-imp-binop-i-some*:
assumes *safe-binop-i iop*
shows $\exists c. app-binop-i\ iop\ c1\ c2 = Some\ c$
 $\langle proof \rangle$

lemma *cvtop-secret-imp-cvt-some*:
assumes $\mathcal{S}\mathcal{C} \vdash [\mathcal{S}C\ v,\ \mathcal{S}Cvtop\ t2\ Convert\ t1\ sx] : (ts \rightarrow ts')$
 $is-secret-t\ (typeof\ v)$
shows $\exists v'. cvt\ t2\ sx\ v = Some\ v'$

$\langle \text{proof} \rangle$

lemma *publics-agree-imp-reduce-simple:*

assumes $\langle es \rangle \ a \rightsquigarrow \langle es-a \rangle$
 $\text{exprs-public-agree } es \ es'$
 $\mathcal{S} \cdot \mathcal{C} \vdash es : (ts \rightarrow ts')$
 $\text{trust-}t \ \mathcal{C} = \text{Untrusted}$
shows $\exists a' \ es'-a. \langle es' \rangle \ a' \rightsquigarrow \langle es'-a \rangle \wedge \text{exprs-public-agree } es-a \ es'-a \wedge (a \sim\!-\!a \ a')$
 $\langle \text{proof} \rangle$

lemma *exprs-public-agree-imp-reduce:*

assumes $\langle s;vs;es \rangle \ a \rightsquigarrow\!-\ i \ \langle s-a;vs-a;es-a \rangle$
 $\text{exprs-public-agree } es \ es'$
 $\text{publics-agree } vs \ vs'$
 $\text{store-public-agree } s \ s'$
 $\text{store-typing } s \ \mathcal{S}$
 $tv\!s = \text{map } \text{typeof } vs$
 $i < \text{length } (\text{inst } s)$
 $\mathcal{C} = ((s\text{-inst } \mathcal{S})!i)(\text{trust-}t := \text{Untrusted}, \text{local} := (\text{local } ((s\text{-inst } \mathcal{S})!i) \ @ \ tv\!s), \text{label} := \text{arb-labs}, \text{return} := \text{arb-return})$
 $\mathcal{S} \cdot \mathcal{C} \vdash es : (ts \rightarrow ts')$
shows $\exists a' \ s'-a \ vs'-a \ es'-a. \langle s';vs';es' \rangle \ a' \rightsquigarrow\!-\ i \ \langle s'-a;vs'-a;es'-a \rangle \wedge$
 $\text{exprs-public-agree } es-a \ es'-a \wedge$
 $\text{publics-agree } vs-a \ vs'-a \wedge$
 $\text{store-public-agree } s-a \ s'-a \wedge$
 $(a \sim\!-\!a \ a')$
 $\langle \text{proof} \rangle$

lemma *actions-indistinguishable-secrets:*

assumes $r\text{-actions } \langle s;vs;es \rangle \ i \ as$
 $\text{exprs-public-agree } es \ es'$
 $\text{publics-agree } vs \ vs'$
 $\text{store-public-agree } s \ s'$
 $\text{store-typing } s \ \mathcal{S}$
 $tv\!s = \text{map } \text{typeof } vs$
 $i < \text{length } (\text{inst } s)$
 $\mathcal{C} = ((s\text{-inst } \mathcal{S})!i)(\text{trust-}t := \text{Untrusted}, \text{local} := (\text{local } ((s\text{-inst } \mathcal{S})!i) \ @ \ tv\!s), \text{label} := \text{arb-labs}, \text{return} := \text{arb-return})$
 $\mathcal{S} \cdot \mathcal{C} \vdash es : (ts \rightarrow ts')$
shows $\exists as'. (r\text{-actions } \langle s';vs';es' \rangle \ i \ as') \wedge \text{list-all2 } \text{action-indistinguishable } as \ as'$
 $\langle \text{proof} \rangle$

lemma *function-actions-indistinguishable-secrets:*

assumes $r\text{-actions } \langle s;vs;es \rangle \ i \ as$
 $\text{exprs-public-agree } es \ es'$
 $\text{publics-agree } vs \ vs'$
 $\text{store-public-agree } s \ s'$
 $\text{store-typing } s \ \mathcal{S}$

$\mathcal{S} \cdot \text{Untrusted} \cdot rs \Vdash -i \text{ vs; es : } ts'$
shows $\exists as'. (r\text{-actions } \langle s'; vs'; es' \rangle \ i \ as') \wedge \text{list-all2 action-indistinguishable as}$
 as'
 $\langle \text{proof} \rangle$

theorem *config-actions-indistinguishable-secrets:*

assumes $\vdash -i \text{ s; vs; es : } (\text{Untrusted}, ts)$
 $r\text{-actions } \langle s; vs; es \rangle \ i \ as$
 $\text{exprs-public-agree es es'}$
 $\text{publics-agree vs vs'}$
 $\text{store-public-agree s s'}$
shows $\exists as'. (r\text{-actions } \langle s'; vs'; es' \rangle \ i \ as') \wedge \text{list-all2 action-indistinguishable as}$
 as'
 $\langle \text{proof} \rangle$

lemma *config-indistinguishable-imp-reduce:*

assumes $\langle s; vs; es \rangle \ a \rightsquigarrow -i \langle s-a; vs-a; es-a \rangle$
 $(s, vs, es) \sim -c (s', vs', es')$
 $\vdash -i \text{ s; vs; es : } (\text{Untrusted}, ts)$
shows $\exists a' \ s' \text{-a } vs' \text{-a } es' \text{-a}. \langle s'; vs'; es' \rangle \ a' \rightsquigarrow -i \langle s' \text{-a}; vs' \text{-a}; es' \text{-a} \rangle \wedge$
 $((s-a, vs-a, es-a) \sim -c (s' \text{-a}, vs' \text{-a}, es' \text{-a})) \wedge$
 $(a \sim -a \ a')$
 $\langle \text{proof} \rangle$

definition *config-bisimulation* $:: ((s \times v \text{ list} \times e \text{ list}) \times \text{nat}) \text{ rel} \Rightarrow \text{bool}$ **where**

config-bisimulation $R \equiv$
 $\forall (((s1, vs1, es1), i1), ((s2, vs2, es2), i2)) \in R.$
 $(\forall s1' \ vs1' \ es1' \ a. \langle s1; vs1; es1 \rangle \ a \rightsquigarrow -i1 \langle s1'; vs1'; es1' \rangle \longrightarrow (\exists s2' \ vs2' \ es2' \ a'.$
 $\langle s2; vs2; es2 \rangle \ a' \rightsquigarrow -i2 \langle s2'; vs2'; es2' \rangle \wedge (((s1', vs1', es1'), i1), ((s2', vs2', es2'), i2)) \in$
 $R \wedge (a \sim -a \ a'))))$
 $\wedge (\forall s2' \ vs2' \ es2' \ a. \langle s2; vs2; es2 \rangle \ a \rightsquigarrow -i2 \langle s2'; vs2'; es2' \rangle \longrightarrow (\exists s1' \ vs1' \ es1' \ a'.$
 $\langle s1; vs1; es1 \rangle \ a' \rightsquigarrow -i1 \langle s1'; vs1'; es1' \rangle \wedge (((s1', vs1', es1'), i1), ((s2', vs2', es2'), i2)) \in$
 $R \wedge (a \sim -a \ a'))))$

definition *config-bisimilar* $:: ((s \times v \text{ list} \times e \text{ list}) \times \text{nat}) \text{ rel}$ **where**

config-bisimilar $\equiv \bigcup \{ R. \text{config-bisimulation } R \}$

lemma *config-bisimilar-ex-config-bisimulation:*

assumes $((s, vs, es), i), ((s', vs', es'), i') \in \text{config-bisimilar}$
shows $\exists R. \text{config-bisimulation } R \wedge (((s, vs, es), i), ((s', vs', es'), i')) \in R$
 $\langle \text{proof} \rangle$

definition *typed-indistinguishable-pairs* $:: ((s \times v \text{ list} \times e \text{ list}) \times \text{nat}) \text{ rel}$ **where**

typed-indistinguishable-pairs \equiv
 $\{ (((s, vs, es), i1), ((s', vs', es'), i2)). ((s, vs, es) \sim -c (s', vs', es')) \wedge i1 = i2$
 $\wedge (\exists ts. \vdash -i1 \text{ s; vs; es : } (\text{Untrusted}, ts)) \}$

lemma *config-bisimulation-typed-indistinguishable-pairs1:*

assumes $((s1, vs1, es1), i1), ((s2, vs2, es2), i2) \in \text{typed-indistinguishable-pairs}$

$$\langle s1;vs1;es1 \rangle a1 \rightsquigarrow_{-i1} \langle s1';vs1';es1' \rangle$$
shows $(\exists s2' vs2' es2' a2 ts. \langle s2;vs2;es2 \rangle a2 \rightsquigarrow_{-i2} \langle s2';vs2';es2' \rangle \wedge$

$$(((s1',vs1',es1'),i1),((s2',vs2',es2'),i2)) \in$$

typed-indistinguishable-pairs \wedge

$$(a1 \sim_a a2))$$

<proof>

lemma *config-bisimulation-typed-indistinguishable-pairs2*:
assumes $((s1,vs1,es1),i1), ((s2,vs2,es2),i2) \in \textit{typed-indistinguishable-pairs}$
 $\langle s2;vs2;es2 \rangle a2 \rightsquigarrow_{-i2} \langle s2';vs2';es2' \rangle$
shows $(\exists s1' vs1' es1' a1 ts. \langle s1;vs1;es1 \rangle a1 \rightsquigarrow_{-i1} \langle s1';vs1';es1' \rangle \wedge$

$$(((s1',vs1',es1'),i1),((s2',vs2',es2'),i2)) \in$$

typed-indistinguishable-pairs \wedge

$$(a2 \sim_a a1))$$

<proof>

lemma *config-bisimulation-typed-indistinguishable-pairs*:
config-bisimulation typed-indistinguishable-pairs
<proof>

theorem *config-indistinguishable-imp-config-bisimilar*:
assumes $(s,vs,es) \sim_c (s',vs',es')$
 $\vdash_i s;vs;es : (\textit{Untrusted},ts)$
shows $((s,vs,es),i), ((s',vs',es'),i) \in \textit{config-bisimilar}$

<proof>

inductive *reduce-relpowp* :: $s \Rightarrow v \textit{ list} \Rightarrow e \textit{ list} \Rightarrow \textit{action list} \Rightarrow \textit{nat} \Rightarrow s \Rightarrow v \textit{ list}$
 $\Rightarrow e \textit{ list} \Rightarrow \textit{bool}$ $(\langle -;;- \rangle \rightsquigarrow^{\wedge}_{-i} \langle -;;- \rangle \ 60)$ **where**
 $\langle s;vs;es \rangle \square \rightsquigarrow^{\wedge}_{-i} \langle s;vs;es \rangle$
 $\mid \llbracket \langle s;vs;es \rangle a \rightsquigarrow_{-i} \langle s'';vs'';es'' \rangle; \langle s'';vs'';es'' \rangle as \rightsquigarrow^{\wedge}_{-i} \langle s';vs';es' \rangle \rrbracket \Longrightarrow \langle s;vs;es \rangle$
 $(a \# as) \rightsquigarrow^{\wedge}_{-i} \langle s';vs';es' \rangle$

theorem *config-indistinguishable-trace-noninterference*:
assumes $\langle s;vs;es \rangle as \rightsquigarrow^{\wedge}_{-i} \langle s-as;vs-as;es-as \rangle$
 $(s,vs,es) \sim_c (s',vs',es')$
 $\vdash_i s;vs;es : (\textit{Untrusted},ts)$
shows $\exists s'-as vs'-as es'-as as'. \langle s';vs';es' \rangle as' \rightsquigarrow^{\wedge}_{-i} \langle s'-as;vs'-as;es'-as \rangle \wedge$

$$\textit{list-all2 action-indistinguishable as as'} \wedge$$

$$\textit{config-indistinguishable (s-as,vs-as,es-as) (s'-as,vs'-as,es'-as)}$$

<proof>

lemma *rep-config-untrusted-quot-typing*:
assumes $((s,vs,es),i) = (\textit{rep-config-untrusted-quot } x)$
shows $\exists ts. \vdash_i s;vs;es : (\textit{Untrusted},ts)$

<proof>

end

13 Constant Time (coinductive)

theory *Wasm-Constant-Time* **imports** *Wasm-Secret* **begin**

lemma *equivp-observation*: *equivp (llist-all2 action-indistinguishable)*
 <proof>

quotient-type (overloaded) *observation = action llist / llist-all2 action-indistinguishable*
 <proof>

coinductive *config-is-trace* :: $[(s \times v \text{ list} \times e \text{ list}), \text{nat}, \text{action llist}] \Rightarrow \text{bool}$ **where**
 base: $\llbracket \forall s' vs' es' a'. \neg (s; vs; es) \ a' \rightsquigarrow\!-\!i \ (s'; vs'; es') \rrbracket \Longrightarrow \text{config-is-trace } (s, vs, es)$
i LNil
 | step: $\llbracket (s; vs; es) \ a' \rightsquigarrow\!-\!i \ (s'; vs'; es'); \text{config-is-trace } (s', vs', es') \ i \ tr \rrbracket \Longrightarrow \text{config-is-trace } (s, vs, es) \ i \ (LCons \ a \ tr)$

definition *config-trace-set* :: $[(s \times v \text{ list} \times e \text{ list}), \text{nat}] \Rightarrow (\text{action llist}) \text{ set}$ **where**
config-trace-set $\equiv \lambda c \ i. \text{Collect } (\text{config-is-trace } c \ i)$

definition *ct-prop* :: $[(s \times v \text{ list} \times e \text{ list}), \text{nat}, \text{action llist}] \Rightarrow \text{bool}$ **where**
ct-prop $c \ i \ tr \equiv \exists tr'. \text{llist-all2 action-indistinguishable } tr \ tr' \wedge \text{config-is-trace } c \ i \ tr'$

coinductive *P-co* :: $[(s \times v \text{ list} \times e \text{ list}), \text{nat}, \text{action llist}] \Rightarrow \text{bool}$ **where**
 base: $\llbracket \forall s' vs' es' a'. \neg (s; vs; es) \ a' \rightsquigarrow\!-\!i \ (s'; vs'; es') \rrbracket \Longrightarrow P\text{-co } (s, vs, es) \ i \ LNil$
 | step: $\llbracket (s; vs; es) \ a' \rightsquigarrow\!-\!i \ (s'; vs'; es'); \text{action-indistinguishable } a \ a'; P\text{-co } (s', vs', es') \ i \ tr \rrbracket \Longrightarrow P\text{-co } (s, vs, es) \ i \ (LCons \ a \ tr)$

thm *P-co.coinduct*

lemma *ct-prop-coinduct-weak*[*consumes 1, case-names ct-prop*]:

assumes *base*: $X \ x \ i \ x \ b$

and *step*:

$(\bigwedge x1 \ x2 \ x3. \\ X \ x1 \ x2 \ x3 \Longrightarrow \\ (\exists s \ vs \ es \ i. \\ \quad x1 = (s, \ vs, \ es) \wedge \\ \quad x2 = i \wedge \\ \quad x3 = LNil \wedge \\ \quad (\forall s' \ vs' \ es' \ a'. \neg (s; vs; es) \ a' \rightsquigarrow\!-\!i \ (s'; vs'; es')) \vee \\ (\exists s \ vs \ es \ a' \ i \ s' \ vs' \ es' \ a \ tr. \\ \quad x1 = (s, \ vs, \ es) \wedge \\ \quad x2 = i \wedge \\ \quad x3 = LCons \ a \ tr \wedge \\ \quad (s; vs; es) \ a' \rightsquigarrow\!-\!i \ (s'; vs'; es') \wedge \\ \quad (a \rightsquigarrow\!-\!a \ a') \wedge \\ \quad X \ (s', \ vs', \ es') \ i \ tr)))$

shows *ct-prop* $x \ a \ i \ x \ b$

<proof>

lemma *config-indistinguishable-imp-reduce2*:

assumes $(s, vs, es) \sim_c (s', vs', es')$
 $\vdash_i s; vs; es : (Untrusted, ts)$
 $config-is-trace (s, vs, es) i tr$
shows $ct-prop (s', vs', es') i tr$
 $\langle proof \rangle$

lemma *config-indistinguishable-imp-reduce3*:

assumes $(s, vs, es) \sim_c (s', vs', es')$
 $\vdash_i s; vs; es : (Untrusted, ts)$
 $config-is-trace (s, vs, es) i tr$
shows $\exists tr'. llist-all2 action-indistinguishable tr tr' \wedge config-is-trace (s', vs', es') i tr'$
 $\langle proof \rangle$

lemma *program-actions-set2-indistinguishable-secrets-co*:

assumes $tr \in (config-trace-set (s, vs, es) i)$
 $(s, vs, es) \sim_c (s', vs', es')$
 $\vdash_i s; vs; es : (Untrusted, ts)$
shows $\exists tr' \in (config-trace-set (s', vs', es') i). llist-all2 action-indistinguishable tr tr'$
 $\langle proof \rangle$

lemma *program-actions2-indistinguishable-secrets-abs-set-co*:

assumes $t \in (image abs-observation (config-trace-set (s, vs, es) i))$
 $(s, vs, es) \sim_c (s', vs', es')$
 $\vdash_i s; vs; es : (Untrusted, ts)$
shows $t \in (image abs-observation (config-trace-set (s', vs', es') i))$
 $\langle proof \rangle$

lemma *program-actions2-indistinguishable-secrets-abs-set-equiv-co*:

assumes $(s, vs, es) \sim_c (s', vs', es')$
 $\vdash_i s; vs; es : (Untrusted, ts)$
shows $(image abs-observation (config-trace-set (s, vs, es) i)) = (image abs-observation (config-trace-set (s', vs', es') i))$
 $\langle proof \rangle$

lift-definition *config-obs-set* :: $((s \times v\ list \times e\ list) \times nat) \Rightarrow observation\ set$ **is**
 $(\lambda(c, i). (config-trace-set c i)) \langle proof \rangle$

lift-definition *config-untrusted-quot-obs-set* :: $config-untrusted-quot \Rightarrow observation\ set$ **is** $(\lambda c. config-obs-set c)$
 $\langle proof \rangle$

definition *constant-time* :: $((s \times v\ list \times e\ list) \times nat) \Rightarrow bool$ **where**
 $constant-time = (\lambda(c, i). \forall c'. (c \sim_c c') \longrightarrow ((config-obs-set (c, i)) = (config-obs-set (c', i))))$

theorem *config-untrusted-constant-time*:

assumes $\vdash\text{-}i\ s;vs;es : (Untrusted,ts)$
shows *constant-time* $((s,vs,es),i)$
 $\langle proof \rangle$

lift-definition *config-untrusted-quot-constant-time* :: *config-untrusted-quot* \Rightarrow *bool*
is *constant-time*
 $\langle proof \rangle$

lemma *config-untrusted-quot-constant-time-trivial*:
config-untrusted-quot-constant-time = $(\lambda x. True)$
 $\langle proof \rangle$

end

14 Constant Time (inductive)

theory *Wasm-Constant-Time-Ind* **imports** *Wasm-Secret* **begin**

definition *config-actions* :: $[s, v\ list, e\ list, nat, action\ list] \Rightarrow bool$ (*p'-actions* $\langle\!-\!;\!-\!\rangle$ - - 60) **where**
config-actions $s\ vs\ es\ i\ as \equiv (\exists s'\ vs'\ es'. \langle s;vs;es \rangle\ as \hat{\sim}\text{-}i\ \langle s';vs';es' \rangle)$

definition *config-trace-set-ind* :: $[(s \times v\ list \times e\ list), nat] \Rightarrow (action\ list)\ set$
where
config-trace-set-ind $\equiv \lambda(s,vs,es)\ i. Collect\ (config-actions\ s\ vs\ es\ i)$

lemma *config-actions-indistinguishable-secrets-ind*:
assumes (*p-actions* $\langle s;vs;es \rangle\ i\ as$)
 $(s,vs,es) \sim\text{-}c\ (s',vs',es')$
 $\vdash\text{-}i\ s;vs;es : (Untrusted,ts)$
shows $\exists as'. (p\text{-actions}\ \langle s';vs';es' \rangle\ i\ as') \wedge list\text{-all2}\ action\text{-indistinguishable}\ as\ as'$
 $\langle proof \rangle$

lemma *config-actions-indistinguishable-secrets-abs-ind*:
assumes (*p-actions* $\langle s;vs;es \rangle\ i\ as$)
 $(s,vs,es) \sim\text{-}c\ (s',vs',es')$
 $\vdash\text{-}i\ s;vs;es : (Untrusted,ts)$
shows $\exists as'. (p\text{-actions}\ \langle s';vs';es' \rangle\ i\ as') \wedge (\$A\ as) = (\$A\ as')$
 $\langle proof \rangle$

lemma *config-trace-set-ind-indistinguishable-secrets-ind*:
assumes $as \in (config\text{-trace}\text{-set}\text{-ind}\ (s,vs,es)\ i)$
 $(s,vs,es) \sim\text{-}c\ (s',vs',es')$
 $\vdash\text{-}i\ s;vs;es : (Untrusted,ts)$
shows $\exists as' \in (config\text{-trace}\text{-set}\text{-ind}\ (s',vs',es')\ i). list\text{-all2}\ action\text{-indistinguishable}\ as\ as'$
 $\langle proof \rangle$

lemma *config-actions-indistinguishable-secrets-abs-set-ind*:
assumes $t \in (\text{image } \text{abs-obs } (\text{config-trace-set-ind } (s, vs, es) \ i))$
 $(s, vs, es) \sim\text{-}c (s', vs', es')$
 $\vdash\text{-}i \ s; vs; es : (\text{Untrusted}, ts)$
shows $t \in (\text{image } \text{abs-obs } (\text{config-trace-set-ind } (s', vs', es') \ i))$
 $\langle \text{proof} \rangle$

lemma *config-actions-indistinguishable-secrets-abs-set-equiv-ind*:
assumes $(s, vs, es) \sim\text{-}c (s', vs', es')$
 $\vdash\text{-}i \ s; vs; es : (\text{Untrusted}, ts)$
shows $(\text{image } \text{abs-obs } (\text{config-trace-set-ind } (s, vs, es) \ i)) = (\text{image } \text{abs-obs } (\text{config-trace-set-ind } (s', vs', es') \ i))$
 $\langle \text{proof} \rangle$

lift-definition *config-obs-set-ind* :: $((s \times v \text{ list} \times e \text{ list}) \times \text{nat}) \Rightarrow \text{observation set}$
is $(\lambda(c, i). (\text{config-trace-set-ind } c \ i)) \ \langle \text{proof} \rangle$

lift-definition *config-untrusted-quot-obs-set-ind* :: *config-untrusted-quot* \Rightarrow *observation set* **is** $(\lambda c. \text{config-obs-set-ind } c)$
 $\langle \text{proof} \rangle$

definition *constant-time-ind* :: $((s \times v \text{ list} \times e \text{ list}) \times \text{nat}) \Rightarrow \text{bool}$ **where**
 $\text{constant-time-ind} = (\lambda(c, i). \forall c'. (c \sim\text{-}c \ c') \longrightarrow ((\text{config-obs-set-ind } (c, i)) = (\text{config-obs-set-ind } (c', i))))$

theorem *config-untrusted-constant-time-ind*:
assumes $\vdash\text{-}i \ s; vs; es : (\text{Untrusted}, ts)$
shows $\text{constant-time-ind } ((s, vs, es), i)$
 $\langle \text{proof} \rangle$

lift-definition *config-untrusted-quot-constant-time-ind* :: *config-untrusted-quot* \Rightarrow *bool* **is** *constant-time-ind*
 $\langle \text{proof} \rangle$

lemma *config-untrusted-quot-constant-time-trivial-ind*:
 $\text{config-untrusted-quot-constant-time-ind} = (\lambda x. \text{True})$
 $\langle \text{proof} \rangle$

end

15 Set Based Leakage Model (sketch)

theory *Wasm-Leakage* **imports** *Wasm-Secret* **begin**

datatype *arith-leakage* =
 $\text{Unop-i32-leakage } \text{unop-i}$
 $| \text{Unop-i64-leakage } \text{unop-i}$
 $| \text{Unop-f32-leakage } \text{unop-f } f32$
 $| \text{Unop-f64-leakage } \text{unop-f } f64$

- | *Binop-i32-Some-safe-leakage binop-i*
- | *Binop-i32-None-safe-leakage binop-i*
- | *Binop-i64-Some-safe-leakage binop-i*
- | *Binop-i64-None-safe-leakage binop-i*
- | *Binop-i32-Some-leakage binop-i i32 i32*
- | *Binop-i32-None-leakage binop-i i32 i32*
- | *Binop-i64-Some-leakage binop-i i64 i64*
- | *Binop-i64-None-leakage binop-i i64 i64*
- | *Binop-f32-Some-leakage binop-f f32 f32*
- | *Binop-f32-None-leakage binop-f f32 f32*
- | *Binop-f64-Some-leakage binop-f f64 f64*
- | *Binop-f64-None-leakage binop-f f64 f64*
- | *Testop-i32-leakage testop*
- | *Testop-i64-leakage testop*
- | *Relop-i32-leakage relop-i*
- | *Relop-i64-leakage relop-i*
- | *Relop-f32-leakage relop-f f32 f32*
- | *Relop-f64-leakage relop-f f64 f64*

datatype *host-leakage* =

- | *Callcl-host-Some-leakage mem list*
- | *Callcl-host-None-leakage mem list*

datatype *leakage* =

- | *Arith-leakage arith-leakage*
- | *Host-leakage host-leakage*
- | *Empty-leakage*
- | *Convert-Some-int-leakage t t*
- | *Convert-None-int-leakage t t*
- | *Convert-Some-leakage t t v*
- | *Convert-None-leakage t t v*
- | *Select-leakage i32 option*
- | *If-false-leakage i32*
- | *If-true-leakage i32*
- | *Br-if-false-leakage i32*
- | *Br-if-true-leakage i32*
- | *Br-table-leakage i32*
- | *Br-table-length-leakage i32*
- | *Call-indirect-Some-leakage i32*
- | *Call-indirect-None-leakage i32*
- | *Callcl-native-leakage nat*
- | *Load-Some-leakage t nat a off*
- | *Load-None-leakage t nat a off*
- | *Load-packed-Some-leakage tp sx nat a off*
- | *Load-packed-None-leakage tp sx nat a off*
- | *Store-Some-leakage t nat a off*
- | *Store-None-leakage t nat a off*
- | *Store-packed-Some-leakage t tp nat a off*
- | *Store-packed-None-leakage t tp nat a off*

| *Current-memory-leakage* nat
 | *Grow-memory-Some-leakage* nat nat
 | *Grow-memory-None-leakage* nat nat

definition *action-leakage* :: *action* \Rightarrow *leakage* **where**

action-leakage a =

(case a of
 | *Unop-i32-action* op' \Rightarrow *Arith-leakage* (*Unop-i32-leakage* op')
 | *Unop-i64-action* op' \Rightarrow *Arith-leakage* (*Unop-i64-leakage* op')
 | *Unop-f32-action* op' c \Rightarrow *Arith-leakage* (*Unop-f32-leakage* op' c)
 | *Unop-f64-action* op' c \Rightarrow *Arith-leakage* (*Unop-f64-leakage* op' c)
 | *Binop-i32-Some-action* op' c1 c2 \Rightarrow *Arith-leakage* (if (*safe-binop-i* op')
 then *Binop-i32-Some-safe-leakage* op'
 else *Binop-i32-Some-leakage* op' c1 c2)
 | *Binop-i32-None-action* op' c1 c2 \Rightarrow *Arith-leakage* (if (*safe-binop-i* op')
 then *Binop-i32-None-safe-leakage* op'
 else *Binop-i32-None-leakage* op' c1 c2)
 | *Binop-i64-Some-action* op' c1 c2 \Rightarrow *Arith-leakage* (if (*safe-binop-i* op')
 then *Binop-i64-Some-safe-leakage* op'
 else *Binop-i64-Some-leakage* op' c1 c2)
 | *Binop-i64-None-action* op' c1 c2 \Rightarrow *Arith-leakage* (if (*safe-binop-i* op')
 then *Binop-i64-None-safe-leakage* op'
 else *Binop-i64-None-leakage* op' c1 c2)
 | *Binop-f32-Some-action* op' c1 c2 \Rightarrow *Arith-leakage* (*Binop-f32-Some-leakage* op' c1 c2)
 | *Binop-f32-None-action* op' c1 c2 \Rightarrow *Arith-leakage* (*Binop-f32-None-leakage* op' c1 c2)
 | *Binop-f64-Some-action* op' c1 c2 \Rightarrow *Arith-leakage* (*Binop-f64-Some-leakage* op' c1 c2)
 | *Binop-f64-None-action* op' c1 c2 \Rightarrow *Arith-leakage* (*Binop-f64-None-leakage* op' c1 c2)
 | *Testop-i32-action* op' \Rightarrow *Arith-leakage* (*Testop-i32-leakage* op')
 | *Testop-i64-action* op' \Rightarrow *Arith-leakage* (*Testop-i64-leakage* op')
 | *Relop-i32-action* op' \Rightarrow *Arith-leakage* (*Relop-i32-leakage* op')
 | *Relop-i64-action* op' \Rightarrow *Arith-leakage* (*Relop-i64-leakage* op')
 | *Relop-f32-action* op' c1 c2 \Rightarrow *Arith-leakage* (*Relop-f32-leakage* op' c1 c2)
 | *Relop-f64-action* op' c1 c2 \Rightarrow *Arith-leakage* (*Relop-f64-leakage* op' c1 c2)
 | *Convert-Some-action* t1 t2 c \Rightarrow (if *is-int-t* t1 \wedge *is-int-t* t2
 then *Convert-Some-int-leakage* t1 t2
 else *Convert-Some-leakage* t1 t2 c)
 | *Convert-None-action* t1 t2 c \Rightarrow (if *is-int-t* t1 \wedge *is-int-t* t2
 then *Convert-None-int-leakage* t1 t2
 else *Convert-None-leakage* t1 t2 c)
 | *Reinterpret-action* \Rightarrow *Empty-leakage*
 | *Classify-action* \Rightarrow *Empty-leakage*
 | *Declassify-action* \Rightarrow *Empty-leakage*
 | *Unreachable-action* \Rightarrow *Empty-leakage*
 | *Nop-action* \Rightarrow *Empty-leakage*
 | *Drop-action* \Rightarrow *Empty-leakage*

| *Select-action* $sec\ c \Rightarrow$ if ($sec = Secret$) then *Select-leakage* *None* else *Select-leakage* *(Some c)*
 | *Block-action* \Rightarrow *Empty-leakage*
 | *Loop-action* \Rightarrow *Empty-leakage*
 | *If-false-action* $c \Rightarrow$ *If-false-leakage* c
 | *If-true-action* $c \Rightarrow$ *If-true-leakage* c
 | *Label-const-action* \Rightarrow *Empty-leakage*
 | *Label-trap-action* \Rightarrow *Empty-leakage*
 | *Br-action* \Rightarrow *Empty-leakage*
 | *Br-if-false-action* $c \Rightarrow$ *Br-if-false-leakage* c
 | *Br-if-true-action* $c \Rightarrow$ *Br-if-true-leakage* c
 | *Br-table-action* $c \Rightarrow$ *Br-table-leakage* c
 | *Br-table-length-action* $c \Rightarrow$ *Br-table-length-leakage* c
 | *Local-const-action* \Rightarrow *Empty-leakage*
 | *Local-trap-action* \Rightarrow *Empty-leakage*
 | *Return-action* \Rightarrow *Empty-leakage*
 | *Tee-local-action* \Rightarrow *Empty-leakage*
 | *Trap-action* \Rightarrow *Empty-leakage*
 | *Call-action* \Rightarrow *Empty-leakage*
 | *Call-indirect-Some-action* $c \Rightarrow$ *Call-indirect-Some-leakage* c
 | *Call-indirect-None-action* $c \Rightarrow$ *Call-indirect-None-leakage* c
 | *Callcl-native-action* $n \Rightarrow$ *Callcl-native-leakage* n
 | *Callcl-host-Some-action* $s\ args\ s'\ out\ tr\ tf\ host\ hs \Rightarrow$ *Host-leakage* (*Callcl-host-Some-leakage* $(map\ fst\ (filter\ (\lambda(m,sec). sec = Public)\ (mem\ s))))$)
 | *Callcl-host-None-action* $s\ args\ tr\ tf\ host\ hs \Rightarrow$ *Host-leakage* (*Callcl-host-Some-leakage* $(map\ fst\ (filter\ (\lambda(m,sec). sec = Public)\ (mem\ s))))$)
 | *Get-local-action* \Rightarrow *Empty-leakage*
 | *Set-local-action* \Rightarrow *Empty-leakage*
 | *Get-global-action* \Rightarrow *Empty-leakage*
 | *Set-global-action* \Rightarrow *Empty-leakage*
 | *Load-Some-action* $t\ n\ a\ off \Rightarrow$ *Load-Some-leakage* $t\ n\ a\ off$
 | *Load-None-action* $t\ n\ a\ off \Rightarrow$ *Load-None-leakage* $t\ n\ a\ off$
 | *Load-packed-Some-action* $tp\ sx\ n\ a\ off \Rightarrow$ *Load-packed-Some-leakage* $tp\ sx\ n\ a\ off$
 | *Load-packed-None-action* $tp\ sx\ n\ a\ off \Rightarrow$ *Load-packed-None-leakage* $tp\ sx\ n\ a\ off$
 | *Store-Some-action* $t\ n\ a\ off \Rightarrow$ *Store-Some-leakage* $t\ n\ a\ off$
 | *Store-None-action* $t\ n\ a\ off \Rightarrow$ *Store-None-leakage* $t\ n\ a\ off$
 | *Store-packed-Some-action* $t\ tp\ n\ a\ off \Rightarrow$ *Store-packed-Some-leakage* $t\ tp\ n\ a\ off$
 | *Store-packed-None-action* $t\ tp\ n\ a\ off \Rightarrow$ *Store-packed-None-leakage* $t\ tp\ n\ a\ off$
 | *Current-memory-action* $l \Rightarrow$ *Current-memory-leakage* l
 | *Grow-memory-Some-action* $l\ c \Rightarrow$ *Grow-memory-Some-leakage* $l\ c$
 | *Grow-memory-None-action* $l\ c \Rightarrow$ *Grow-memory-None-leakage* $l\ c$
 | *Label-action* \Rightarrow *Empty-leakage*
 | *Local-action* \Rightarrow *Empty-leakage*

lemma *memory-agree-filter*:

assumes *memory-public-agree* $m\ m'$

shows $(\lambda(m,sec). sec = Public)\ m = (\lambda(m,sec). sec = Public)\ m'$

(proof)

```

lemma memories-agree-filter:
  assumes memories-public-agree ms ms'
  shows filter (λ(m,sec). sec = Public) ms = filter (λ(m,sec). sec = Public) ms'
  ⟨proof⟩

lemma action-indistinguishable-imp-action-leakage-eq:
  assumes a ~-a a'
           action-leakage a = obs
  shows action-leakage a' = obs
  ⟨proof⟩

end

```

References

- [1] C. Watt. Mechanising and verifying the webassembly specification. In *Proceedings of the 7th ACM SIGPLAN International Conference on Certified Programs and Proofs*, CPP 2018, pages 53–65, New York, NY, USA, 2018. ACM.