CT-WASM

anonymous authors

November 6, 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

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
typedecl *i32*
typedecl *i64*
typedecl *f32*
typedecl *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* ⇒ *nat* ⇒ *bytes* ⇒ *bytes* **is** (λ*a n as*. *takefill* (*Abs-uint8 a*) *n as*) ⟨*proof*⟩
lift-definition *bytes-replicate* :: *nat* ⇒ *byte* ⇒ *bytes* **is** (λ*n b*. *replicate n* (*Abs-uint8 b*)) ⟨*proof*⟩
definition *msbyte* :: *bytes* ⇒ *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* ⇒ *nat* ⇒ *nat* ⇒ *bytes* **is** (λ*m n l*. *take l* (*drop n m*)) ⟨*proof*⟩
lift-definition *write-bytes* :: *mem* ⇒ *nat* ⇒ *bytes* ⇒ *mem* **is** (λ*m n bs*. (*take n m*) @ *bs* @ (*drop* (*n* + *length bs*) *m*)) ⟨*proof*⟩
lift-definition *mem-append* :: *mem* ⇒ *bytes* ⇒ *mem* **is** *append* ⟨*proof*⟩
typedecl *host*
typedecl *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* | *Abs* | *Ceil* | *Floor* | *Trunc* | *Nearest* | *Sqrt*

datatype
binop-i = *Add* | *Sub* | *Mul* | *Div sx* | *Rem sx* | *And* | *Or* | *Xor* | *Shl* | *Shr sx* | *Rotl* | *Rotr*

datatype
binop-f = *Addf* | *Subf* | *Mulf* | *Divf* | *Min* | *Max* | *Copysign*

datatype
testop = *Eqz*

datatype
relop-i = *Eq* | *Ne* | *Lt sx* | *Gt sx* | *Le sx* | *Ge sx*

datatype
relop-f = *Eqf* | *Nef* | *Ltf* | *Gtf* | *Lef* | *Gef*

datatype
cvtop = *Convert* | *Reinterpret* | *Classify* | *Declassify*

datatype — values
v =
ConstInt32 sec i32
| *ConstInt64 sec i64*
| *ConstFloat32 f32*
| *ConstFloat64 f64*

datatype — basic instructions
b-e =
Unreachable
| *Nop*
| *Drop*
| *Select sec*
| *Block tf b-e list*
| *Loop tf b-e list*
| *If tf b-e list b-e list*
| *Br i*
| *Br-if i*
| *Br-table i list i*
| *Return*
| *Call i*
| *Call-indirect i*
| *Get-local i*
| *Set-local i*
| *Tee-local i*
| *Get-global i*
| *Set-global i*
| *Load t (tp × 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* [¡hole¿] 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* :: $'a \Rightarrow 'a$
fixes *int-ctz* :: $'a \Rightarrow 'a$
fixes *int-popcnt* :: $'a \Rightarrow 'a$

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

```
  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** ⟨*proof*⟩ **end**
instantiation *i64* :: *wasm-int* **begin instance** ⟨*proof*⟩ **end**
instantiation *f32* :: *wasm-float* **begin instance** ⟨*proof*⟩ **end**
instantiation *f64* :: *wasm-float* **begin instance** ⟨*proof*⟩ **end**

consts

ui32-trunc-f32 :: *f32* ⇒ *i32 option*
si32-trunc-f32 :: *f32* ⇒ *i32 option*
ui32-trunc-f64 :: *f64* ⇒ *i32 option*
si32-trunc-f64 :: *f64* ⇒ *i32 option*

ui64-trunc-f32 :: *f32* ⇒ *i64 option*
si64-trunc-f32 :: *f32* ⇒ *i64 option*
ui64-trunc-f64 :: *f64* ⇒ *i64 option*
si64-trunc-f64 :: *f64* ⇒ *i64 option*

f32-convert-ui32 :: *i32* ⇒ *f32*
f32-convert-si32 :: *i32* ⇒ *f32*
f32-convert-ui64 :: *i64* ⇒ *f32*
f32-convert-si64 :: *i64* ⇒ *f32*

f64-convert-ui32 :: *i32* ⇒ *f64*
f64-convert-si32 :: *i32* ⇒ *f64*
f64-convert-ui64 :: *i64* ⇒ *f64*
f64-convert-si64 :: *i64* ⇒ *f64*

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

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

definition *mem-size* :: *mem* ⇒ *nat* **where**

mem-size m = *length* (*Rep-mem m*)

definition *mem-grow* :: *mem* ⇒ *nat* ⇒ *mem* **where**
mem-grow m n = *mem-append m* (*bytes-replicate* (*n* ∗ *64000*) *0*)

definition *load* :: *mem* ⇒ *nat* ⇒ *off* ⇒ *nat* ⇒ *bytes option* **where**
load m n off l = (*if* (*mem-size m* ≥ (*n*+*off*+*l*))
then Some (*read-bytes m* (*n*+*off*) *l*)
else None)

definition *sign-extend* :: *sx* ⇒ *nat* ⇒ *bytes* ⇒ *bytes* **where**
sign-extend sx l bytes = (*let msb* = *msb* (*msbyte bytes*) *in*
let byte = (*case sx of U* ⇒ *0* | *S* ⇒ *if msb then* −*1 else 0*) *in*
bytes-takefill byte l bytes)

definition *load-packed* :: *sx* ⇒ *mem* ⇒ *nat* ⇒ *off* ⇒ *nat* ⇒ *nat* ⇒ *bytes option* **where**
load-packed sx m n off lp l = *map-option* (*sign-extend sx l*) (*load m n off lp*)

definition *store* :: *mem* ⇒ *nat* ⇒ *off* ⇒ *bytes* ⇒ *nat* ⇒ *mem option* **where**
store m n off bs l = (*if* (*mem-size m* ≥ (*n*+*off*+*l*))
then Some (*write-bytes m* (*n*+*off*) (*bytes-takefill 0 l bs*))
else None)

definition *store-packed* :: *mem* ⇒ *nat* ⇒ *off* ⇒ *bytes* ⇒ *nat* ⇒ *mem option* **where**
store-packed = *store*

consts
wasm-deserialise :: *bytes* ⇒ *t* ⇒ *v*

host-apply :: *s* ⇒ *tf* ⇒ *host* ⇒ *v list* ⇒ *host-state* ⇒ (*s* × *v list*) *option*

definition *typeof* :: *v* ⇒ *t* **where**
typeof v = (*case v of*
ConstInt32 sec - ⇒ (*T-i32 sec*)
| *ConstInt64 sec -* ⇒ (*T-i64 sec*)
| *ConstFloat32 -* ⇒ *T-f32*
| *ConstFloat64 -* ⇒ *T-f64*)

definition *trust-compat* :: *trust* ⇒ *trust* ⇒ *bool* **where**
trust-compat tr tr' = (*tr* = *Trusted* ∨ (*tr* = *Untrusted* ∧ *tr'* = *Untrusted*))

definition *classify-t* :: *t* ⇒ *t* **where**
classify-t t = (*case t of*
T-i32 - ⇒ *T-i32 Secret*
| *T-i64 -* ⇒ *T-i64 Secret*
| *T-f32* ⇒ *T-f32*
| *T-f64* ⇒ *T-f64*)

definition *classify* :: $v \Rightarrow v$ **where**
classify v = (*case v of*
ConstInt32 sec c $\Rightarrow$ *ConstInt32 Secret c*
| *ConstInt64 sec c* $\Rightarrow$ *ConstInt64 Secret c*
| *ConstFloat32 c* $\Rightarrow$ *ConstFloat32 c*
| *ConstFloat64 c* $\Rightarrow$ *ConstFloat64 c*)

definition *declassify-t* :: $t \Rightarrow t$ **where**
declassify-t t = (*case t of*
T-i32 - $\Rightarrow$ *T-i32 Public*
| *T-i64* - $\Rightarrow$ *T-i64 Public*
| *T-f32* $\Rightarrow$ *T-f32*
| *T-f64* $\Rightarrow$ *T-f64*)

definition *declassify* :: $v \Rightarrow v$ **where**
declassify v = (*case v of*
ConstInt32 sec c $\Rightarrow$ *ConstInt32 Public c*
| *ConstInt64 sec c* $\Rightarrow$ *ConstInt64 Public c*
| *ConstFloat32 c* $\Rightarrow$ *ConstFloat32 c*
| *ConstFloat64 c* $\Rightarrow$ *ConstFloat64 c*)

definition *option-projl* :: $('a \times 'b)$ *option* $\Rightarrow$ $'a$ *option* **where**
option-projl x = *map-option fst x*

definition *option-projr* :: $('a \times 'b)$ *option* $\Rightarrow$ $'b$ *option* **where**
option-projr x = *map-option snd x*

definition *t-length* :: $t \Rightarrow$ *nat* **where**
t-length t = (*case t of*
T-i32 - $\Rightarrow$ *4*
| *T-i64* - $\Rightarrow$ *8*
| *T-f32* $\Rightarrow$ *4*
| *T-f64* $\Rightarrow$ *8*)

definition *tp-length* :: *tp* $\Rightarrow$ *nat* **where**
tp-length tp = (*case tp of*
Tp-i8 $\Rightarrow$ *1*
| *Tp-i16* $\Rightarrow$ *2*
| *Tp-i32* $\Rightarrow$ *4*)

definition *t-sec* :: $t \Rightarrow$ *sec* **where**
t-sec t = (*case t of*
T-i32 sec $\Rightarrow$ *sec*
| *T-i64 sec* $\Rightarrow$ *sec*
| *T-f32* $\Rightarrow$ *Public*
| *T-f64* $\Rightarrow$ *Public*)

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

is-public-t t ≡ ((*t-sec t*) = *Public*)

abbreviation *is-secret-t* :: *t* ⇒ *bool* **where**
is-secret-t t ≡ ((*t-sec t*) = *Secret*)

definition *is-int-t* :: *t* ⇒ *bool* **where**
is-int-t t = (*case t of*
T-i32 - ⇒ *True*
| *T-i64* - ⇒ *True*
| *T-f32* ⇒ *False*
| *T-f64* ⇒ *False*)

definition *is-float-t* :: *t* ⇒ *bool* **where**
is-float-t t = (*case t of*
T-i32 - ⇒ *False*
| *T-i64* - ⇒ *False*
| *T-f32* ⇒ *True*
| *T-f64* ⇒ *True*)

definition *is-mut* :: *tg* ⇒ *bool* **where**
is-mut tg = (*tg-mut tg* = *T-mut*)

definition *safe-binop-i* :: *binop-i* ⇒ *bool* **where**
safe-binop-i bop =
(*case bop of*
Div - ⇒ *False*
| *Rem* - ⇒ *False*
| - ⇒ *True*)

definition *app-unop-i* :: *unop-i* ⇒ *'i::wasm-int* ⇒ *'i::wasm-int* **where**
app-unop-i iop c =
(*case iop of*
Ctz ⇒ *int-ctz c*
| *Clz* ⇒ *int-clz c*
| *Popcnt* ⇒ *int-popcnt c*)

definition *app-unop-f* :: *unop-f* ⇒ *'f::wasm-float* ⇒ *'f::wasm-float* **where**
app-unop-f fop c =
(*case fop of*
Neg ⇒ *float-neg c*
| *Abs* ⇒ *float-abs c*
| *Ceil* ⇒ *float-ceil c*
| *Floor* ⇒ *float-floor c*
| *Trunc* ⇒ *float-trunc c*
| *Nearest* ⇒ *float-nearest c*
| *Sqrt* ⇒ *float-sqrt c*)

definition *app-binop-i* :: *binop-i* ⇒ *'i::wasm-int* ⇒ *'i::wasm-int* ⇒ (*'i::wasm-int*) *option* **where**

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

definition *app-binop-f* :: *binop-f* ⇒ *'f::wasm-float* ⇒ *'f::wasm-float* ⇒ (*'f::wasm-float*) *option* **where**
app-binop-f fop c1 c2 = (*case fop of*
Addf ⇒ *Some* (*float-add c1 c2*)
| *Subf* ⇒ *Some* (*float-sub c1 c2*)
| *Mulf* ⇒ *Some* (*float-mul c1 c2*)
| *Divf* ⇒ *Some* (*float-div c1 c2*)
| *Min* ⇒ *Some* (*float-min c1 c2*)
| *Max* ⇒ *Some* (*float-max c1 c2*)
| *Copysign* ⇒ *Some* (*float-copysign c1 c2*))

definition *app-testop-i* :: *testop* ⇒ *'i::wasm-int* ⇒ *bool* **where**
app-testop-i testop c = (*case testop of Eqz* ⇒ *int-eqz c*)

definition *app-relop-i* :: *relop-i* ⇒ *'i::wasm-int* ⇒ *'i::wasm-int* ⇒ *bool* **where**
app-relop-i rop c1 c2 = (*case rop of*
Eq ⇒ *int-eq c1 c2*
| *Ne* ⇒ *int-ne c1 c2*
| *Lt U* ⇒ *int-lt-u c1 c2*
| *Lt S* ⇒ *int-lt-s c1 c2*
| *Gt U* ⇒ *int-gt-u c1 c2*
| *Gt S* ⇒ *int-gt-s c1 c2*
| *Le U* ⇒ *int-le-u c1 c2*
| *Le S* ⇒ *int-le-s c1 c2*
| *Ge U* ⇒ *int-ge-u c1 c2*
| *Ge S* ⇒ *int-ge-s c1 c2*)

definition *app-relop-f* :: *relop-f* ⇒ *'f::wasm-float* ⇒ *'f::wasm-float* ⇒ *bool* **where**
app-relop-f rop c1 c2 = (*case rop of*
Eqf ⇒ *float-eq c1 c2*
| *Nef* ⇒ *float-ne c1 c2*
| *Ltf* ⇒ *float-lt c1 c2*

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

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

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

definition *cl-type* :: $cl \Rightarrow$ *tf-t* **where**
cl-type cl = (*case cl of Func-native - tf - -* ⇒ *tf* | *Func-host tf -* ⇒ *tf*)

definition *rglob-is-mut* :: *global* ⇒ *bool* **where**
rglob-is-mut g = (*g-mut g* = *T-mut*)

definition *stypes* :: *s* ⇒ *nat* ⇒ *nat* ⇒ *tf-t* **where**
stypes s i j = ((*types* ((*inst s*)!*i*))!*j*)

definition *sfunc-ind* :: *s* ⇒ *nat* ⇒ *nat* ⇒ *nat* **where**
sfunc-ind s i j = ((*inst.funcs* ((*inst s*)!*i*))!*j*)

definition *sfunc* :: *s* ⇒ *nat* ⇒ *nat* ⇒ *cl* **where**
sfunc s i j = (*funcs s*)!(*sfunc-ind s i j*)

definition *sglob-ind* :: *s* ⇒ *nat* ⇒ *nat* ⇒ *nat* **where**
sglob-ind s i j = ((*inst.globs* ((*inst s*)!*i*))!*j*)

definition *sglob* :: *s* ⇒ *nat* ⇒ *nat* ⇒ *global* **where**
sglob s i j = (*globs s*)!(*sglob-ind s i j*)

definition *sglob-val* :: *s* ⇒ *nat* ⇒ *nat* ⇒ *v* **where**
sglob-val s i j = *g-val* (*sglob s i j*)

definition *smem-ind* :: *s* ⇒ *nat* ⇒ *nat option* **where**
smem-ind s i = (*inst.mem* ((*inst s*)!*i*))

definition *stab-s* :: *s* ⇒ *nat* ⇒ *nat* ⇒ *cl option* **where**
stab-s s i j = (*let stabinst* = ((*tab s*)!*i*) *in* (*if* (*length* (*stabinst*) > *j*) *then* (*stabinst*!*j*) *else None*))

definition *stab* :: *s* ⇒ *nat* ⇒ *nat* ⇒ *cl option* **where**
stab s i j = (*case* (*inst.tab* ((*inst s*)!*i*)) *of Some k* => *stab-s s k j* | *None* => *None*)

definition *supdate-glob-s* :: *s* ⇒ *nat* ⇒ *v* ⇒ *s* **where**
supdate-glob-s s k v = *s*⦇*globs* := (*globs s*)[*k*:=((*globs s*)!*k*)⦇*g-val* := *v*⦈]⦈

definition *supdate-glob* :: *s* ⇒ *nat* ⇒ *nat* ⇒ *v* ⇒ *s* **where**
supdate-glob s i j v = (*let k* = *sglob-ind s i j in supdate-glob-s s k v*)

definition *is-const* :: *e* ⇒ *bool* **where**
is-const e = (*case e of Basic* (*C* -) ⇒ *True* | - ⇒ *False*)

definition *const-list* :: *e list* ⇒ *bool* **where**
const-list xs = *list-all is-const xs*

inductive *store-extension* :: *s* ⇒ *s* ⇒ *bool* **where**
⟦*insts* = *insts′*; *fs* = *fs′*; *tclss* = *tclss′*; *list-all2* (λ(*bs*,*sec*) (*bs′*,*sec′*). *mem-size bs* ≤ *mem-size bs′* ∧ *sec* = *sec′*) *bss bss′*; *gs* = *gs′*⟧ ⟹
store-extension (|*s.inst* = *insts*, *s.funcs* = *fs*, *s.tab* = *tclss*, *s.mem* = *bss*, *s.globs* = *gs*|)
(|*s.inst* = *insts′*, *s.funcs* = *fs′*, *s.tab* = *tclss′*, *s.mem* = *bss′*, *s.globs* = *gs′*|)

abbreviation *to-e-list* :: *b-e list* ⇒ *e list* ($∗ - *60*) **where**
to-e-list b-es ≡ *map Basic b-es*

abbreviation *v-to-e-list* :: *v list* ⇒ *e list* ($$∗ - *60*) **where**
v-to-e-list ves ≡ *map* (λ*v*. $*C v*) *ves*

inductive *Lfilled* :: *nat* ⇒ *Lholed* ⇒ *e list* ⇒ *e list* ⇒ *bool* **where**

L0:⟦*const-list vs*; *lholed* = (*LBase vs es′*)⟧ ⟹ *Lfilled 0 lholed es* (*vs* @ *es* @ *es′*)

| *LN*:⟦*const-list vs*; *lholed* = (*LRec vs n es′ l es″*); *Lfilled k l es lfilledk*⟧ ⟹ *Lfilled* (*k*+*1*) *lholed es* (*vs* @ [*Label n es′ lfilledk*] @ *es″*)

inductive *Lfilled-exact* :: *nat* ⇒ *Lholed* ⇒ *e list* ⇒ *e list* ⇒ *bool* **where**

L0:⟦*lholed* = (*LBase* [] [])⟧ ⟹ *Lfilled-exact 0 lholed es es*

| *LN*:⟦*const-list vs*; *lholed* = (*LRec vs n es′ l es″*); *Lfilled-exact k l es lfilledk*⟧ ⟹ *Lfilled-exact* (*k*+*1*) *lholed es* (*vs* @ [*Label n es′ lfilledk*] @ *es″*)

definition *load-store-t-bounds* :: *a* ⇒ *tp option* ⇒ *t* ⇒ *bool* **where**
load-store-t-bounds a tp t = (*case tp of*
None ⇒ *2*ˆ*a* ≤ *t-length t*
| *Some tp* ⇒ *2*ˆ*a* ≤ *tp-length tp* ∧ *tp-length tp* < *t-length t* ∧ *is-int-t t*)

definition *memory-public-agree* :: (*mem* × *sec*) ⇒ (*mem* × *sec*) ⇒ *bool* **where**
memory-public-agree x y = (*x* = *y* ∨ (*mem-size* (*fst x*) = *mem-size* (*fst y*) ∧ (*snd x* = *Secret*) ∧ (*snd y* = *Secret*)))

abbreviation *memories-public-agree* :: (*mem* $\times$ *sec*) *list* $\Rightarrow$ (*mem* $\times$ *sec*) *list* $\Rightarrow$ *bool* **where**
memories-public-agree xs ys $\equiv$ *list-all2 memory-public-agree xs ys*

definition *public-agree* :: *v* $\Rightarrow$ *v* $\Rightarrow$ *bool* **where**
public-agree x y = (*y* = *x* $\vee$ ((*typeof y*) = (*typeof x*) $\wedge$ *is-secret-t* (*typeof x*)))

abbreviation *publics-agree* :: *v list* $\Rightarrow$ *v list* $\Rightarrow$ *bool* **where**
publics-agree xs ys $\equiv$ *list-all2 public-agree xs ys*

definition *global-public-agree* :: *global* $\Rightarrow$ *global* $\Rightarrow$ *bool* **where**
global-public-agree x y = (*g-mut x* = *g-mut y* $\wedge$ *public-agree* (*g-val x*) (*g-val y*))

abbreviation *globals-public-agree* :: *global list* $\Rightarrow$ *global list* $\Rightarrow$ *bool* **where**
globals-public-agree xs ys $\equiv$ *list-all2 global-public-agree xs ys*

definition *store-public-agree* :: *s* $\Rightarrow$ *s* $\Rightarrow$ *bool* **where**
store-public-agree s s' = (*inst s* = *inst s'* $\wedge$
funcs s = *funcs s'* $\wedge$
tab s = *tab s'* $\wedge$
memories-public-agree (*mem s*) (*mem s'*) $\wedge$
globals-public-agree (*globs s*) (*globs s'*))

inductive *expr-public-agree* :: *e* $\Rightarrow$ *e* $\Rightarrow$ *bool* **where**
expr-public-agree e e
| $[\![$*public-agree v v'*$]\!]$ $\Longrightarrow$
expr-public-agree ($*C v*) ($*C v'*)
| $[\![$*list-all2 expr-public-agree* ($* *bes*) ($* *bes'*)$]\!]$ $\Longrightarrow$
expr-public-agree ($*Block tf bes*) ($*Block tf bes'*)
| $[\![$*list-all2 expr-public-agree* ($* *bes*) ($* *bes'*)$]\!]$ $\Longrightarrow$
expr-public-agree ($*Loop tf bes*) ($*Loop tf bes'*)
| $[\![$*list-all2 expr-public-agree* ($* *bes1*) ($* *bes1'*); *list-all2 expr-public-agree* ($* *bes2*) ($* *bes2'*)$]\!]$ $\Longrightarrow$
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'*$]\!]$ $\Longrightarrow$
expr-public-agree (*Label n les es*) (*Label n les' es'*)
| $[\![$*publics-agree vs vs'*; *list-all2 expr-public-agree es es'*$]\!]$ $\Longrightarrow$
expr-public-agree (*Local n i vs es*) (*Local n i vs' es'*)

abbreviation *exprs-public-agree* :: *e list* $\Rightarrow$ *e list* $\Rightarrow$ *bool* **where**
exprs-public-agree es es' $\equiv$ *list-all2 expr-public-agree es es'*

inductive *lholed-public-agree* :: *Lholed* $\Rightarrow$ *Lholed* $\Rightarrow$ *bool* **where**
$[\![$*exprs-public-agree ves ves'*; *exprs-public-agree es es'*$]\!]$ $\Longrightarrow$ *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'*$]\!]$ $\Longrightarrow$
lholed-public-agree (*LRec ves n les LN es*) (*LRec ves' n les' LN' es'*)

```
definition cvt-i32 :: sx option ⇒ v ⇒ i32 option where
  cvt-i32 sx v = (case v of
                   ConstInt32 - c ⇒ None
                 | ConstInt64 - c ⇒ Some (wasm-wrap c)
                 | ConstFloat32 c ⇒ (case sx of
                                      Some U ⇒ ui32-trunc-f32 c
                                    | Some S ⇒ si32-trunc-f32 c
                                    | None ⇒ None)
                 | ConstFloat64 c ⇒ (case sx of
                                      Some U ⇒ ui32-trunc-f64 c
                                    | Some S ⇒ si32-trunc-f64 c
                                    | None ⇒ None))

definition cvt-i64 :: sx option ⇒ v ⇒ i64 option where
  cvt-i64 sx v = (case v of
                   ConstInt32 - c ⇒ (case sx of
                                      Some U ⇒ Some (wasm-extend-u c)
                                    | Some S ⇒ Some (wasm-extend-s c)
                                    | None ⇒ None)
                 | ConstInt64 - c ⇒ None
                 | ConstFloat32 c ⇒ (case sx of
                                      Some U ⇒ ui64-trunc-f32 c
                                    | Some S ⇒ si64-trunc-f32 c
                                    | None ⇒ None)
                 | ConstFloat64 c ⇒ (case sx of
                                      Some U ⇒ ui64-trunc-f64 c
                                    | Some S ⇒ si64-trunc-f64 c
                                    | None ⇒ None))

definition cvt-f32 :: sx option ⇒ v ⇒ f32 option where
  cvt-f32 sx v = (case v of
                   ConstInt32 - c ⇒ (case sx of
                                     Some U ⇒ Some (f32-convert-ui32 c)
                                   | Some S ⇒ Some (f32-convert-si32 c)
                                   | - ⇒ None)
                 | ConstInt64 - c ⇒ (case sx of
                                     Some U ⇒ Some (f32-convert-ui64 c)
                                   | Some S ⇒ Some (f32-convert-si64 c)
                                   | - ⇒ None)
                 | ConstFloat32 c ⇒ None
                 | ConstFloat64 c ⇒ Some (wasm-demote c))

definition cvt-f64 :: sx option ⇒ v ⇒ f64 option where
  cvt-f64 sx v = (case v of
                   ConstInt32 - c ⇒ (case sx of
                                     Some U ⇒ Some (f64-convert-ui32 c)
                                   | Some S ⇒ Some (f64-convert-si32 c)
                                   | - ⇒ None)
```

| *ConstInt64* - *c* ⇒ (*case sx of*
Some U ⇒ *Some* (*f64-convert-ui64 c*)
| *Some S* ⇒ *Some* (*f64-convert-si64 c*)
| - ⇒ *None*)
| *ConstFloat32 c* ⇒ *Some* (*wasm-promote c*)
| *ConstFloat64 c* ⇒ *None*)

definition *cvt* :: *t* ⇒ *sx option* ⇒ *v* ⇒ *v option* **where**
cvt t sx v = (*case t of*
(*T-i32 sec*) ⇒ (*case* (*cvt-i32 sx v*) *of Some c* ⇒ *Some* (*ConstInt32 sec c*) | *None* ⇒ *None*)
| (*T-i64 sec*) ⇒ (*case* (*cvt-i64 sx v*) *of Some c* ⇒ *Some* (*ConstInt64 sec c*) | *None* ⇒ *None*)
| *T-f32* ⇒ (*case* (*cvt-f32 sx v*) *of Some c* ⇒ *Some* (*ConstFloat32 c*) | *None* ⇒ *None*)
| *T-f64* ⇒ (*case* (*cvt-f64 sx v*) *of Some c* ⇒ *Some* (*ConstFloat64 c*) | *None* ⇒ *None*))

definition *bits* :: *v* ⇒ *bytes* **where**
bits v = (*case v of*
ConstInt32 - *c* ⇒ (*serialise-i32 c*)
| *ConstInt64* - *c* ⇒ (*serialise-i64 c*)
| *ConstFloat32 c* ⇒ (*serialise-f32 c*)
| *ConstFloat64 c* ⇒ (*serialise-f64 c*))

definition *bitzero* :: *t* ⇒ *v* **where**
bitzero t = (*case t of*
(*T-i32 sec*) ⇒ *ConstInt32 sec 0*
| (*T-i64 sec*) ⇒ *ConstInt64 sec 0*
| *T-f32* ⇒ *ConstFloat32 0*
| *T-f64* ⇒ *ConstFloat64 0*)

definition *n-zeros* :: *t list* ⇒ *v list* **where**
n-zeros ts = (*map* (*λt. bitzero t*) *ts*)

lemma *is-int-t-exists*:
assumes *is-int-t t*
shows ∃ *sec. t* = (*T-i32 sec*) ∨ *t* = (*T-i64 sec*)
⟨*proof*⟩

lemma *is-float-t-exists*:
assumes *is-float-t t*
shows ∃ *sec. t* = *T-f32* ∨ *t* = *T-f64*
⟨*proof*⟩

lemma *int-float-disjoint*: *is-int-t t* = −(*is-float-t t*)
⟨*proof*⟩

lemma *types-agree-imp-types-agree-insecure*:
assumes *types-agree t v*
shows *types-agree-insecure t v*
⟨*proof*⟩

lemma *stab-unfold*:
assumes *stab s i j = Some cl*
shows $\exists k.$ *inst.tab* ((*inst s*)!*i*) = *Some k* $\wedge$ *length* ((*tab s*)!*k*) $>$ *j* $\wedge$((*tab s*)!*k*)!*j* = *Some cl*
⟨*proof*⟩

lemma *inj-basic*: *inj Basic*
⟨*proof*⟩

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

lemma *to-e-list-1*:[\$ *a*] = \$∗ [*a*]
⟨*proof*⟩

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

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

lemma *v-exists-b-e*:$\exists$ *ves*. (\$\$∗*vs*) = (\$∗*ves*)
⟨*proof*⟩

lemma *Lfilled-exact-imp-Lfilled*:
assumes *Lfilled-exact n lholed es LI*
shows *Lfilled n lholed es LI*
⟨*proof*⟩

lemma *Lfilled-exact-app-imp-exists-Lfilled*:
assumes *const-list ves*
Lfilled-exact n lholed (*ves*@*es*) *LI*
shows $\exists$ *lholed'*. *Lfilled n lholed' es LI*
⟨*proof*⟩

lemma *Lfilled-imp-exists-Lfilled-exact*:
assumes *Lfilled n lholed es LI*
shows $\exists$ *lholed' ves es-c*. *const-list ves* $\wedge$ *Lfilled-exact n lholed'* (*ves*@*es*@*es-c*) *LI*
⟨*proof*⟩

lemma *n-zeros-typeof*:
n-zeros ts = vs $\Longrightarrow$ (*ts = map typeof vs*)
⟨*proof*⟩

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

inductive *b-e-typing* :: [*t-context*, *b-e list*, *tf*] ⇒ *bool* (- ⊢ - : - *60*) **where**
— *num ops*
const:$\mathcal{C}$ ⊢ [*C v*] : ([] -> [(*typeof v*)])
| *unop-i*:*is-int-t t* ⟹ $\mathcal{C}$ ⊢ [*Unop-i t* -] : ([*t*] -> [*t*])
| *unop-f*:*is-float-t t* ⟹ $\mathcal{C}$ ⊢ [*Unop-f t* -] : ([*t*] -> [*t*])
| *binop-i*:⟦*is-int-t t*; (*is-secret-t t* ⟶ *safe-binop-i iop*)⟧ ⟹ $\mathcal{C}$ ⊢ [*Binop-i t iop*] : ([*t*,*t*] -> [*t*])
| *binop-f*:*is-float-t t* ⟹ $\mathcal{C}$ ⊢ [*Binop-f t* -] : ([*t*,*t*] -> [*t*])
| *testop*:*is-int-t t* ⟹ $\mathcal{C}$ ⊢ [*Testop t* -] : ([*t*] -> [(*T-i32* (*t-sec t*))])
| *relop-i*:*is-int-t t* ⟹ $\mathcal{C}$ ⊢ [*Relop-i t* -] : ([*t*,*t*] -> [(*T-i32* (*t-sec t*))])
| *relop-f*:*is-float-t t* ⟹ $\mathcal{C}$ ⊢ [*Relop-f t* -] : ([*t*,*t*] -> [(*T-i32* (*t-sec t*))])
— *convert*
| *convert*:⟦(*t1* ≠ *t2*); *t-sec t1* = *t-sec t2*; (*sx* = *None*) = ((*is-float-t t1* ∧ *is-float-t t2*) ∨ (*is-int-t t1* ∧ *is-int-t t2* ∧ (*t-length t1* < *t-length t2*)))⟧ ⟹ $\mathcal{C}$ ⊢ [*Cvtop t1 Convert t2 sx*] : ([*t2*] -> [*t1*])
— *reinterpret*
| *reinterpret*:⟦(*t1* ≠ *t2*); *t-sec t1* = *t-sec t2*; *t-length t1* = *t-length t2*⟧ ⟹ $\mathcal{C}$ ⊢ [*Cvtop t1 Reinterpret t2 None*] : ([*t2*] -> [*t1*])
— *classify*
| *classify*:⟦*is-int-t t2*; *is-public-t t2*; *classify-t t2* = *t1*⟧ ⟹ $\mathcal{C}$ ⊢ [*Cvtop t1 Classify t2 None*] : ([*t2*] -> [*t1*])
— *declassify*
| *declassify*:⟦(*trust-t* $\mathcal{C}$) = *Trusted*; *is-int-t t2*; *is-secret-t t2*; *declassify-t t2* = *t1*⟧ ⟹ $\mathcal{C}$ ⊢ [*Cvtop t1 Declassify t2 None*] : ([*t2*] -> [*t1*])
— *unreachable*, *nop*, *drop*, *select*
| *unreachable*:$\mathcal{C}$ ⊢ [*Unreachable*] : (*ts* -> *ts′*)
| *nop*:$\mathcal{C}$ ⊢ [*Nop*] : ([] -> [])
| *drop*:$\mathcal{C}$ ⊢ [*Drop*] : ([*t*] -> [])
| *select*:⟦*sec* = *Secret* ⟶ *is-secret-t t*⟧ ⟹ $\mathcal{C}$ ⊢ [*Select sec*] : ([*t*,*t*,(*T-i32 sec*)] -> [*t*])
— *block*
| *block*:⟦*tf* = (*tn* -> *tm*); $\mathcal{C}$⦇*label* := ([*tm*] @ (*label* $\mathcal{C}$))⦈ ⊢ *es* : (*tn* -> *tm*)⟧ ⟹ $\mathcal{C}$ ⊢ [*Block tf es*] : (*tn* -> *tm*)
— *loop*
| *loop*:⟦*tf* = (*tn* -> *tm*); $\mathcal{C}$⦇*label* := ([*tn*] @ (*label* $\mathcal{C}$))⦈ ⊢ *es* : (*tn* -> *tm*)⟧ ⟹ $\mathcal{C}$ ⊢ [*Loop tf es*] : (*tn* -> *tm*)
— *if then else*
| *if-wasm*:⟦*tf* = (*tn* -> *tm*); $\mathcal{C}$⦇*label* := ([*tm*] @ (*label* $\mathcal{C}$))⦈ ⊢ *es1* : (*tn* -> *tm*); $\mathcal{C}$⦇*label* := ([*tm*] @ (*label* $\mathcal{C}$))⦈ ⊢ *es2* : (*tn* -> *tm*)⟧ ⟹ $\mathcal{C}$ ⊢ [*If tf es1 es2*] : (*tn* @ [(*T-i32 Public*)] -> *tm*)
— *br*
| *br*:⟦*i* < *length*(*label* $\mathcal{C}$); (*label* $\mathcal{C}$)!*i* = *ts*⟧ ⟹ $\mathcal{C}$ ⊢ [*Br i*] : (*t1s* @ *ts* -> *t2s*)
— *br-if*
| *br-if*:⟦*i* < *length*(*label* $\mathcal{C}$); (*label* $\mathcal{C}$)!*i* = *ts*⟧ ⟹ $\mathcal{C}$ ⊢ [*Br-if i*] : (*ts* @ [(*T-i32 Public*)] -> *ts*)

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

inductive *cl-typing* :: [*s-context*, *cl*, *tf-t*] $\Rightarrow$ *bool* **where**
$\llbracket i <$ *length* (*s-inst* $\mathcal{S}$); ((*s-inst* $\mathcal{S}$)!i) = $\mathcal{C}$; *tf* = (*t1s* -> *t2s*); $\mathcal{C}$(|*trust-t* := *tr*, *local* := (*local* $\mathcal{C}$) @ *t1s* @ *ts*, *label* := ([*t2s*] @ (*label* $\mathcal{C}$)), *return* := *Some t2s*|) $\vdash$ *es* : ([] -> *t2s*)$\rrbracket$ $\Longrightarrow$ *cl-typing* $\mathcal{S}$ (*Func-native i* (*tr*,*tf*) *ts es*) (*tr*,(*t1s* -> *t2s*))
| *cl-typing* $\mathcal{S}$ (*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**

$\mathcal{C} \vdash \mathit{b\text{-}es} : \mathit{tf} \Longrightarrow \mathcal{S}{\bullet}\mathcal{C} \vdash \${*}\mathit{b\text{-}es} : \mathit{tf}$

$| \; [\![\mathcal{S}{\bullet}\mathcal{C} \vdash \mathit{es} : (\mathit{t1s} \text{ -> } \mathit{t2s});\ \mathcal{S}{\bullet}\mathcal{C} \vdash [e] : (\mathit{t2s} \text{ -> } \mathit{t3s})]\!] \Longrightarrow \mathcal{S}{\bullet}\mathcal{C} \vdash \mathit{es} \text{ @ } [e] : (\mathit{t1s} \text{ -> } \mathit{t3s})$

$| \; \mathcal{S}{\bullet}\mathcal{C} \vdash \mathit{es} : (\mathit{t1s} \text{ -> } \mathit{t2s}) \Longrightarrow \mathcal{S}{\bullet}\mathcal{C} \vdash \mathit{es} : (\mathit{ts} \text{ @ } \mathit{t1s} \text{ -> } \mathit{ts} \text{ @ } \mathit{t2s})$

$| \; \mathcal{S}{\bullet}\mathcal{C} \vdash [\mathit{Trap}] : \mathit{tf}$

$| \; [\![\mathcal{S}{\bullet}(\mathit{trust\text{-}t}\ \mathcal{C}){\bullet}\mathit{Some\ ts} \Vdash\text{-}i\ \mathit{vs};\mathit{es} : \mathit{ts};\ \mathit{length\ ts} = n]\!] \Longrightarrow \mathcal{S}{\bullet}\mathcal{C} \vdash [\mathit{Local}\ n\ i\ \mathit{vs}\ \mathit{es}] : ([] \text{ -> } \mathit{ts})$

$| \; [\![\mathit{trust\text{-}compat}\ (\mathit{trust\text{-}t}\ \mathcal{C})\ \mathit{tr};\ \mathit{cl\text{-}typing}\ \mathcal{S}\ \mathit{cl}\ (\mathit{tr},\mathit{tf})]\!] \Longrightarrow \mathcal{S}{\bullet}\mathcal{C} \vdash [\mathit{Callcl}\ \mathit{cl}] : \mathit{tf}$

$| \; [\![\mathcal{S}{\bullet}\mathcal{C} \vdash \mathit{e0s} : (\mathit{ts} \text{ -> } \mathit{t2s});\ \mathcal{S}{\bullet}\mathcal{C}(\!|\mathit{label} := ([\mathit{ts}] \text{ @ } (\mathit{label}\ \mathcal{C}))|\!) \vdash \mathit{es} : ([] \text{ -> } \mathit{t2s});\ \mathit{length\ ts} = n]\!] \Longrightarrow \mathcal{S}{\bullet}\mathcal{C} \vdash [\mathit{Label}\ n\ \mathit{e0s}\ \mathit{es}] : ([] \text{ -> } \mathit{t2s})$

$| \; [\![i < (\mathit{length}\ (\mathit{s\text{-}inst}\ \mathcal{S}));\ \mathit{tvs} = \mathit{map\ typeof\ vs};\ \mathcal{C} = ((\mathit{s\text{-}inst}\ \mathcal{S})!i)(\!|\mathit{trust\text{-}t} := \mathit{tr},\ \mathit{local} := (\mathit{local}\ ((\mathit{s\text{-}inst}\ \mathcal{S})!i) \text{ @ } \mathit{tvs}),\ \mathit{return} := \mathit{rs}|\!);\ \mathcal{S}{\bullet}\mathcal{C} \vdash \mathit{es} : ([] \text{ -> } \mathit{ts});\ (\mathit{rs} = \mathit{Some\ ts}) \lor \mathit{rs} = \mathit{None}]\!] \Longrightarrow \mathcal{S}{\bullet}\mathit{tr}{\bullet}\mathit{rs} \Vdash\text{-}i\ \mathit{vs};\mathit{es} : \mathit{ts}$

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

definition $\mathit{memi\text{-}agree}\ \mathit{sm}\ j\ m = ((\exists j'\ m'.\ j = \mathit{Some}\ j' \land j' < \mathit{length\ sm} \land m = \mathit{Some}\ m' \land \mathit{sm}!j' = m') \lor j = \mathit{None} \land m = \mathit{None})$

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

inductive *inst-typing* :: [*s-context*, *inst*, *t-context*] $\Rightarrow$ *bool* **where**
$[\![\mathit{list\text{-}all2}\ (\mathit{funci\text{-}agree}\ (\mathit{s\text{-}funcs}\ \mathcal{S}))\ \mathit{fs}\ \mathit{tfs};\ \mathit{list\text{-}all2}\ (\mathit{globi\text{-}agree}\ (\mathit{s\text{-}globs}\ \mathcal{S}))\ \mathit{gs}\ \mathit{tgs};\ (i = \mathit{Some}\ i' \land i' < \mathit{length}\ (\mathit{s\text{-}tab}\ \mathcal{S}) \land (\mathit{s\text{-}tab}\ \mathcal{S})!i' = (\mathit{the}\ n)) \lor (i = \mathit{None} \land n = \mathit{None});\ \mathit{memi\text{-}agree}\ (\mathit{s\text{-}mem}\ \mathcal{S})\ j\ m]\!] \Longrightarrow \mathit{inst\text{-}typing}\ \mathcal{S}\ (\!|\mathit{types} = \mathit{ts},\ \mathit{funcs} = \mathit{fs},\ \mathit{tab} = i,\ \mathit{mem} = j,\ \mathit{globs} = \mathit{gs}|\!)\ (\!|\mathit{trust\text{-}t} = \mathit{tr},\ \mathit{types\text{-}t} = \mathit{ts},\ \mathit{func\text{-}t} = \mathit{tfs},\ \mathit{global} = \mathit{tgs},\ \mathit{table} = n,\ \mathit{memory} = m,\ \mathit{local} = [],\ \mathit{label} = [],\ \mathit{return} = \mathit{None}|\!)$

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

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

definition $\mathit{mem\text{-}agree}\ \mathit{bs}\ m = (\lambda\ (\mathit{bs},\mathit{sec})\ (m,\mathit{sec}').\ m \le \mathit{mem\text{-}size}\ \mathit{bs} \land \mathit{sec} = \mathit{sec}')\ \mathit{bs}\ m$

inductive *store-typing* :: [*s*, *s-context*] ⇒ *bool* **where**
⟦𝒮 = (|*s-inst* = 𝒞*s*, *s-funcs* = *tfs*, *s-tab* = *ns*, *s-mem* = *ms*, *s-globs* = *tgs*|); *list-all2* (*inst-typing* 𝒮) *insts* 𝒞*s*; *list-all2* (*cl-typing* 𝒮) *fs* *tfs*; *list-all* (*tab-agree* 𝒮) (*concat* *tclss*); *list-all2* (λ *tcls* *n*. *n* ≤ *length* *tcls*) *tclss* *ns*; *list-all2* *mem-agree* *bss* *ms*; *list-all2* *glob-agree* *gs* *tgs*⟧ ⟹ *store-typing* (|*s.inst* = *insts*, *s.funcs* = *fs*, *s.tab* = *tclss*, *s.mem* = *bss*, *s.globs* = *gs*|) 𝒮

inductive *config-typing* :: [*nat*, *s*, *v list*, *e list*, (*trust* × *t list*)] ⇒ *bool* (⊢′- - -;-;- : - *60*) **where**
⟦*store-typing* *s* 𝒮; 𝒮•*tr*•*None* ⊩-*i* *vs*;*es* : *ts*⟧ ⟹ ⊢-*i* *s*;*vs*;*es* : (*tr*,*ts*)

inductive *reduce-simple* :: [*e list*, *action*, *e list*] ⇒ *bool* ((|-|) -⇝ (|-|) *60*) **where**
— *integer unary ops*
unop-i32:(|[$*C* (*ConstInt32* *sec′* *c*), $(*Unop-i* (*T-i32* *sec*) *iop*)]|) (*Unop-i32-action* *iop*)⇝ (|[$*C* (*ConstInt32* *sec* (*app-unop-i* *iop* *c*))]|)
| *unop-i64*:(|[$*C* (*ConstInt64* *sec′* *c*), $(*Unop-i* (*T-i64* *sec*) *iop*)]|) (*Unop-i64-action* *iop*)⇝ (|[$*C* (*ConstInt64* *sec* (*app-unop-i* *iop* *c*))]|)
— *float unary ops*
| *unop-f32*:(|[$*C* (*ConstFloat32* *c*), $(*Unop-f* *T-f32* *fop*)]|) (*Unop-f32-action* *fop* *c*)⇝ (|[$*C* (*ConstFloat32* (*app-unop-f* *fop* *c*))]|)
| *unop-f64*:(|[$*C* (*ConstFloat64* *c*), $(*Unop-f* *T-f64* *fop*)]|) (*Unop-f64-action* *fop* *c*)⇝ (|[$*C* (*ConstFloat64* (*app-unop-f* *fop* *c*))]|)
— *int32 binary ops*
| *binop-i32-Some*:⟦*app-binop-i* *iop* *c1* *c2* = (*Some* *c*)⟧ ⟹ (|[$*C* (*ConstInt32* *sec′* *c1*), $*C* (*ConstInt32* *sec″* *c2*), $(*Binop-i* (*T-i32* *sec*) *iop*)]|) (*Binop-i32-Some-action* *iop* *c1* *c2*)⇝ (|[$*C* (*ConstInt32* *sec* *c*)]|)
| *binop-i32-None*:⟦*app-binop-i* *iop* *c1* *c2* = *None*⟧ ⟹ (|[$*C* (*ConstInt32* *sec′* *c1*), $*C* (*ConstInt32* *sec″* *c2*), $(*Binop-i* (*T-i32* *sec*) *iop*)]|) (*Binop-i32-None-action* *iop* *c1* *c2*)⇝ (|[*Trap*]|)
— *int64 binary ops*
| *binop-i64-Some*:⟦*app-binop-i* *iop* *c1* *c2* = (*Some* *c*)⟧ ⟹ (|[$*C* (*ConstInt64* *sec′* *c1*), $*C* (*ConstInt64* *sec″* *c2*), $(*Binop-i* (*T-i64* *sec*) *iop*)]|) (*Binop-i64-Some-action* *iop* *c1* *c2*)⇝ (|[$*C* (*ConstInt64* *sec* *c*)]|)
| *binop-i64-None*:⟦*app-binop-i* *iop* *c1* *c2* = *None*⟧ ⟹ (|[$*C* (*ConstInt64* *sec′* *c1*), $*C* (*ConstInt64* *sec″* *c2*), $(*Binop-i* (*T-i64* *sec*) *iop*)]|) (*Binop-i64-None-action* *iop* *c1* *c2*)⇝ (|[*Trap*]|)
— *float32 binary ops*
| *binop-f32-Some*:⟦*app-binop-f* *fop* *c1* *c2* = (*Some* *c*)⟧ ⟹ (|[$*C* (*ConstFloat32* *c1*), $*C* (*ConstFloat32* *c2*), $(*Binop-f* *T-f32* *fop*)]|) (*Binop-f32-Some-action* *fop* *c1* *c2*)⇝ (|[$*C* (*ConstFloat32* *c*)]|)
| *binop-f32-None*:⟦*app-binop-f* *fop* *c1* *c2* = *None*⟧ ⟹ (|[$*C* (*ConstFloat32* *c1*), $*C* (*ConstFloat32* *c2*), $(*Binop-f* *T-f32* *fop*)]|) (*Binop-f32-None-action* *fop* *c1* *c2*)⇝ (|[*Trap*]|)
— *float64 binary ops*
| *binop-f64-Some*:⟦*app-binop-f* *fop* *c1* *c2* = (*Some* *c*)⟧ ⟹ (|[$*C* (*ConstFloat64* *c1*), $*C* (*ConstFloat64* *c2*), $(*Binop-f* *T-f64* *fop*)]|) (*Binop-f64-Some-action* *fop* *c1* *c2*)⇝

⦇[$C (ConstFloat64 c)]⦈
| binop-f64-None:⟦app-binop-f fop c1 c2 = None⟧ ⟹ ⦇[$C (ConstFloat64 c1), $C (ConstFloat64 c2), $(Binop-f T-f64 fop)]⦈ (Binop-f64-None-action fop c1 c2)↝ ⦇[Trap]⦈
— testops
| testop-i32:⦇[$C (ConstInt32 sec′ c), $(Testop (T-i32 sec) testop)]⦈ (Testop-i32-action testop)↝ ⦇[$C ConstInt32 sec (wasm-bool (app-testop-i testop c))]⦈
| testop-i64:⦇[$C (ConstInt64 sec′ c), $(Testop (T-i64 sec) testop)]⦈ (Testop-i64-action testop)↝ ⦇[$C ConstInt32 sec (wasm-bool (app-testop-i testop c))]⦈
— int relops
| relop-i32:⦇[$C (ConstInt32 sec′ c1), $C (ConstInt32 sec″ c2), $(Relop-i (T-i32 sec) iop)]⦈ (Relop-i32-action iop)↝ ⦇[$C (ConstInt32 sec (wasm-bool (app-relop-i iop c1 c2)))]⦈
| relop-i64:⦇[$C (ConstInt64 sec′ c1), $C (ConstInt64 sec″ c2), $(Relop-i (T-i64 sec) iop)]⦈ (Relop-i64-action iop)↝ ⦇[$C (ConstInt32 sec (wasm-bool (app-relop-i iop c1 c2)))]⦈
— float relops
| relop-f32:⦇[$C (ConstFloat32 c1), $C (ConstFloat32 c2), $(Relop-f T-f32 fop)]⦈ (Relop-f32-action fop c1 c2)↝ ⦇[$C (ConstInt32 Public (wasm-bool (app-relop-f fop c1 c2)))]⦈
| relop-f64:⦇[$C (ConstFloat64 c1), $C (ConstFloat64 c2), $(Relop-f T-f64 fop)]⦈ (Relop-f64-action fop c1 c2)↝ ⦇[$C (ConstInt32 Public (wasm-bool (app-relop-f fop c1 c2)))]⦈
— convert
| convert-Some:⟦types-agree-insecure t1 v; cvt t2 sx v = (Some v′)⟧ ⟹ ⦇[$(C v), $(Cvtop t2 Convert t1 sx)]⦈ (Convert-Some-action t1 t2 v)↝ ⦇[$(C v′)]⦈
| convert-None:⟦types-agree-insecure t1 v; cvt t2 sx v = None⟧ ⟹ ⦇[$(C v), $(Cvtop t2 Convert t1 sx)]⦈ (Convert-None-action t1 t2 v)↝ ⦇[Trap]⦈
— reinterpret
| reinterpret:types-agree-insecure t1 v ⟹ ⦇[$(C v), $(Cvtop t2 Reinterpret t1 None)]⦈ (Reinterpret-action)↝ ⦇[$(C (wasm-deserialise (bits v) t2))]⦈
— classify
| classify:types-agree-insecure t1 v ⟹ ⦇[$(C v), $(Cvtop t2 Classify t1 None)]⦈ (Classify-action)↝ ⦇[$(C (classify v))]⦈
— declassify
| declassify:types-agree-insecure t1 v ⟹ ⦇[$(C v), $(Cvtop t2 Declassify t1 None)]⦈ (Declassify-action)↝ ⦇[$(C (declassify v))]⦈
— unreachable
| unreachable:⦇[$ Unreachable]⦈ (Unreachable-action)↝ ⦇[Trap]⦈
— nop
| nop:⦇[$ Nop]⦈ (Nop-action)↝ ⦇[]⦈
— drop
| drop:⦇[$(C v), ($ Drop)]⦈ (Drop-action)↝ ⦇[]⦈
— select
| select-false:int-eq n 0 ⟹ ⦇[$(C v1), $(C v2), $C (ConstInt32 sec n), ($ Select sec′)]⦈ (Select-action sec′ n)↝ ⦇[$(C v2)]⦈
| select-true:int-ne n 0 ⟹ ⦇[$(C v1), $(C v2), $C (ConstInt32 sec n), ($ Select sec′)]⦈ (Select-action sec′ n)↝ ⦇[$(C v1)]⦈
— block

| *block*:⟦*const-list vs*; *length vs* = *n*; *length t1s* = *n*; *length t2s* = *m*⟧ ⟹ ⦇*vs* @ [$(*Block* (*t1s* -> *t2s*) *es*)]⦈ (*Block-action*)↝ ⦇[*Label m* [] (*vs* @ ($* *es*))]⦈
— *loop*
| *loop*:⟦*const-list vs*; *length vs* = *n*; *length t1s* = *n*; *length t2s* = *m*⟧ ⟹ ⦇*vs* @ [$(*Loop* (*t1s* -> *t2s*) *es*)]⦈ (*Loop-action*)↝ ⦇[*Label n* [$(*Loop* (*t1s* -> *t2s*) *es*)] (*vs* @ ($* *es*))]⦈
— *if*
| *if-false*:*int-eq n 0* ⟹ ⦇[$*C* (*ConstInt32 sec n*), $(*If tf e1s e2s*)]⦈ (*If-false-action n*)↝ ⦇[$(*Block tf e2s*)]⦈
| *if-true*:*int-ne n 0* ⟹ ⦇[$*C* (*ConstInt32 sec n*), $(*If tf e1s e2s*)]⦈ (*If-true-action n*)↝ ⦇[$(*Block tf e1s*)]⦈
— *label*
| *label-const*:*const-list vs* ⟹ ⦇[*Label n es vs*]⦈ (*Label-const-action*)↝ ⦇*vs*⦈
| *label-trap*:⦇[*Label n es* [*Trap*]]⦈ (*Label-trap-action*)↝ ⦇[*Trap*]⦈
— *br*
| *br*:⟦*const-list vs*; *length vs* = *n*; *Lfilled i lholed* (*vs* @ [$(*Br i*)]) *LI*⟧ ⟹ ⦇[*Label n es LI*]⦈ (*Br-action*)↝ ⦇*vs* @ *es*⦈
— *br-if*
| *br-if-false*:*int-eq n 0* ⟹ ⦇[$*C* (*ConstInt32 sec n*), $(*Br-if i*)]⦈ (*Br-if-false-action n*)↝ ⦇[]⦈
| *br-if-true*:*int-ne n 0* ⟹ ⦇[$*C* (*ConstInt32 sec n*), $(*Br-if i*)]⦈ (*Br-if-true-action n*)↝ ⦇[$(*Br i*)]⦈
— *br-table*
| *br-table*:⟦*length is* > (*nat-of-int c*)⟧ ⟹ ⦇[$*C* (*ConstInt32 sec c*), $(*Br-table is i*)]⦈ (*Br-table-action c*)↝ ⦇[$(*Br* (*is*!(*nat-of-int c*)))]⦈
| *br-table-length*:⟦*length is* ≤ (*nat-of-int c*)⟧ ⟹ ⦇[$*C* (*ConstInt32 sec c*), $(*Br-table is i*)]⦈ (*Br-table-length-action c*)↝ ⦇[$(*Br i*)]⦈
— *local*
| *local-const*:⟦*const-list es*; *length es* = *n*⟧ ⟹ ⦇[*Local n i vs es*]⦈ (*Local-const-action*)↝ ⦇*es*⦈
| *local-trap*:⦇[*Local n i vs* [*Trap*]]⦈ (*Local-trap-action*)↝ ⦇[*Trap*]⦈
— *return*
| *return*:⟦*const-list vs*; *length vs* = *n*; *Lfilled j lholed* (*vs* @ [$*Return*]) *es*⟧ ⟹ ⦇[*Local n i vls es*]⦈ (*Return-action*)↝ ⦇*vs*⦈
— *tee-local*
| *tee-local*:*is-const v* ⟹ ⦇[*v*, $(*Tee-local i*)]⦈ (*Tee-local-action*)↝ ⦇[*v*, *v*, $(*Set-local i*)]⦈
| *trap*:⟦*es* ≠ [*Trap*]; *Lfilled 0 lholed* [*Trap*] *es*⟧ ⟹ ⦇*es*⦈ (*Trap-action*)↝ ⦇[*Trap*]⦈

inductive *reduce* :: [*s*, *v list*, *e list*, *action*, *nat*, *s*, *v list*, *e list*] ⇒ *bool* (⦇-;-;-⦈ -↝′- - ⦇-;-;-⦈ *60*) **where**
— *lifting basic reduction*
basic:⦇*e*⦈ *a*↝ ⦇*e′*⦈ ⟹ ⦇*s*;*vs*;*e*⦈ *a*↝-*i* ⦇*s*;*vs*;*e′*⦈
— *call*
| *call*:⦇*s*;*vs*;[$(*Call j*)]⦈ (*Call-action*)↝-*i* ⦇*s*;*vs*;[*Callcl* (*sfunc s i j*)]⦈
— *call-indirect*
| *call-indirect-Some*:⟦*stab s i* (*nat-of-int c*) = *Some cl*; *stypes s i j* = *tf*; *cl-type cl* = *tf*⟧ ⟹ ⦇*s*;*vs*;[$*C* (*ConstInt32 sec c*), $(*Call-indirect j*)]⦈ (*Call-indirect-Some-action*

c)↝-i ⦇s;vs;[Callcl cl]⦈
| call-indirect-None:⟦(stab s i (nat-of-int c) = Some cl ∧ stypes s i j ≠ cl-type cl) ∨ stab s i (nat-of-int c) = None⟧ ⟹ ⦇s;vs;[$C (ConstInt32 sec c), $(Call-indirect j)]⦈ (Call-indirect-None-action c)↝-i ⦇s;vs;[Trap]⦈
— call
| callcl-native:⟦cl = Func-native j (tr,(t1s -> t2s)) ts es; ves = ($$* vcs); length vcs = n; length ts = k; length t1s = n; length t2s = m; (n-zeros ts = zs) ⟧ ⟹ ⦇s;vs;ves @ [Callcl cl]⦈ (Callcl-native-action n)↝-i ⦇s;vs;[Local m j (vcs@zs) [$(Block ([] -> t2s) es)]]⦈
| callcl-host-Some:⟦cl = Func-host (tr,(t1s -> t2s)) f; ves = ($$* vcs); length vcs = n; length t1s = n; length t2s = m; host-apply s (t1s -> t2s) f vcs hs = Some (s', vcs')⟧ ⟹ ⦇s;vs;ves @ [Callcl cl]⦈ (Callcl-host-Some-action s vcs s' vcs' tr (t1s -> t2s) f hs)↝-i ⦇s';vs;($$* vcs')⦈
| callcl-host-None:⟦cl = Func-host (tr,(t1s -> t2s)) f; ves = ($$* vcs); length vcs = n; length t1s = n; length t2s = m⟧ ⟹ ⦇s;vs;ves @ [Callcl cl]⦈ (Callcl-host-None-action s vcs tr (t1s -> t2s) f hs)↝-i ⦇s;vs;[Trap]⦈
— get-local
| get-local:⟦length vi = j⟧ ⟹ ⦇s;(vi @ [v] @ vs);[$(Get-local j)]⦈ (Get-local-action)↝-i ⦇s;(vi @ [v] @ vs);[$(C v)]⦈
— set-local
| set-local:⟦length vi = j⟧ ⟹ ⦇s;(vi @ [v] @ vs);[$(C v'), $(Set-local j)]⦈ (Set-local-action)↝-i ⦇s;(vi @ [v'] @ vs);[]⦈
— get-global
| get-global:⦇s;vs;[$(Get-global j)]⦈ (Get-global-action)↝-i ⦇s;vs;[$ C(sglob-val s i j)]⦈
— set-global
| set-global:supdate-glob s i j v = s' ⟹ ⦇s;vs;[$(C v), $(Set-global j)]⦈ (Set-global-action)↝-i ⦇s';vs;[]⦈
— load
| load-Some:⟦smem-ind s i = Some j; ((mem s)!j) = (m,sec); load m (nat-of-int k) off (t-length t) = Some bs⟧ ⟹ ⦇s;vs;[$C (ConstInt32 sec' k), $(Load t None a off)]⦈ (Load-Some-action t (nat-of-int k) a off)↝-i ⦇s;vs;[$C (wasm-deserialise bs t)]⦈
| load-None:⟦smem-ind s i = Some j; ((mem s)!j) = (m,sec); load m (nat-of-int k) off (t-length t) = None⟧ ⟹ ⦇s;vs;[$C (ConstInt32 sec' k), $(Load t None a off)]⦈ (Load-None-action t (nat-of-int k) a off)↝-i ⦇s;vs;[Trap]⦈
— load packed
| load-packed-Some:⟦smem-ind s i = Some j; ((mem s)!j) = (m,sec); load-packed sx m (nat-of-int k) off (tp-length tp) (t-length t) = Some bs⟧ ⟹ ⦇s;vs;[$C (ConstInt32 sec' k), $(Load t (Some (tp, sx)) a off)]⦈ (Load-packed-Some-action tp sx (nat-of-int k) a off)↝-i ⦇s;vs;[$C (wasm-deserialise bs t)]⦈
| load-packed-None:⟦smem-ind s i = Some j; ((mem s)!j) = (m,sec); load-packed sx m (nat-of-int k) off (tp-length tp) (t-length t) = None⟧ ⟹ ⦇s;vs;[$C (ConstInt32 sec' k), $(Load t (Some (tp, sx)) a off)]⦈ (Load-packed-None-action tp sx (nat-of-int k) a off)↝-i ⦇s;vs;[Trap]⦈
— store
| store-Some:⟦types-agree-insecure t v; smem-ind s i = Some j; ((mem s)!j) = (m,sec); store m (nat-of-int k) off (bits v) (t-length t) = Some mem'⟧ ⟹ ⦇s;vs;[$C (ConstInt32 sec' k), $C v, $(Store t None a off)]⦈ (Store-Some-action t (nat-of-int

k) *a off*)↝-*i* ⦇*s*⦇*mem*:= ((*mem s*)[*j* := (*mem′*,*sec*)])⦈;*vs*;[]⦈
| *store-None*:⟦*types-agree-insecure t v*; *smem-ind s i* = *Some j*; ((*mem s*)!*j*) = (*m*,*sec*); *store m* (*nat-of-int k*) *off* (*bits v*) (*t-length t*) = *None*⟧ ⟹ ⦇*s*;*vs*;[$*C* (*ConstInt32 sec′ k*), $*C v*, $(*Store t None a off*)]⦈ (*Store-None-action t* (*nat-of-int k*) *a off*)↝-*i* ⦇*s*;*vs*;[*Trap*]⦈
— *store packed*
| *store-packed-Some*:⟦*types-agree-insecure t v*; *smem-ind s i* = *Some j*; ((*mem s*)!*j*) = (*m*,*sec*); *store-packed m* (*nat-of-int k*) *off* (*bits v*) (*tp-length tp*) = *Some mem′*⟧ ⟹ ⦇*s*;*vs*;[$*C* (*ConstInt32 sec′ k*), $*C v*, $(*Store t* (*Some tp*) *a off*)]⦈ (*Store-packed-Some-action t tp* (*nat-of-int k*) *a off*)↝-*i* ⦇*s*⦇*mem*:= ((*mem s*)[*j* := (*mem′*,*sec*)])⦈;*vs*;[]⦈
| *store-packed-None*:⟦*types-agree-insecure t v*; *smem-ind s i* = *Some j*; ((*mem s*)!*j*) = (*m*,*sec*); *store-packed m* (*nat-of-int k*) *off* (*bits v*) (*tp-length tp*) = *None*⟧ ⟹ ⦇*s*;*vs*;[$*C* (*ConstInt32 sec′ k*), $*C v*, $(*Store t* (*Some tp*) *a off*)]⦈ (*Store-packed-None-action t tp* (*nat-of-int k*) *a off*)↝-*i* ⦇*s*;*vs*;[*Trap*]⦈
— *current-memory*
| *current-memory*:⟦*smem-ind s i* = *Some j*; ((*mem s*)!*j*) = (*m*,*sec*); *mem-size m* = *n*⟧ ⟹ ⦇*s*;*vs*;[$(*Current-memory*)]⦈ (*Current-memory-action n*)↝-*i* ⦇*s*;*vs*;[$*C* (*ConstInt32 Public* (*int-of-nat n*))]⦈
— *grow-memory*
| *grow-memory*:⟦*smem-ind s i* = *Some j*; ((*mem s*)!*j*) = (*m*,*sec*); *mem-size m* = *n*; *mem-grow m* (*nat-of-int c*) = *mem′*⟧ ⟹ ⦇*s*;*vs*;[$*C* (*ConstInt32 sec′ c*), $(*Grow-memory*)]⦈ (*Grow-memory-Some-action n* (*nat-of-int c*))↝-*i* ⦇*s*⦇*mem*:= ((*mem s*)[*j* := (*mem′*,*sec*)])⦈;*vs*;[$*C* (*ConstInt32 Public* (*int-of-nat n*))]⦈
— *grow-memory fail*
| *grow-memory-fail*:⟦*smem-ind s i* = *Some j*; ((*mem s*)!*j*) = (*m*,*sec*); *mem-size m* = *n*⟧ ⟹ ⦇*s*;*vs*;[$*C* (*ConstInt32 sec′ c*),$(*Grow-memory*)]⦈ (*Grow-memory-None-action n* (*nat-of-int c*))↝-*i* ⦇*s*;*vs*;[$*C* (*ConstInt32 Public int32-minus-one*)]⦈

— *inductive label reduction*
| *label*:⟦⦇*s*;*vs*;*es*⦈ *a*↝-*i* ⦇*s′*;*vs′*;*es′*⦈; *Lfilled k lholed es les*; *Lfilled k lholed es′ les′*⟧ ⟹ ⦇*s*;*vs*;*les*⦈ *a*↝-*i* ⦇*s′*;*vs′*;*les′*⦈
— *inductive local reduction*
| *local*:⟦⦇*s*;*vs*;*es*⦈ *a*↝-*i* ⦇*s′*;*vs′*;*es′*⦈⟧ ⟹ ⦇*s*;*v0s*;[*Local n i vs es*]⦈ *a*↝-*j* ⦇*s′*;*v0s*;[*Local n i vs′ es′*]⦈

end

4 Host Properties

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

lemma *mem-grow-size*:
assumes *mem-grow m n* = *m′*
shows (*mem-size m* + (*64000* ∗ *n*)) = *mem-size m′*
⟨*proof*⟩

lemma *load-size*:

$(load\ m\ n\ off\ l = None) = (mem\text{-}size\ m < (off + n + l))$
⟨proof⟩

lemma *load-packed-size*:
$(load\text{-}packed\ sx\ m\ n\ off\ lp\ l = None) = (mem\text{-}size\ m < (off + n + lp))$
⟨proof⟩

lemma *store-size1*:
$(store\ m\ n\ off\ v\ l = None) = (mem\text{-}size\ m < (off + n + l))$
⟨proof⟩

lemma *store-size*:
assumes $(store\ m\ n\ off\ v\ l = Some\ m')$
shows $mem\text{-}size\ m = mem\text{-}size\ m'$
⟨proof⟩

lemma *store-packed-size1*:
$(store\text{-}packed\ m\ n\ off\ v\ l = None) = (mem\text{-}size\ m < (off + n + l))$
⟨proof⟩

lemma *store-packed-size*:
assumes $(store\text{-}packed\ m\ n\ off\ v\ l = Some\ m')$
shows $mem\text{-}size\ m = mem\text{-}size\ m'$
⟨proof⟩

axiomatization where
wasm-deserialise-type:*typeof* $(wasm\text{-}deserialise\ bs\ t) = t$

axiomatization where
host-apply-preserve-store: *list-all2 types-agree t1s vs* $\Longrightarrow$ *host-apply s* (*t1s* -> *t2s*) *f vs hs* = *Some* (s', vs') $\Longrightarrow$ *store-extension s* s'
and *host-apply-respect-type*:*list-all2 types-agree t1s vs* $\Longrightarrow$ *host-apply s* (*t1s* -> *t2s*) *f vs hs* = *Some* (s', vs') $\Longrightarrow$ *list-all2 types-agree t2s* vs'
and *host-trust-security-Some*:*store-public-agree s* s' $\Longrightarrow$ *publics-agree vs* vs' $\Longrightarrow$ *host-apply s* (*t1s* -> *t2s*) *f vs hs* = *Some* (*s-a*, *vs-a*) $\Longrightarrow$
$\exists$ s'-*a* vs'-*a*. *host-apply* s' (*t1s* -> *t2s*) *f* vs' hs' = *Some* (s'-*a*, vs'-*a*) $\wedge$
store-public-agree s-a s'-*a* $\wedge$
publics-agree vs-a vs'-*a*
and *host-trust-security-None*:*store-public-agree s* s' $\Longrightarrow$ *publics-agree vs* vs' $\Longrightarrow$ *host-apply s* (*t1s* -> *t2s*) *f vs hs* = *None* $\Longrightarrow$
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* = *ConstInt32 sec c*
⟨*proof*⟩

lemma *typeof-i64*:
assumes *typeof v* = (*T-i64 sec*)
shows $\exists$ *c. v* = *ConstInt64 sec c*
⟨*proof*⟩

lemma *typeof-f32*:
assumes *typeof v* = *T-f32*
shows $\exists$ *c. v* = *ConstFloat32 c*
⟨*proof*⟩

lemma *typeof-f64*:
assumes *typeof v* = *T-f64*
shows $\exists$ *c. v* = *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. typeof v* = *t*
⟨*proof*⟩

lemma *lfilled-collapse1*:

assumes *Lfilled n lholed* $(vs@es)$ *LI*
const-list vs
length vs $\geq l$
shows $\exists$ *lholed'*. *Lfilled n lholed'* $((drop\ (length\ vs - l)\ vs)@es)$ *LI*
⟨*proof*⟩

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

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

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

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

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

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

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

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

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

$i < length\ (inst\ s)$
shows $Some\ cl \in set\ (concat\ (s.tab\ s))$
⟨*proof*⟩

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

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

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

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

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

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

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

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

n)
⟨*proof*⟩

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

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

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

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

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

lemma *b-e-type-unop-testop*:
assumes $\mathcal{C} \vdash [e] : (ts \text{ -> } ts')$
$e = Unop\text{-}i\ t\ iop \lor e = Unop\text{-}f\ t\ fop \lor e = Testop\ t\ testop$
shows $\exists\, ts''.\ ts = ts''@[t] \land ts' = ts''@[arity\text{-}1\text{-}result\ e]$
$e = Unop\text{-}f\ t\ fop \Longrightarrow is\text{-}float\text{-}t\ t$
⟨*proof*⟩

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

lemma *b-e-type-drop*:
  assumes $\mathcal{C} \vdash [e] : (ts \text{ -> } ts')$
    $e = Drop$
  shows $\exists\, t.\ ts = ts'@[t]$
  ⟨*proof*⟩

lemma *b-e-type-select*:
  assumes $\mathcal{C} \vdash [e] : (ts \text{ -> } ts')$
    $e = Select\ sec$
  shows $\exists\, ts''\ t.\ ts = ts''@[t,t,(T\text{-}i32\ sec)] \wedge ts' = ts''@[t] \wedge (sec = Secret \longrightarrow is\text{-}secret\text{-}t\ t)$
  ⟨*proof*⟩

lemma *b-e-type-call*:
  assumes $\mathcal{C} \vdash [e] : (ts \text{ -> } ts')$
    $e = Call\ i$
  shows $i < length\ (func\text{-}t\ \mathcal{C})$
    $\exists\, tr\ ts''\ tf1\ tf2.\ trust\text{-}compat\ (trust\text{-}t\ \mathcal{C})\ tr \wedge ts = ts''@tf1 \wedge ts' = ts''@tf2 \wedge (func\text{-}t\ \mathcal{C})!i = (tr,(tf1 \text{ -> } tf2))$
  ⟨*proof*⟩

lemma *b-e-type-call-indirect*:
  assumes $\mathcal{C} \vdash [e] : (ts \text{ -> } ts')$
    $e = Call\text{-}indirect\ i$
  shows $i < length\ (types\text{-}t\ \mathcal{C})$
    $\exists\, tr\ ts''\ tf1\ tf2.\ trust\text{-}compat\ (trust\text{-}t\ \mathcal{C})\ tr \wedge ts = ts''@tf1@[(T\text{-}i32\ Public)] \wedge ts' = ts''@tf2 \wedge (types\text{-}t\ \mathcal{C})!i = (tr,(tf1 \text{ -> } tf2))$
  ⟨*proof*⟩

lemma *b-e-type-get-local*:
  assumes $\mathcal{C} \vdash [e] : (ts \text{ -> } ts')$
    $e = Get\text{-}local\ i$
  shows $\exists\, t.\ ts' = ts@[t] \wedge (local\ \mathcal{C})!i = t\ i < length(local\ \mathcal{C})$
  ⟨*proof*⟩

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

lemma *b-e-type-comp2-relift*:
assumes $\mathcal{C} \vdash [e1] : (t1s \text{ -> } ts')\ \mathcal{C} \vdash [e2] : (ts' \text{ -> } t2s)$
shows $\mathcal{S}\bullet\mathcal{C} \vdash [\$e1,\ \$e2] : (ts@t1s \text{ -> } ts@t2s)$
$\langle proof \rangle$

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

lemma *e-type-comp*:

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

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

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

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

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

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

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

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

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

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

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

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

lemma *e-type-local*:
assumes $\mathcal{S}$•$\mathcal{C}$ $\vdash$ [*Local n i vs es*] : (*ts* -> *ts'*)
shows $\exists$ *tls*. *i* < *length* (*s-inst* $\mathcal{S}$)
$\wedge$ *length tls* = *n*
$\wedge$ ($\mathcal{S}$•((*s-inst* $\mathcal{S}$)!*i*)(|*trust-t* := (*trust-t* $\mathcal{C}$), *local* := (*local* ((*s-inst* $\mathcal{S}$)!*i*)) @ (*map typeof vs*), *return* := *Some tls*|) $\vdash$ *es* : ([] -> *tls*))
$\wedge$ *ts'* = *ts* @ *tls*
⟨*proof*⟩

lemma *e-type-local-shallow*:
assumes $\mathcal{S}$•$\mathcal{C}$ $\vdash$ [*Local n i vs es*] : (*ts* -> *ts'*)
shows $\exists$ *tls*. *length tls* = *n* $\wedge$ *ts'* = *ts*@*tls* $\wedge$ ($\mathcal{S}$•(*trust-t* $\mathcal{C}$)•(*Some tls*) $\Vdash$-*i vs*;*es* : *tls*)
⟨*proof*⟩

lemma *e-type-const-unwrap*:

assumes *is-const e*
shows $\exists\, v.\ e = \$C\ v$
⟨*proof*⟩

lemma *is-const-list1*:
assumes $ves = map\ (Basic \circ EConst)\ vs$
shows *const-list ves*
⟨*proof*⟩

lemma *is-const-list*:
assumes $ves = \$\$*\ vs$
shows *const-list ves*
⟨*proof*⟩

lemma *const-list-cons-last*:
assumes $const\text{-}list\ (es@[e])$
shows *const-list es*
is-const e
⟨*proof*⟩

lemma *e-type-const1*:
assumes *is-const e*
shows $\exists\, t.\ (\mathcal{S}\bullet\mathcal{C} \vdash [e] : (ts \text{ -> } ts@[t]))$
⟨*proof*⟩

lemma *e-type-const*:
assumes *is-const e*
$\mathcal{S}\bullet\mathcal{C} \vdash [e] : (ts \text{ -> } ts')$
shows $\exists\, t.\ (ts' = ts@[t]) \wedge (\mathcal{S}\bullet\mathcal{C}' \vdash [e] : ([] \text{ -> } [t]))$
⟨*proof*⟩

lemma *const-typeof*:
assumes $\mathcal{S}\bullet\mathcal{C} \vdash [\$C\ v] : ([] \text{ -> } [t])$
shows $typeof\ v = t$
⟨*proof*⟩

lemma *e-type-const-list*:
assumes *const-list vs*
$\mathcal{S}\bullet\mathcal{C} \vdash vs : (ts \text{ -> } ts')$
shows $\exists\, tvs.\ ts' = ts\ @\ tvs \wedge length\ vs = length\ tvs \wedge (\mathcal{S}\bullet\mathcal{C}' \vdash vs : ([] \text{ -> } tvs))$
⟨*proof*⟩

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

$\wedge$ *is-const v2*
⟨*proof*⟩

lemma *e-type-const-list-cons*:
assumes *const-list vs*
$\mathcal{S}$•$\mathcal{C}$ ⊢ *vs* : ([] -> (*ts1*@*ts2*))
shows ∃ *vs1 vs2*. ($\mathcal{S}$•$\mathcal{C}$ ⊢ *vs1* : ([] -> *ts1*))
$\wedge$ ($\mathcal{S}$•$\mathcal{C}$ ⊢ *vs2* : (*ts1* -> (*ts1*@*ts2*)))
$\wedge$ *vs* = *vs1*@*vs2*
$\wedge$ *const-list vs1*
$\wedge$ *const-list vs2*
⟨*proof*⟩

lemma *e-type-const-conv-vs*:
assumes *const-list ves*
shows ∃ *vs*. *ves* = $$* *vs*
⟨*proof*⟩

lemma *types-exist-lfilled*:
assumes *Lfilled k lholed es lfilled*
$\mathcal{S}$•$\mathcal{C}$ ⊢ *lfilled* : (*ts* -> *ts'*)
shows ∃ *t1s t2s* $\mathcal{C}'$ *arb-label*. ($\mathcal{S}$•$\mathcal{C}$(|*label* := *arb-label*@(*label* $\mathcal{C}$)|) ⊢ *es* : (*t1s* -> *t2s*))
⟨*proof*⟩

lemma *types-exist-lfilled-weak*:
assumes *Lfilled k lholed es lfilled*
$\mathcal{S}$•$\mathcal{C}$ ⊢ *lfilled* : (*ts* -> *ts'*)
shows ∃ *t1s t2s* $\mathcal{C}'$ *arb-label arb-return*. ($\mathcal{S}$•$\mathcal{C}$(|*label* := *arb-label*, *return* := *arb-return*|) ⊢ *es* : (*t1s* -> *t2s*))
⟨*proof*⟩

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

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

lemma *store-typing-imp-mem-agree-Some*:
  assumes *store-typing s* $\mathcal{S}$
    $i <$ *length* (*s-inst* $\mathcal{S}$)
    smem-ind s i = *Some j*
  shows $j <$ *length* (*s-mem* $\mathcal{S}$)
    mem-agree ((*mem s*)!j) ((*s-mem* $\mathcal{S}$)!j)
    $\exists\, x.$ ((*s-mem* $\mathcal{S}$)!j) = x $\wedge$ (*memory* ((*s-inst* $\mathcal{S}$)!i)) = *Some x*
⟨*proof*⟩

lemma *store-typing-imp-mem-agree-None*:
  assumes *store-typing s* $\mathcal{S}$
    $i <$ *length* (*s-inst* $\mathcal{S}$)
    smem-ind s i = *None*
  shows (*memory* ((*s-inst* $\mathcal{S}$)!i)) = *None*
⟨*proof*⟩

lemma *store-typing-imp-mem-agree-inst*:
  assumes *store-typing s* $\mathcal{S}$
    $i <$ *length* (*s-inst* $\mathcal{S}$)
  shows *option-projr* (*memory* ((*s-inst* $\mathcal{S}$)!i)) = *map-option* ($\lambda j.$ *snd* ((*mem s*)!j)) (*smem-ind s i*)
⟨*proof*⟩

lemma *store-preserved-mem*:
  assumes *store-typing s* $\mathcal{S}$
    $s' = s(\!|s.mem := (s.mem\ s)[i := (mem',sec)]|\!)$
    mem-size mem' $\geq$ *mem-size orig-mem*
    ((*s.mem s*)!i) = (*orig-mem*, *sec*)
  shows *store-typing s'* $\mathcal{S}$
⟨*proof*⟩

lemma *types-agree-imp-e-typing*:
  assumes *types-agree t v*
  shows $\mathcal{S}$•$\mathcal{C}$ $\vdash$ [\$*C v*] : ([] -> [t])
  ⟨*proof*⟩

lemma *list-types-agree-imp-e-typing*:
  assumes *list-all2 types-agree ts vs*
  shows $\mathcal{S}$•$\mathcal{C}$ $\vdash$ \$\$* *vs* : ([] -> *ts*)
  ⟨*proof*⟩

lemma *b-e-typing-imp-list-types-agree*:
  assumes $\mathcal{C}$ $\vdash$ (*map* ($\lambda v.$ *C v*) *vs*) : (*ts'* -> *ts'*@*ts*)
  shows *list-all2 types-agree ts vs*
  ⟨*proof*⟩

lemma *e-typing-imp-list-types-agree*:
  assumes $\mathcal{S}$•$\mathcal{C}$ $\vdash$ (\$\$* *vs*) : (*ts'* -> *ts'*@*ts*)

shows *list-all2 types-agree ts vs*
⟨*proof*⟩

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 *Lfilled k lfilled es les*
Lfilled k lfilled es les'
shows *les = les'*
⟨*proof*⟩

lemma *b-e-typing-trust-compat*:
assumes $\mathcal{C} \vdash$ *es : tf*
trust-compat tr (trust-t $\mathcal{C}$*)*
shows $\mathcal{C}$(|*trust-t := tr*|) $\vdash$ *es : tf*
⟨*proof*⟩

lemma *e-typing-s-typing-trust-compat*:
$\mathcal{S}$•$\mathcal{C}$ $\vdash$ *es : tf* $\Longrightarrow$ *trust-compat tr (trust-t* $\mathcal{C}$*)* $\Longrightarrow$ $\mathcal{S}$•$\mathcal{C}$(|*trust-t := tr*|) $\vdash$ *es : tf*
$\mathcal{S}$•*tr'*•*r* $\Vdash$*-i vs;es : ts* $\Longrightarrow$ *trust-compat tr tr'* $\Longrightarrow$ $\mathcal{S}$•*tr*•*r* $\Vdash$*-i vs;es : ts*
⟨*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'*
⟨*proof*⟩

lemma *store-preserved1*:
assumes (|*s;vs;es*|) *a⇝-i* (|*s';vs';es'*|)
store-typing s $\mathcal{S}$
$\mathcal{S}$•$\mathcal{C}$ $\vdash$ *es : (ts -> ts')*
$\mathcal{C}$ = ((*s-inst* $\mathcal{S}$)!*i*)(|*trust-t := tr, local := local ((s-inst* $\mathcal{S}$*)!i) @ (map typeof vs), label := arb-label, return := arb-return*|)
i < length (s-inst $\mathcal{S}$*)*
shows *store-typing s'* $\mathcal{S}$
⟨*proof*⟩

lemma *store-preserved*:
assumes (|*s;vs;es*|) *a⇝-i* (|*s';vs';es'*|)

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

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

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

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

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

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

⟨*proof*⟩

lemma *types-preserved-binop-relop*:
assumes $(\![[\$C\ v1,\ \$C\ v2,\ \$e]]\!)\ a\leadsto (\![[\$C\ v']]\!)$
$\mathcal{S}\cdot\mathcal{C} \vdash [\$C\ v1,\ \$C\ v2,\ \$e] : (ts\ \text{->}\ ts')$
$e = \textit{Binop-i}\ t\ iop \lor e = \textit{Binop-f}\ t\ fop \lor e = \textit{Relop-i}\ t\ irop \lor e = \textit{Relop-f}$
$t\ frop$
shows $\mathcal{S}\cdot\mathcal{C} \vdash [\$C\ v'] : (ts\ \text{->}\ ts')$
⟨*proof*⟩

lemma *types-preserved-drop*:
assumes $(\![[\$C\ v,\ \$e]]\!)\ a\leadsto (\![[]]\!)$
$\mathcal{S}\cdot\mathcal{C} \vdash [\$C\ v,\ \$e] : (ts\ \text{->}\ ts')$
$(e = (\textit{Drop}))$
shows $\mathcal{S}\cdot\mathcal{C} \vdash [] : (ts\ \text{->}\ ts')$
⟨*proof*⟩

lemma *typeof-select*:
assumes $\mathcal{S}\cdot\mathcal{C} \vdash [\$C\ v1,\ \$C\ v2,\ \$C\ vn,\ \$e] : (ts\ \text{->}\ ts')$
$(e = \textit{Select}\ sec)$
shows $\textit{t-sec}\ (\textit{typeof}\ vn) = sec$
$\textit{typeof}\ v1 = \textit{typeof}\ v2$
$sec = \textit{Secret} \longrightarrow \textit{is-secret-t}\ (\textit{typeof}\ v1)$
$sec = \textit{Secret} \longrightarrow \textit{is-secret-t}\ (\textit{typeof}\ v2)$
⟨*proof*⟩

lemma *types-preserved-select*:
assumes $(\![[\$C\ v1,\ \$C\ v2,\ \$C\ vn,\ \$e]]\!)\ a\leadsto (\![[\$C\ v3]]\!)$
$\mathcal{S}\cdot\mathcal{C} \vdash [\$C\ v1,\ \$C\ v2,\ \$C\ vn,\ \$e] : (ts\ \text{->}\ ts')$
$(e = \textit{Select}\ sec)$
shows $\mathcal{S}\cdot\mathcal{C} \vdash [\$C\ v3] : (ts\ \text{->}\ ts')$
⟨*proof*⟩

lemma *types-preserved-block*:
assumes $(\!|vs\ @\ [\$\textit{Block}\ (tn\ \text{->}\ tm)\ es]|\!)\ a\leadsto (\![[\textit{Label}\ m\ []\ (vs\ @\ (\$*\ es))]]\!)$
$\mathcal{S}\cdot\mathcal{C} \vdash vs\ @\ [\$\textit{Block}\ (tn\ \text{->}\ tm)\ es] : (ts\ \text{->}\ ts')$
$\textit{const-list}\ vs$
$\textit{length}\ vs = n$
$\textit{length}\ tn = n$
$\textit{length}\ tm = m$
shows $\mathcal{S}\cdot\mathcal{C} \vdash [\textit{Label}\ m\ []\ (vs\ @\ (\$*\ es))] : (ts\ \text{->}\ ts')$
⟨*proof*⟩

lemma *typeof-if*:
assumes $\mathcal{S}\cdot\mathcal{C} \vdash [\$C\ \textit{ConstInt32}\ sec\ n,\ \$\textit{If}\ tf\ e1s\ e2s] : (ts\ \text{->}\ ts')$
shows $sec = \textit{Public}$
⟨*proof*⟩

lemma *types-preserved-if*:

assumes $(\!|[\$C\ \mathit{ConstInt32}\ \mathit{sec}\ n,\ \$\mathit{If}\ \mathit{tf}\ \mathit{e1s}\ \mathit{e2s}]|\!)\ a\leadsto\ (\!|[\$\mathit{Block}\ \mathit{tf}\ \mathit{es}']|\!)$
$\mathcal{S}\cdot\mathcal{C} \vdash [\$C\ \mathit{ConstInt32}\ \mathit{sec}\ n,\ \$\mathit{If}\ \mathit{tf}\ \mathit{e1s}\ \mathit{e2s}] : (\mathit{ts}\ \text{->}\ \mathit{ts}')$
shows $\mathcal{S}\cdot\mathcal{C} \vdash [\$\mathit{Block}\ \mathit{tf}\ \mathit{es}'] : (\mathit{ts}\ \text{->}\ \mathit{ts}')$
⟨*proof*⟩

lemma *types-preserved-tee-local*:
assumes $(\!|[v,\ \$\mathit{Tee\text{-}local}\ i]|\!)\ a\leadsto\ (\!|[v,\ v,\ \$\mathit{Set\text{-}local}\ i]|\!)$
$\mathcal{S}\cdot\mathcal{C} \vdash [v,\ \$\mathit{Tee\text{-}local}\ i] : (\mathit{ts}\ \text{->}\ \mathit{ts}')$
is-const v
shows $\mathcal{S}\cdot\mathcal{C} \vdash [v,\ v,\ \$\mathit{Set\text{-}local}\ i] : (\mathit{ts}\ \text{->}\ \mathit{ts}')$
⟨*proof*⟩

lemma *types-preserved-loop*:
assumes $(\!|\mathit{vs}\ @\ [\$\mathit{Loop}\ (\mathit{t1s}\ \text{->}\ \mathit{t2s})\ \mathit{es}]|\!)\ a\leadsto\ (\!|[\mathit{Label}\ n\ [\$\mathit{Loop}\ (\mathit{t1s}\ \text{->}\ \mathit{t2s})\ \mathit{es}]$ $(\mathit{vs}\ @\ (\$*\ \mathit{es}))]|\!)$
$\mathcal{S}\cdot\mathcal{C} \vdash \mathit{vs}\ @\ [\$\mathit{Loop}\ (\mathit{t1s}\ \text{->}\ \mathit{t2s})\ \mathit{es}] : (\mathit{ts}\ \text{->}\ \mathit{ts}')$
const-list vs
$\mathit{length}\ \mathit{vs} = n$
$\mathit{length}\ \mathit{t1s} = n$
$\mathit{length}\ \mathit{t2s} = m$
shows $\mathcal{S}\cdot\mathcal{C} \vdash [\mathit{Label}\ n\ [\$\mathit{Loop}\ (\mathit{t1s}\ \text{->}\ \mathit{t2s})\ \mathit{es}]\ (\mathit{vs}\ @\ (\$*\ \mathit{es}))] : (\mathit{ts}\ \text{->}\ \mathit{ts}')$
⟨*proof*⟩

lemma *types-preserved-label-value*:
assumes $(\!|[\mathit{Label}\ n\ \mathit{es0}\ \mathit{vs}]|\!)\ a\leadsto\ (\!|\mathit{vs}|\!)$
$\mathcal{S}\cdot\mathcal{C} \vdash [\mathit{Label}\ n\ \mathit{es0}\ \mathit{vs}] : (\mathit{ts}\ \text{->}\ \mathit{ts}')$
const-list vs
shows $\mathcal{S}\cdot\mathcal{C} \vdash \mathit{vs} : (\mathit{ts}\ \text{->}\ \mathit{ts}')$
⟨*proof*⟩

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

lemma *types-preserved-br-if*:
assumes $(\!|[\$C\ \mathit{ConstInt32}\ \mathit{sec}\ n,\ \$\mathit{Br\text{-}if}\ i]|\!)\ a\leadsto\ (\!|e|\!)$
$\mathcal{S}\cdot\mathcal{C} \vdash [\$C\ \mathit{ConstInt32}\ \mathit{sec}\ n,\ \$\mathit{Br\text{-}if}\ i] : (\mathit{ts}\ \text{->}\ \mathit{ts}')$
$e = [\$\mathit{Br}\ i] \lor e = []$
shows $\mathcal{S}\cdot\mathcal{C} \vdash e : (\mathit{ts}\ \text{->}\ \mathit{ts}')$
⟨*proof*⟩

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

lemma *types-preserved-br-table*:
assumes $(\!|[\$C\ \mathit{ConstInt32}\ \mathit{sec}\ c,\ \$\mathit{Br\text{-}table}\ \mathit{is}\ i]|\!)\ a\leadsto\ (\!|[\$\mathit{Br}\ i']|\!)$

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

lemma *types-preserved-local-const*:
assumes (|[*Local n i vs es*]|) *a*⇝ (|*es*|)
$\mathcal{S}\cdot\mathcal{C} \vdash$ [*Local n i vs es*] : (*ts* -> *ts'*)
const-list es
shows $\mathcal{S}\cdot\mathcal{C} \vdash$ *es*: (*ts* -> *ts'*)
⟨*proof*⟩

lemma *typing-map-typeof*:
assumes *ves* = \$\$* *vs*
$\mathcal{S}\cdot\mathcal{C} \vdash$ *ves* : ([] -> *tvs*)
shows *tvs* = *map typeof vs*
⟨*proof*⟩

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

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

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

lemma *types-preserved-callcl-host-some*:
assumes $\mathcal{S}\cdot\mathcal{C} \vdash$ *ves* @ [*Callcl cl*] : (*ts* -> *ts'*)
cl = *Func-host* (*tr*,(*t1s* -> *t2s*)) *f*
ves = $$* *vcs*
length vcs = *n*
length t1s = *n*
length t2s = *m*
host-apply s (*t1s* -> *t2s*) *f vcs hs* = *Some* (*s'*, *vcs'*)
store-typing s $\mathcal{S}$
shows $\mathcal{S}\cdot\mathcal{C} \vdash$ $$* *vcs'* : (*ts* -> *ts'*)
⟨*proof*⟩

lemma *types-imp-concat*:
assumes $\mathcal{S}\cdot\mathcal{C} \vdash$ *es* @ [*e*] @ *es'* : (*ts* -> *ts'*)
$\bigwedge$*tes tes'*. (($\mathcal{S}\cdot\mathcal{C} \vdash$ [*e*] : (*tes* -> *tes'*)) $\Longrightarrow$ ($\mathcal{S}\cdot\mathcal{C} \vdash$ [*e'*] : (*tes* -> *tes'*)))
shows $\mathcal{S}\cdot\mathcal{C} \vdash$ *es* @ [*e'*] @ *es'* : (*ts* -> *ts'*)
⟨*proof*⟩

lemma *type-const-return*:
assumes *Lfilled i lholed* (*vs* @ [$*Return*]) *LI*
(*return* $\mathcal{C}$) = *Some tcs*
length tcs = *length vs*
$\mathcal{S}\cdot\mathcal{C} \vdash$ *LI* : (*ts* -> *ts'*)
const-list vs
shows $\mathcal{S}\cdot\mathcal{C}' \vdash$ *vs* : ([] -> *tcs*)
⟨*proof*⟩

lemma *types-preserved-return*:
assumes ⦇[*Local n i vls LI*]⦈ *a*⇝ ⦇*ves*⦈
$\mathcal{S}\cdot\mathcal{C} \vdash$ [*Local n i vls LI*] : (*ts* -> *ts'*)
const-list ves
length ves = *n*
Lfilled j lholed (*ves* @ [$*Return*]) *LI*
shows $\mathcal{S}\cdot\mathcal{C} \vdash$ *ves* : (*ts* -> *ts'*)
⟨*proof*⟩

lemma *type-const-br*:
assumes *Lfilled i lholed* (*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* -> *ts'*)
const-list vs
shows $\mathcal{S}\cdot\mathcal{C}' \vdash$ *vs* : ([] -> *tcs*)
⟨*proof*⟩

lemma *types-preserved-br*:
assumes ⦇[*Label n es0 LI*]⦈ *a*⇝ ⦇*vs* @ *es0*⦈

$\mathcal{S}{\cdot}\mathcal{C} \vdash$ [*Label n es0 LI*] : (*ts* -> *ts'*)
const-list vs
length vs = *n*
Lfilled i lholed (*vs* @ [\$*Br i*]) *LI*
shows $\mathcal{S}{\cdot}\mathcal{C} \vdash$ (*vs* @ *es0*) : (*ts* -> *ts'*)
⟨*proof*⟩

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

lemma *types-preserved-b-e1*:
assumes (|*es*|) *a*⇝ (|*es'*|)
store-typing s $\mathcal{S}$
$\mathcal{S}{\cdot}\mathcal{C} \vdash$ *es* : (*ts* -> *ts'*)
shows $\mathcal{S}{\cdot}\mathcal{C} \vdash$ *es'* : (*ts* -> *ts'*)
⟨*proof*⟩

lemma *types-preserved-b-e*:
assumes (|*es*|) *a*⇝ (|*es'*|)
store-typing s $\mathcal{S}$
$\mathcal{S}{\cdot}$*tr*${\cdot}$*None* ⊩-*i vs*;*es* : *ts*
shows $\mathcal{S}{\cdot}$*tr*${\cdot}$*None* ⊩-*i vs*;*es'* : *ts*
⟨*proof*⟩

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

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

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

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

⟨*proof*⟩

lemma *types-preserved-load*:
assumes $\mathcal{S}\bullet\mathcal{C} \vdash [\$C\ ConstInt32\ sec\ k,\ \$Load\ t\ tp\ a\ off] : (ts \text{ -> } ts')$
$typeof\ v = t$
shows $\mathcal{S}\bullet\mathcal{C} \vdash [\$C\ v] : (ts \text{ -> } ts')$
$sec = Public$
⟨*proof*⟩

lemma *types-preserved-get-local*:
assumes $\mathcal{S}\bullet\mathcal{C} \vdash [\$Get\text{-}local\ i] : (ts \text{ -> } ts')$
$length\ vi = i$
$(local\ \mathcal{C}) = map\ typeof\ (vi\ @\ [v]\ @\ vs)$
shows $\mathcal{S}\bullet\mathcal{C} \vdash [\$C\ v] : (ts \text{ -> } ts')$
⟨*proof*⟩

lemma *types-preserved-set-local*:
assumes $\mathcal{S}\bullet\mathcal{C} \vdash [\$C\ v',\ \$Set\text{-}local\ i] : (ts \text{ -> } ts')$
$length\ vi = i$
$(local\ \mathcal{C}) = map\ typeof\ (vi\ @\ [v]\ @\ vs)$
shows $(\mathcal{S}\bullet\mathcal{C} \vdash [] : (ts \text{ -> } ts')) \wedge map\ typeof\ (vi\ @\ [v]\ @\ vs) = map\ typeof\ (vi\ @\ [v']\ @\ vs)$
⟨*proof*⟩

lemma *types-preserved-get-global*:
assumes $typeof\ (sglob\text{-}val\ s\ i\ j) = tg\text{-}t\ (global\ \mathcal{C}\ !\ j)$
$\mathcal{S}\bullet\mathcal{C} \vdash [\$Get\text{-}global\ j] : (ts \text{ -> } ts')$
shows $\mathcal{S}\bullet\mathcal{C} \vdash [\$C\ sglob\text{-}val\ s\ i\ j] : (ts \text{ -> } ts')$
⟨*proof*⟩

lemma *lholed-same-type*:
assumes $Lfilled\ k\ lholed\ es\ les$
$Lfilled\ k\ lholed\ es'\ les'$
$\mathcal{S}\bullet\mathcal{C} \vdash les : (ts \text{ -> } ts')$
$\bigwedge arb\text{-}labs\ ts\ ts'.$
$\mathcal{S}\bullet(\mathcal{C}(\!|label := arb\text{-}labs@(label\ \mathcal{C})|\!)) \vdash es : (ts \text{ -> } ts')$
$\Longrightarrow \mathcal{S}\bullet(\mathcal{C}(\!|label := arb\text{-}labs@(label\ \mathcal{C})|\!)) \vdash es' : (ts \text{ -> } ts')$
shows $(\mathcal{S}\bullet\mathcal{C} \vdash les' : (ts \text{ -> } ts'))$
⟨*proof*⟩

lemma *types-preserved-e1*:
assumes $(\!|s;vs;es|\!)\ a\leadsto\text{-}i\ (\!|s';vs';es'|\!)$
$store\text{-}typing\ s\ \mathcal{S}$
$tvs = map\ typeof\ vs$
$i < length\ (inst\ s)$
$\mathcal{C} = ((s\text{-}inst\ \mathcal{S})!i)(\!|trust\text{-}t := tr,\ local := (local\ ((s\text{-}inst\ \mathcal{S})!i)\ @\ tvs),\ label := arb\text{-}labs,\ return := arb\text{-}return|\!)$
$\mathcal{S}\bullet\mathcal{C} \vdash es : (ts \text{ -> } ts')$
shows $(\mathcal{S}\bullet\mathcal{C} \vdash es' : (ts \text{ -> } ts')) \wedge (map\ typeof\ vs = map\ typeof\ vs')$

⟨*proof*⟩

lemma *types-preserved-e*:
assumes $(\!| s;vs;es |\!)$ $a\leadsto$-i $(\!| s';vs';es' |\!)$
store-typing s $\mathcal{S}$
$\mathcal{S}\cdot tr\cdot None \Vdash$-$i$ $vs;es : ts$
shows $\mathcal{S}\cdot tr\cdot None \Vdash$-$i$ $vs';es' : ts$
⟨*proof*⟩

6.2 Progress

lemma *const-list-no-progress*:
assumes *const-list es*
shows $\neg(\!| s;vs;es |\!)$ $a\leadsto$- i $(\!| s';vs';es' |\!)$
⟨*proof*⟩

lemma *empty-no-progress*:
assumes $es = []$
shows $\neg(\!| s;vs;es |\!)$ $a\leadsto$- i $(\!| s';vs';es' |\!)$
⟨*proof*⟩

lemma *trap-no-progress*:
assumes $es = [Trap]$
shows $\neg(\!| s;vs;es |\!)$ $a\leadsto$- i $(\!| s';vs';es' |\!)$
⟨*proof*⟩

lemma *terminal-no-progress*:
assumes *const-list es* $\vee$ $es = [Trap]$
shows $\neg(\!| s;vs;es |\!)$ $a\leadsto$- i $(\!| s';vs';es' |\!)$
⟨*proof*⟩

lemma *progress-L0*:
assumes $(\!| s;vs;es |\!)$ $a\leadsto$- i $(\!| s';vs';es' |\!)$
const-list cs
shows $(\!| s;vs;cs@es@es\text{-}c |\!)$ $a\leadsto$- i $(\!| s';vs';cs@es'@es\text{-}c |\!)$
⟨*proof*⟩

lemma *progress-L0-left*:
assumes $(\!| s;vs;es |\!)$ $a\leadsto$- i $(\!| s';vs';es' |\!)$
const-list cs
shows $(\!| s;vs;cs@es |\!)$ $a\leadsto$- i $(\!| s';vs';cs@es' |\!)$
⟨*proof*⟩

lemma *progress-L0-trap*:
assumes *const-list cs*
$cs \neq [] \vee es \neq []$
shows $\exists\, a.\ (\!| s;vs;cs@[Trap]@es |\!)$ $a\leadsto$- i $(\!| s;vs;[Trap] |\!)$
⟨*proof*⟩

lemma *progress-LN*:
  assumes $(Lfilled\ j\ lholed\ [\$Br\ (j{+}k)]\ es)$
  $\mathcal{S}{\bullet}\mathcal{C} \vdash es : ([] \text{ -> } ts)$
  $(label\ \mathcal{C})!k = tvs$
  shows $\exists\, lholed'\ vs\ \mathcal{C}'.\ (Lfilled\ j\ lholed'\ (vs@[\$Br\ (j{+}k)])\ es)$
  $\wedge\ (\mathcal{S}{\bullet}\mathcal{C}' \vdash vs : ([] \text{ -> } tvs))$
  $\wedge\ const\text{-}list\ vs$
  ⟨*proof*⟩

lemma *progress-LN-return*:
  assumes $(Lfilled\ j\ lholed\ [\$Return]\ es)$
  $\mathcal{S}{\bullet}\mathcal{C} \vdash es : ([] \text{ -> } ts)$
  $(return\ \mathcal{C}) = Some\ tvs$
  shows $\exists\, lholed'\ vs\ \mathcal{C}'.\ (Lfilled\ j\ lholed'\ (vs@[\$Return])\ es)$
  $\wedge\ (\mathcal{S}{\bullet}\mathcal{C}' \vdash vs : ([] \text{ -> } tvs))$
  $\wedge\ const\text{-}list\ vs$
  ⟨*proof*⟩

lemma *progress-LN1*:
  assumes $(Lfilled\ j\ lholed\ [\$Br\ (j{+}k)]\ es)$
  $\mathcal{S}{\bullet}\mathcal{C} \vdash es : (ts \text{ -> } ts')$
  shows $length\ (label\ \mathcal{C}) > k$
  ⟨*proof*⟩

lemma *progress-LN2*:
  assumes $(Lfilled\ j\ lholed\ e1s\ lfilled)$
  shows $\exists\, lfilled'.\ (Lfilled\ j\ lholed\ e2s\ lfilled')$
  ⟨*proof*⟩

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

lemma *const-of-i32*:
  assumes $const\text{-}list\ cs$
  $\mathcal{S}{\bullet}\mathcal{C} \vdash cs : ([] \text{ -> } [(T\text{-}i32\ sec)])$
  shows $\exists\, c.\ cs = [\$C\ ConstInt32\ sec\ c]$
⟨*proof*⟩

lemma *const-of-i64*:
  assumes $const\text{-}list\ cs$
  $\mathcal{S}{\bullet}\mathcal{C} \vdash cs : ([] \text{ -> } [(T\text{-}i64\ sec)])$
  shows $\exists\, c.\ cs = [\$C\ ConstInt64\ sec\ c]$
⟨*proof*⟩

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$\quad\quad cs\text{-}es \neq [Trap]$
$\quad\quad \neg$ *const-list* (*cs-es*)
$\quad\quad$ *store-typing s* $\mathcal{S}$
shows $\exists\, a\ s'\ vs'\ es'.\ (\!|s;vs;cs\text{-}es|\!)\ a\rightsquigarrow\text{-}i\ (\!|s';vs';es'|\!)$
$\langle proof \rangle$

end

7 Soundness Theorems

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

theorem *preservation*:
assumes $\vdash\text{-}i\ s;vs;es : (tr,ts)$
$\quad\quad (\!|s;vs;es|\!)\ a\rightsquigarrow\text{-}i\ (\!|s';vs';es'|\!)$
shows $\vdash\text{-}i\ s';vs';es' : (tr,ts)$
$\langle proof \rangle$

theorem *progress*:
assumes $\vdash\text{-}i\ s;vs;es : (tr,ts)$
shows $const\text{-}list\ es \vee es = [Trap] \vee (\exists\, a\ s'\ vs'\ es'.\ (\!|s;vs;es|\!)\ a\rightsquigarrow\text{-}i\ (\!|s';vs';es'|\!))$
$\langle proof \rangle$

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* ⇒ *ct* ⇒ *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* ⇒ *ct list* ⇒ *bool* **where**
ct-list-eq ct1s ct2s = *list-all2 ct-eq ct1s ct2s*

definition *ct-prefix* :: *ct list* ⇒ *ct list* ⇒ *bool* **where**
ct-prefix xs ys = (∃ *as bs*. *ys* = *as*@*bs* ∧ *ct-list-eq as xs*)

definition *ct-suffix* :: *ct list* ⇒ *ct list* ⇒ *bool* **where**
ct-suffix xs ys = (∃ *as bs*. *ys* = *as*@*bs* ∧ *ct-list-eq bs xs*)

lemma *ct-eq-commute*:
assumes *ct-eq x y*
shows *ct-eq y x*
⟨*proof*⟩

lemma *ct-eq-flip*: $ct\text{-}eq^{-1-1}$ = *ct-eq*
⟨*proof*⟩

lemma *exists-secret*: ∃ *t*. *is-secret-t t*
⟨*proof*⟩

lemma *ct-eq-common-tsome*: *ct-eq x y* = (∃ *t*. *ct-eq x* (*TSome t*) ∧ *ct-eq* (*TSome t*) *y*)
⟨*proof*⟩

lemma *ct-list-eq-commute*:
assumes *ct-list-eq xs ys*
shows *ct-list-eq ys xs*
⟨*proof*⟩

lemma *ct-list-eq-refl*: *ct-list-eq xs xs*
⟨*proof*⟩

lemma *ct-list-eq-length*:
assumes *ct-list-eq xs ys*
shows *length xs* = *length ys*
⟨*proof*⟩

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

shows *ct-list-eq* $(xs@xs')$ $(ys@ys')$
⟨*proof*⟩

lemma *ct-list-eq-ts-conv-eq*:
ct-list-eq (*to-ct-list ts*) (*to-ct-list ts'*) = (*ts* = *ts'*)
⟨*proof*⟩

lemma *ct-list-eq-exists*: $\exists$ *ys*. *ct-list-eq xs* (*to-ct-list ys*)
⟨*proof*⟩

lemma *ct-list-eq-common-tsome-list*:
ct-list-eq xs ys = ($\exists$ *zs*. *ct-list-eq xs* (*to-ct-list zs*) $\wedge$ *ct-list-eq* (*to-ct-list zs*) *ys*)
⟨*proof*⟩

lemma *ct-list-eq-cons-ct-list*:
assumes *ct-list-eq* (*to-ct-list as*) (*xs* @ *ys*)
shows $\exists$ *bs bs'*. *as* = *bs* @ *bs'* $\wedge$ *ct-list-eq* (*to-ct-list bs*) *xs* $\wedge$ *ct-list-eq* (*to-ct-list bs'*) *ys*
⟨*proof*⟩

lemma *ct-list-eq-cons-ct-list1*:
assumes *ct-list-eq* (*to-ct-list as*) (*xs* @ (*to-ct-list ys*))
shows $\exists$ *bs*. *as* = *bs* @ *ys* $\wedge$ *ct-list-eq* (*to-ct-list bs*) *xs*
⟨*proof*⟩

lemma *ct-list-eq-shared*:
assumes *ct-list-eq xs* (*to-ct-list as*)
ct-list-eq ys (*to-ct-list as*)
shows *ct-list-eq xs ys*
⟨*proof*⟩

lemma *ct-list-eq-take*:
assumes *ct-list-eq xs ys*
shows *ct-list-eq* (*take n xs*) (*take n ys*)
⟨*proof*⟩

lemma *ct-prefixI* [*intro?*]:
assumes *ys* = *as* @ *zs*
ct-list-eq as xs
shows *ct-prefix xs ys*
⟨*proof*⟩

lemma *ct-prefixE* [*elim?*]:
assumes *ct-prefix xs ys*
obtains *as zs* **where** *ys* = *as* @ *zs* *ct-list-eq as xs*
⟨*proof*⟩

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

⟨*proof*⟩

lemma *ct-prefix-nil*:*ct-prefix* [] *xs*
$\neg$*ct-prefix* (*x* # *xs*) []
⟨*proof*⟩

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

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*)
⟨*proof*⟩

lemma *ct-suffix-to-ct-prefix* [*code*]: *ct-suffix* *xs* *ys* = *ct-prefix* (*rev* *xs*) (*rev* *ys*)
⟨*proof*⟩

lemma *inj-TSome*: *inj* *TSome*
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *ct-suffixI* [*intro?*]:
assumes *ys* = *as* @ *zs*
ct-list-eq *zs* *xs*
shows *ct-suffix* *xs* *ys*
⟨*proof*⟩

lemma *ct-suffixE* [*elim?*]:
assumes *ct-suffix* *xs* *ys*
obtains *as* *zs* **where** *ys* = *as* @ *zs* *ct-list-eq* *zs* *xs*
⟨*proof*⟩

lemma *ct-suffix-nil*: *ct-suffix* [] *ts*
⟨*proof*⟩

lemma *ct-suffix-refl*: *ct-suffix* *ts* *ts*
⟨*proof*⟩

lemma *ct-suffix-length*:
assumes *ct-suffix* *ts* *ts'*
shows *length* *ts* $\leq$ *length* *ts'*
⟨*proof*⟩

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*: ∃ *ts-c*. *ct-suffix x1* (*to-ct-list ts-c*)
  ⟨*proof*⟩

lemma *ct-suffix-ct-list-eq-exists*:
  assumes *ct-suffix x1 x2*
  shows ∃ *ts-c*. *ct-suffix x1* (*to-ct-list ts-c*) ∧ *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 ∃ *as bs*. *zs* = *as*@*bs* ∧ *ct-list-eq ys* (*to-ct-list bs*) ∧ *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 ∃ *as b*. *zs* = *as*@[*b*] ∧ *ct-eq y* (*TSome b*) ∧ *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 ∃ *as*. *zs* = *as*@*ys* ∧ *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'*)

$\langle proof \rangle$

lemma *ct-suffix-extend-any1*:
assumes *ct-suffix xs ys*
length xs $<$ *length ys*
shows *ct-suffix* (*TAny#xs*) *ys*
$\langle proof \rangle$

lemma *ct-suffix-singleton-any*: *ct-suffix* [*TAny*] [*t*]
$\langle proof \rangle$

lemma *ct-suffix-cons-it*: *ct-suffix xs* (*xs'@xs*)
$\langle proof \rangle$

lemma *ct-suffix-singleton*:
assumes *length cts* $>$ *0*
shows *ct-suffix* [*TAny*] *cts*
$\langle proof \rangle$

lemma *ct-suffix-less*:
assumes *ct-suffix* (*xs@xs'*) *ys*
shows *ct-suffix xs' ys*
$\langle proof \rangle$

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

lemma *ct-suffix-shared*:
assumes *ct-suffix cts* (*to-ct-list ts*)
ct-suffix cts' (*to-ct-list ts*)
shows *ct-suffix cts cts'* $\vee$ *ct-suffix cts' cts*
$\langle proof \rangle$

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

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

| *consume - -* = *Bot*

fun *produce* :: *checker-type* ⇒ *checker-type* ⇒ *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*

fun *type-update* :: *checker-type* ⇒ *ct list* ⇒ *checker-type* ⇒ *checker-type* **where**
type-update curr-type cons prods = *produce* (*consume curr-type cons*) *prods*

fun *ens-sec-ct* :: *sec* ⇒ *ct* ⇒ *ct* **where**
ens-sec-ct Secret TAny = *TSecret*
| *ens-sec-ct - ct* = *ct*

fun *select-return-top* :: *sec* ⇒ *ct list* ⇒ *ct* ⇒ *ct* ⇒ *checker-type* **where**
select-return-top sec ts ct1 TAny = (*if* (*sec* = *Secret* ⟶ *can-secret-ct ct1*)
then TopType ((*take* (*length ts* − *3*) *ts*) @ [*ens-sec-ct sec ct1*])
else Bot)
| *select-return-top sec ts TAny ct2* = (*if* (*sec* = *Secret* ⟶ *can-secret-ct ct2*)
then TopType ((*take* (*length ts* − *3*) *ts*) @ [*ens-sec-ct sec ct2*])
else Bot)
| *select-return-top sec ts ct1 TSecret* = (*if* (*can-secret-ct ct1*)
then TopType ((*take* (*length ts* − *3*) *ts*) @ [*ens-sec-ct sec ct1*])
else Bot)
| *select-return-top sec ts TSecret ct2* = (*if* (*can-secret-ct ct2*)
then TopType ((*take* (*length ts* − *3*) *ts*) @ [*ct2*])
else Bot)
| *select-return-top sec ts* (*TSome t1*) (*TSome t2*) = (*if* (*t1* = *t2* ∧ (*sec* = *Secret* ⟶ *is-secret-t t1*))
then (*TopType* ((*take* (*length ts* − *3*) *ts*) @ [*TSome t1*]))
else Bot)

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

⟨*proof*⟩

fun *type-update-select* :: *sec* ⇒ *checker-type* ⇒ *checker-type* **where**
type-update-select sec (*Type ts*) = (*if* (*length ts* ≥ *3* ∧ (*ts*!(*length ts*−*2*)) = (*ts*!(*length ts*−*3*)) ∧ (*sec* = *Secret* ⟶ *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 ⇒ *TopType* [*sec-ct sec*]
| *Suc 0* ⇒ *type-update* (*TopType ts*) [*TSome* (*T-i32 sec*)] (*TopType* [*sec-ct sec*])
| *Suc* (*Suc 0*) ⇒ *type-update* (*TopType ts*) [*sec-ct sec*, *TSome* (*T-i32 sec*)] (*TopType* [*ens-sec-ct sec* (*ts*!(*length ts*−*2*))])
| - ⇒ *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* ⇒ *t list* ⇒ *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' ≠ *Bot*
shows (*sec* = *Secret* ⟶ *can-secret-ct ct1*) ∧ (*sec* = *Secret* ⟶ *can-secret-ct ct2*)
⟨*proof*⟩

lemma *produce-not-bot*:
assumes *produce a b* = *c*
c ≠ *Bot*
shows *a* ≠ *Bot b* ≠ *Bot*
⟨*proof*⟩

lemma *consume-type*:
assumes *consume* (*Type ts*) *ts'* = *c-t*
c-t ≠ *Bot*
shows ∃ *ts''*. *ct-list-eq* (*to-ct-list ts*) ((*to-ct-list ts''*)@*ts'*) ∧ *c-t* = *Type ts''*
⟨*proof*⟩

lemma *consume-top-geq*:
assumes *consume* (*TopType ts*) *ts'* = *c-t*
length ts ≥ *length ts'*
c-t ≠ *Bot*
shows (∃ *as bs*. *ts* = *as*@*bs* ∧ *ct-list-eq bs ts'* ∧ *c-t* = *TopType as*)
⟨*proof*⟩

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

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

lemma *produce-type-type*:
  assumes $produce\ xs\ cons = (Type\ tm)$
  shows $\exists\, tn.\ xs = Type\ tn$
  ⟨*proof*⟩

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

lemma *produce-weaken-type*:
  assumes $produce\ (Type\ tn)\ cons = (Type\ tm)$
  shows $produce\ (Type\ (ts@tn))\ cons = (Type\ (ts@tm))$
  ⟨*proof*⟩

lemma *produce-nil*: $produce\ ts\ (Type\ []) = ts$
  ⟨*proof*⟩

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

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

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

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

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

shows $\exists$ *ts-c. c-types-agree ts ts-c*
⟨*proof*⟩

lemma *consume-c-types-agree*:
assumes *consume* (*Type ts*) *cts* = (*Type ts′*)
c-types-agree ctn ts
shows $\exists$ *c-t′. consume ctn cts* = *c-t′* $\wedge$ *c-types-agree c-t′ ts′*
⟨*proof*⟩

lemma *type-update-type*:
assumes *type-update* (*Type ts*) (*to-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*))
⟨*proof*⟩

lemma *type-update-empty*: *type-update ts cons* (*Type* []) = *consume ts cons*
⟨*proof*⟩

lemma *type-update-top-top*:
assumes *type-update* (*TopType ts*) (*to-ct-list cons*) (*Type prods*) = (*TopType ts′*)
c-types-agree (*TopType ts′*) *t-ag*
shows *ct-suffix* (*to-ct-list prods*) *ts′*
$\exists$ *t-ag′. t-ag* = *t-ag′*@*prods* $\wedge$ *c-types-agree* (*TopType ts*) (*t-ag′*@*cons*)
⟨*proof*⟩

lemma *type-update-select-length0*:
assumes *type-update-select sec* (*TopType cts*) = *tm*
length cts = *0*
tm $\neq$ *Bot*
shows *tm* = *TopType* [*sec-ct sec*]
⟨*proof*⟩

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

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

$sec) \wedge tm = TopType\ [ens\text{-}sec\text{-}ct\ sec\ t1]$
⟨*proof*⟩

lemma *type-update-select-length3*:
assumes $type\text{-}update\text{-}select\ sec\ (TopType\ cts) = (TopType\ ctm)$
$length\ cts \geq 3$
shows $\exists\, cts'\ ct1\ ct2\ ct3.\ cts = cts'@[ct1,\ ct2,\ ct3] \wedge ct\text{-}eq\ ct3\ (TSome\ (T\text{-}i32\ sec))$
$\wedge\ ct\text{-}eq\ ct1\ (sec\text{-}ct\ sec) \wedge ct\text{-}eq\ ct2\ (sec\text{-}ct\ sec)$
⟨*proof*⟩

lemma *type-update-select-type-length3*:
assumes $type\text{-}update\text{-}select\ sec\ (Type\ tn) = (Type\ tm)$
shows $\exists\, t\ ts'.\ tn = ts'@[t,\ t,\ (T\text{-}i32\ sec)]$
⟨*proof*⟩

lemma *select-return-top-exists*:
assumes $select\text{-}return\text{-}top\ sec\ cts\ c1\ c2 = ctm$
$ctm \neq Bot$
shows $\exists\, xs.\ ctm = TopType\ xs$
⟨*proof*⟩

lemma *type-update-select-top-exists*:
assumes $type\text{-}update\text{-}select\ sec\ xs = (TopType\ tm)$
shows $\exists\, tn.\ xs = TopType\ tn$
⟨*proof*⟩

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

lemma *ens-sec-ct-imp-ct-eq*:
assumes $ct\text{-}eq\ (ens\text{-}sec\text{-}ct\ sec\ ct)\ ct'$
shows $ct\text{-}eq\ ct\ ct'$
⟨*proof*⟩

lemma *ens-sec-ct-imp-ct-eq-sec*:
assumes $ct\text{-}eq\ ct\ ct'$
$sec = Secret \longrightarrow can\text{-}secret\text{-}ct\ ct'$
shows $ct\text{-}eq\ (ens\text{-}sec\text{-}ct\ sec\ ct)\ ct'$
⟨*proof*⟩

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* ⟶ *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 ≠ *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* ⇒ *t* ⇒ *sx option* ⇒ *bool* **where**
convert-cond t1 t2 sx = ((*t1* ≠ *t2*) ∧ (*t-sec t1* = *t-sec t2*) ∧ (*sx* = *None*) = ((*is-float-t t1* ∧ *is-float-t t2*)
∨ (*is-int-t t1* ∧ *is-int-t t2* ∧ (*t-length t1* < *t-length t2*))))

fun *same-lab-h* :: *nat list* ⇒ (*t list*) *list* ⇒ *t list* ⇒ (*t list*) *option* **where**
same-lab-h [] - *ts* = *Some ts*
| *same-lab-h* (*i*#*is*) *lab-c ts* = (*if i* ≥ *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* ⇒ (*t list*) *list* ⇒ (*t list*) *option* **where**
same-lab [] *lab-c* = *None*
| *same-lab* (*i*#*is*) *lab-c* = (*if i* ≥ *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* (λ*i*. *i* < *length ls* ∧ *ls*!*i* = *ts*) *ils* ∧ *ts'* = *ts*

⟨*proof*⟩

lemma *same-lab-conv-list-all*:
assumes *same-lab ils ls = Some ts*
shows *list-all* ($\lambda i.$ *i < length ls* $\wedge$ *ls!i = ts*) *ils*
⟨*proof*⟩

lemma *list-all-conv-same-lab-h*:
assumes *list-all* ($\lambda i.$ *i < 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 < 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 $\mathcal{C}$ *es* (*tn* -> *tm*) = *c-types-agree* (*check* $\mathcal{C}$ *es* (*Type tn*)) *tm*
| *check* $\mathcal{C}$ *es ts* = (*case es of*
[] $\Rightarrow$ *ts*
| (*e#es*) $\Rightarrow$ (*case ts of*
Bot $\Rightarrow$ *Bot*
| - $\Rightarrow$ *check* $\mathcal{C}$ *es* (*check-single* $\mathcal{C}$ *e ts*)))

| *check-single* $\mathcal{C}$ (*C v*) *ts* = *type-update ts* [] (*Type* [*typeof v*])
| *check-single* $\mathcal{C}$ (*Unop-i t -*) *ts* = (*if is-int-t t*
then type-update ts [*TSome t*] (*Type* [*t*])
else Bot)
| *check-single* $\mathcal{C}$ (*Unop-f t -*) *ts* = (*if is-float-t t*
then type-update ts [*TSome t*] (*Type* [*t*])
else Bot)
| *check-single* $\mathcal{C}$ (*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* $\mathcal{C}$ (*Binop-f t -*) *ts* = (*if is-float-t t*
then type-update ts [*TSome t*, *TSome t*] (*Type* [*t*])
else Bot)
| *check-single* $\mathcal{C}$ (*Testop t -*) *ts* = (*if is-int-t t*
then type-update ts [*TSome t*] (*Type* [*T-i32* (*t-sec t*)])
else Bot)
| *check-single* $\mathcal{C}$ (*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)

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

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

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

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

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

| *check-single* $\mathcal{C}$ (*Unreachable*) *ts* = *type-update ts* [] (*TopType* [])
| *check-single* $\mathcal{C}$ (*Nop*) *ts* = *ts*
| *check-single* $\mathcal{C}$ (*Drop*) *ts* = *type-update ts* [*TAny*] (*Type* [])
| *check-single* $\mathcal{C}$ (*Select sec*) *ts* = *type-update-select sec ts*

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

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

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

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

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

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

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

| *check-single $\mathcal{C}$ (Return) ts = (case (return $\mathcal{C}$) of*
None ⇒ Bot
| *Some tls ⇒ type-update ts (to-ct-list tls) (TopType []))*

| *check-single $\mathcal{C}$ (Call i) ts = (if i < length (func-t $\mathcal{C}$)*
then (case ((func-t $\mathcal{C}$)!i) of
(tr,(tn -> tm)) ⇒ if (trust-compat (trust-t $\mathcal{C}$) tr)
then type-update ts (to-ct-list tn) (Type tm)
else Bot)
else Bot)

| *check-single $\mathcal{C}$ (Call-indirect i) ts = (if (table $\mathcal{C}$) ≠ None ∧ i < length (types-t $\mathcal{C}$)*
then (case ((types-t $\mathcal{C}$)!i) of
(tr,(tn -> tm)) ⇒ if (trust-compat (trust-t $\mathcal{C}$) tr)
then type-update ts (to-ct-list (tn@[T-i32 Public])) (Type tm)
else Bot)
else Bot)

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

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

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

| *check-single $\mathcal{C}$ (Get-global i) ts = (if i < length (global $\mathcal{C}$)*

then type-update ts [] (*Type* [*tg-t* ((*global* $\mathcal{C}$)!*i*)])
else Bot)

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

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

| *check-single* $\mathcal{C}$ (*Store t tp a off*) *ts* =
(*case* (*memory* $\mathcal{C}$) *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* $\mathcal{C}$ *Current-memory ts* = (*if* (*memory* $\mathcal{C}$) ≠ *None*
then type-update ts [] (*Type* [*T-i32 Public*])
else Bot)

| *check-single* $\mathcal{C}$ *Grow-memory ts* = (*if* (*memory* $\mathcal{C}$) ≠ *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*
$\mathcal{C}$ ⊢ [*e*] : (*t-in* -> *t-out*)
shows ∃ *tn*. *c-types-agree* (*Type x1*) *tn* ∧ $\mathcal{C}$ ⊢ [*e*] : (*tn* -> *tm*)

$\langle proof \rangle$

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

lemma *b-e-check-single-top-not-bot-sound*:
assumes *type-update ts* (*to-ct-list t-in*) (*TopType* []) = *ts'*
$ts \neq Bot$
$ts' \neq Bot$
shows $\exists\, tn.$ *c-types-agree ts tn* $\wedge$ *suffix t-in tn*
$\langle proof \rangle$

lemma *b-e-check-single-type-not-bot-sound*:
assumes *type-update ts* (*to-ct-list t-in*) (*Type t-out*) = *ts'*
$ts \neq Bot$
$ts' \neq Bot$
c-types-agree ts' tm
$\mathcal{C} \vdash [e] : (t\text{-}in \text{ -> } t\text{-}out)$
shows $\exists\, tn.$ *c-types-agree ts tn* $\wedge\ \mathcal{C} \vdash [e] : (tn \text{ -> } tm)$
$\langle proof \rangle$

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

lemma *b-e-check-single-sound-binop-relop*:
assumes *check-single* $\mathcal{C}$ *e tn'* = *tm'*
((*e* = *Binop-i t iop* $\wedge$ *is-int-t t* $\wedge$ (*is-secret-t t* $\longrightarrow$ *safe-binop-i iop*))
$\vee$ (*e* = *Binop-f t fop* $\wedge$ *is-float-t t*)
$\vee$ (*e* = *Relop-i t irop* $\wedge$ *is-int-t t*)
$\vee$ (*e* = *Relop-f t frop* $\wedge$ *is-float-t t*))

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

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

10.2 Completeness

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

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

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

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

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

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

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

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

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

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

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

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

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

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

theorem *b-e-typing-equiv-b-e-type-checker*:
shows ($\mathcal{C} \vdash$ *es* : (*tn* -> *tm*)) = (*b-e-type-checker* $\mathcal{C}$ *es* (*tn* -> *tm*))
⟨*proof*⟩

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 *mem-size* (*fst m*) = *mem-size* (*fst m'*)
⟨*proof*⟩

lemma *store-public-agree-smem-ind-eq*:
assumes *store-public-agree s s'*
shows (*smem-ind s i*) = (*smem-ind s' i*)
⟨*proof*⟩

lemma *store-public-agree-sfunc-eq*:
assumes *store-public-agree s s'*
shows (*sfunc s i j*) = (*sfunc s' i j*)
⟨*proof*⟩

lemma *store-public-agree-stab-eq*:
assumes *store-public-agree s s'*
shows (*stab s i j*) = (*stab s' i j*)
⟨*proof*⟩

lemma *store-public-agree-sglob-ind-eq*:
assumes *store-public-agree s s'*
(*sglob-ind s i j*) < *length* (*globs s*)
shows (*sglob-ind s i j*) = (*sglob-ind s' i j*)
⟨*proof*⟩

lemma *store-public-agree-sglob-val-agree*:
assumes *store-public-agree s s'*
(*sglob-ind s i j*) < *length* (*globs s*)
shows *public-agree* (*sglob-val s i j*) (*sglob-val s' i j*)
⟨*proof*⟩

lemma *store-public-agree-stypes-eq*:
assumes *store-public-agree s s'*
shows (*stypes s i j*) = (*stypes s' i j*)
⟨*proof*⟩

lemma *store-agree-imp-callcl-cond*:
assumes *store-public-agree s s'*
(*stab s i* (*nat-of-int c*) = *Some cl* ∧ *stypes s i j* ≠ *cl-type cl*) ∨ *stab s i* (*nat-of-int c*) = *None*
shows (*stab s' i* (*nat-of-int c*) = *Some cl* ∧ *stypes s' i j* ≠ *cl-type cl*) ∨ *stab s' i* (*nat-of-int c*) = *None*
⟨*proof*⟩

lemma *public-agree-imp-typeof*:
assumes *public-agree v v'*
shows *typeof v* = *typeof v'*

⟨*proof*⟩

lemma *not-typeof-imp-no-public-agree*:
assumes $typeof\ v \neq typeof\ v'$
shows $\neg public\text{-}agree\ v\ v'$
⟨*proof*⟩

lemma *publics-agree-imp-typeof*:
assumes $publics\text{-}agree\ vs\ vs'$
shows $map\ typeof\ vs = map\ typeof\ vs'$
⟨*proof*⟩

lemma *public-agree-imp-types-agree-insecure*:
assumes $types\text{-}agree\text{-}insecure\ t\ v$
$public\text{-}agree\ v\ v'$
shows $types\text{-}agree\text{-}insecure\ t\ v'$
⟨*proof*⟩

lemma *public-agree-imp-types-agree*:
assumes $types\text{-}agree\ t\ v$
$public\text{-}agree\ v\ v'$
shows $types\text{-}agree\ t\ v'$
⟨*proof*⟩

lemma *publics-agree-nil1*:
assumes $publics\text{-}agree\ []\ vs$
shows $vs = []$
⟨*proof*⟩

lemma *publics-agree-nil2*:
assumes $publics\text{-}agree\ vs\ []$
shows $vs = []$
⟨*proof*⟩

lemma *public-agree-refl*: $public\text{-}agree\ v\ v$
⟨*proof*⟩

lemma *public-agree-public-i32*:
assumes $public\text{-}agree\ (ConstInt32\ sec\ c)\ v$
shows $\exists\, c.\ v = (ConstInt32\ sec\ c)$
⟨*proof*⟩

lemma *public-agree-public-i64*:
assumes $public\text{-}agree\ (ConstInt64\ sec\ c)\ v$
shows $\exists\, c.\ v = (ConstInt64\ sec\ c)$
⟨*proof*⟩

lemma *public-agree-public-f32*:
assumes $public\text{-}agree\ (ConstFloat32\ c)\ v$

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

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

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

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

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

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

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

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

lemmas $\mathit{expr\text{-}public\text{-}agree\text{-}refl} = \mathit{expr\text{-}public\text{-}agree.intros}(1)$

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

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

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

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*
⟨*proof*⟩

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

lemma *expr-public-agree-symm*:
assumes *expr-public-agree e e'*
shows *expr-public-agree e' e*
⟨*proof*⟩

lemma *exprs-public-agree-symm*:
assumes *exprs-public-agree es es'*
shows *exprs-public-agree es' es*
⟨*proof*⟩

lemma *store-public-agree-refl*: *store-public-agree s s*
⟨*proof*⟩

lemma *store-public-agree-symm*:
assumes *store-public-agree s s'*
shows *store-public-agree s' s*
⟨*proof*⟩

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

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'*
⟨*proof*⟩

lemma *expr-public-agree-block*:
assumes *expr-public-agree* (\$*Block tf es*) *les*
shows $\exists\, es'.\ les = (\$Block\ tf\ es') \wedge$ *exprs-public-agree* $(\${*}es)$ $(\${*}es')$
⟨*proof*⟩

lemma *expr-public-agree-loop*:
assumes *expr-public-agree* (\$*Loop tf es*) *les*
shows $\exists\, es'.\ les = (\$Loop\ tf\ es') \wedge$ *exprs-public-agree* $(\${*}es)$ $(\${*}es')$
⟨*proof*⟩

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* ($\$*es2$) ($\$*es2'$)
  ⟨*proof*⟩

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'*
  ⟨*proof*⟩

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''*
  ⟨*proof*⟩

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*
  ⟨*proof*⟩

lemma *exprs-public-agree-imp-expr-public-agree*:
  assumes *exprs-public-agree* [*e1*] [*e2*]
  shows *expr-public-agree e1 e2*
  ⟨*proof*⟩

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''*
  ⟨*proof*⟩

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''*
  ⟨*proof*⟩

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'
⟨*proof*⟩

lemma *exprs-public-agree-imp-publics-agree1-const0*:
assumes *exprs-public-agree* [*e*] *es'*
shows $\exists$ *e'*. *es'* = [*e'*] $\wedge$ *expr-public-agree e e'*
⟨*proof*⟩

lemma *b-e-exprs-public-agree-imp-publics-agree1-const0*:
assumes *exprs-public-agree* ($\$$∗[*b-e*]) *es'*
shows $\exists$ *b-e'*. *es'* = [$\$$*b-e'*] $\wedge$ *expr-public-agree* ($\$$*b-e*) ($\$$*b-e'*)
⟨*proof*⟩

lemma *exprs-public-agree-trap-imp-is-trap*:
assumes *exprs-public-agree* [*Trap*] *es*
shows *es* = [*Trap*]
⟨*proof*⟩

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

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'*] $\wedge$
public-agree v1 v1' $\wedge$
public-agree v2 v2' $\wedge$
expr-public-agree e e'
⟨*proof*⟩

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'*] $\wedge$
public-agree v1 v1' $\wedge$
public-agree v2 v2' $\wedge$
public-agree v3 v3' $\wedge$
expr-public-agree e e'
⟨*proof*⟩

lemma *publics-agree-imp-exprs-public-agree-cons*:
assumes *public-agree v v'*
exprs-public-agree es es'
shows *exprs-public-agree* (($\$$*C v*)#*es*) (($\$$*C v'*)#*es'*)
⟨*proof*⟩

lemma *publics-agree-imp-exprs-public-agree*:
assumes *publics-agree ves ves'*
exprs-public-agree es es'

shows *exprs-public-agree* ((\$\$∗ *ves*)@*es*) ((\$\$∗ *ves'*)@*es'*)
⟨*proof*⟩

lemma *expr-public-agree-const*:
assumes *expr-public-agree e e'*
is-const e
shows *is-const e'*
⟨*proof*⟩

lemma *exprs-public-agree-const-list*:
assumes *exprs-public-agree es es'*
const-list es
shows *const-list es'*
⟨*proof*⟩

lemma *exprs-public-agree-basic*:
assumes *exprs-public-agree* (\$∗ *ves*) *es'*
shows ∃ *ves'*. *es'* = (\$∗ *ves'*)
⟨*proof*⟩

lemma *exprs-public-agree-app3*:
assumes *exprs-public-agree* (*vs* @ *es* @ *es'*) *les*
shows ∃ *vs-a es-a es'-a*. *les* = *vs-a* @ *es-a* @ *es'-a* ∧
exprs-public-agree vs vs-a ∧
exprs-public-agree es es-a ∧
exprs-public-agree es' es'-a
⟨*proof*⟩

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

lemma *exprs-public-agree-imp-lholed-public-agree*:
assumes *Lfilled k lholed es les*
exprs-public-agree les les'
shows ∃ *lholed' es'*. *lholed-public-agree lholed lholed'* ∧
exprs-public-agree es es' ∧
Lfilled k lholed' es' les'
⟨*proof*⟩

lemma *lholed-public-agree-imp-exprs-public-agree*:
assumes *lholed-public-agree lholed lholed'*
Lfilled k lholed es les
exprs-public-agree es es'
shows ∃ *les'*. *Lfilled k lholed' es' les'* ∧ *exprs-public-agree les les'*
⟨*proof*⟩

method *solve-exprs-public-agree-imp-b-e-typing-trivial* =
  (*match* **premises in** *A*:*exprs-public-agree* ($\$*$ [*b-e*]) ($\$*$ *bes'*)
          and *B*:$\mathcal{C}$ ⊢ [*b-e*] : *tf*
     for *b-e bes'* $\mathcal{C}$ *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 $\mathcal{C}$ ⊢ *bes* : *tf*
      exprs-public-agree ($\$*$*bes*) ($\$*$*bes'*)
  shows $\mathcal{C}$ ⊢ *bes'* : *tf*
  ⟨*proof*⟩

lemma *exprs-public-agree-imp-e-typing-s-typing*:
  $\mathcal{S}$•$\mathcal{C}$ ⊢ *es* : (*ts* -> *ts'*) ⟹ *exprs-public-agree es es'* ⟹ $\mathcal{S}$•$\mathcal{C}$ ⊢ *es'* : (*ts* -> *ts'*)
  $\mathcal{S}$•*tr*•*rs* ⊩-*i vs*;*es* : *ts'* ⟹ *publics-agree vs vs'* ⟹ *exprs-public-agree es es'* ⟹ $\mathcal{S}$•*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′′*
  ⟨*proof*⟩

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

lemma *config-indistinguishable-refl*:*config-indistinguishable c c*
  ⟨*proof*⟩

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

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

lemma *equivp-config-indistinguishable*:*equivp config-indistinguishable*
  ⟨*proof*⟩

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

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

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

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 $((s,vs,es),i) \sim\text{-}cp\ ((s',vs',es'),i')$
⟨*proof*⟩

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

⟨*proof*⟩

definition *config-inst-length* :: (*s* × *v list* × *e list*) ⇒ *nat* **where**
config-inst-length c = *length* (*inst* (*fst c*))

quotient-type *config-untrusted-quot* = ((*s* × *v list* × *e list*) × *nat*) / *partial*:*config-untrusted-equiv*
⟨*proof*⟩

lift-definition *config-untrusted-quot-inst-length* :: *config-untrusted-quot* ⇒ *nat* **is** (λ(*c*,*i*). *length* (*inst* (*fst c*)))
⟨*proof*⟩

lift-definition *config-untrusted-quot-store-typing* :: *config-untrusted-quot* ⇒ *s-context* ⇒ *bool* **is** (λ(*c*,*i*) 𝒮. *store-typing* (*fst c*) 𝒮)
⟨*proof*⟩

lift-definition *config-untrusted-quot-e-typing* :: [*s-context*, *t-context*, *config-untrusted-quot*, *tf*] ⇒ *bool* **is** (λ𝒮 𝒞 (*c*,*i*) *tf*. (𝒮•𝒞 ⊢ (*snd* (*snd c*)) : *tf*))
⟨*proof*⟩

lift-definition *config-untrusted-quot-s-typing* :: [*s-context*, *trust*, (*t list*) *option*, *config-untrusted-quot*, *t list*] ⇒ *bool* **is** (λ𝒮 *tr rs* (*c*,*i*) *ts*. (𝒮•*tr*•*rs* ⊩-*i* (*fst* (*snd c*));(*snd* (*snd c*)) : *ts*))
⟨*proof*⟩

lift-definition *config-untrusted-quot-config-typing* :: [*config-untrusted-quot*, *trust* × *t list*] ⇒ *bool* **is** (λ((*s*,*vs*,*es*),*i*) *ts*. (⊢-*i s*;*vs*;*es* : *ts*))
⟨*proof*⟩

end

12 Security Proofs

theory *Wasm-Secret* **imports** *Wasm-Secret-Aux AFP/Coinductive/Coinductive HOL−Library.BNF-Corec* **begin**

inductive *action-indistinguishable* :: *action* ⇒ *action* ⇒ *bool* (- ∼′-*a* - *60*) **where**
refl:*a* ∼-*a a*
| *binop-32*:*safe-binop-i iop* ⟹ (*Binop-i32-Some-action iop c1 c2*) ∼-*a* (*Binop-i32-Some-action iop c1′ c2′*)
| *binop-64*:*safe-binop-i iop* ⟹ (*Binop-i64-Some-action iop c1 c2*) ∼-*a* (*Binop-i64-Some-action iop c1′ c2′*)
| *select*:(*Select-action Secret c1*) ∼-*a* (*Select-action Secret c2*)
| *host-Some*:⟦*store-public-agree s s′*; *publics-agree vcs vcs′*; *store-public-agree s-o s′-o*; *publics-agree vcs-o vcs′-o*⟧ ⟹ (*Callcl-host-Some-action s vcs s-o vcs-o Untrusted tf f hs*) ∼-*a* (*Callcl-host-Some-action s′ vcs′ s′-o vcs′-o Untrusted tf f hs′*)
| *host-None*:⟦*store-public-agree s s′*; *publics-agree vcs vcs′*⟧ ⟹ (*Callcl-host-None-action s vcs Untrusted tf f hs*) ∼-*a* (*Callcl-host-None-action s′ vcs′ Untrusted tf f hs′*)
| *convert-Some*:⟦*is-int-t t1*; *is-int-t t2*; *types-agree t1 v*; *public-agree v v′*⟧ ⟹

$(Convert\text{-}Some\text{-}action\ t1\ t2\ v) \sim\text{-}a\ (Convert\text{-}Some\text{-}action\ t1\ t2\ v')$
$|\ convert\text{-}None{:}[\![is\text{-}int\text{-}t\ t1;\ is\text{-}int\text{-}t\ t2;\ types\text{-}agree\ t1\ v;\ public\text{-}agree\ v\ v']\!] \Longrightarrow$
$(Convert\text{-}None\text{-}action\ t1\ t2\ v) \sim\text{-}a\ (Convert\text{-}None\text{-}action\ t1\ t2\ v')$

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* ⇒ *observation* ($\$\mathcal{A}$ - *60*) **where**
abs-obs a ≡ *abs-observation a*

inductive *reduction-actions* :: [*s*, *v list*, *e list*, *nat*, *action list*] ⇒ *bool* (*r′-actions* (|-;-;-|) - - *60*) **where**
$[\![const\text{-}list\ es \lor es = [Trap]]\!] \Longrightarrow r\text{-}actions\ (|s;vs;es|)\ i\ []$
$|\ [\![(|s;vs;es|)\ a{\leadsto}\text{-}i\ (|s';vs';es'|);\ r\text{-}actions\ (|s';vs';es'|)\ i\ as]\!] \Longrightarrow r\text{-}actions\ (|s;vs;es|)\ i\ (a\#as)$

inductive *reduce-weight* :: [*s*, *v list*, *e list*, *nat*, *nat*, *s*, *v list*, *e list*] ⇒ *bool* ((|-;-;-|) |-|⇝′- - (|-;-;-|) *60*) **where**
$(|s;vs;es|)\ a{\leadsto}\text{-}i\ (|s';vs';es'|) \Longrightarrow (|s;vs;es|)\ |(weight\ a)|{\leadsto}\text{-}i\ (|s';vs';es'|)$

inductive *reduction-weight* :: [*s*, *v list*, *e list*, *nat*, *nat*] ⇒ *bool* (*r′-weight* (|-;-;-|) - - *60*) **where**
$[\![const\text{-}list\ es \lor es = [Trap]]\!] \Longrightarrow r\text{-}weight\ (|s;vs;es|)\ i\ 0$
$|\ [\![(|s;vs;es|)\ |w|{\leadsto}\text{-}i\ (|s';vs';es'|);\ r\text{-}weight\ (|s';vs';es'|)\ i\ w']\!] \Longrightarrow r\text{-}weight\ (|s;vs;es|)\ i\ (w+w')$

lemma *r-actions-imp-r-weight*:
assumes *r-actions* (|*s*;*vs*;*es*|) *i as*
shows *r-weight* (|*s*;*vs*;*es*|) *i* (*sum-list* (*map weight as*))
⟨*proof*⟩

lemma *memories-public-agree-helper*:
assumes *smem-ind s i* = *Some j*
store-public-agree s s'
store-typing s $\mathcal{S}$
i < *length* (*inst s*)
s.mem s ! *j* = (*m*, *sec*)
shows *smem-ind s' i* = *Some j*
j < *length* (*s.mem s'*)
memories-public-agree (*s.mem s*) (*s.mem s'*)
memory-public-agree ((*s.mem s*)!*j*) ((*s.mem s'*)!*j*)
∃ *m'*. *s.mem s'* ! *j* = (*m'*, *sec*)
⟨*proof*⟩

lemma *load-helper*:
assumes *smem-ind s i* = *Some j*
s.mem s ! *j* = (*m*, *sec*)
store-typing s $\mathcal{S}$
i < *length* (*inst s*)
$\mathcal{C}$ = (*s-inst* $\mathcal{S}$! *i*)⦇*trust-t* := *tr*, *local* := *local* (*s-inst* $\mathcal{S}$! *i*) @ *tvs*, *label* := *arb-labs*, *return* := *arb-return*⦈
$\mathcal{S}$•$\mathcal{C}$ ⊢ [$*C ConstInt32 sec' k*, $*Load t tp a off*] : (*ts* -> *ts'*)
shows *t-sec t* = *sec*
⟨*proof*⟩

lemma *store-helper*:
assumes *smem-ind s i* = *Some j*
s.mem s ! *j* = (*m*, *sec*)
exprs-public-agree [$*C ConstInt32 sec' k*, $*C v*, $*Store t tp a off*] *es'*
store-public-agree s s'
store-typing s $\mathcal{S}$
i < *length* (*inst s*)
$\mathcal{C}$ = (*s-inst* $\mathcal{S}$! *i*)⦇*trust-t* := *tr*, *local* := *local* (*s-inst* $\mathcal{S}$! *i*) @ *tvs*, *label* := *arb-labs*, *return* := *arb-return*⦈
$\mathcal{S}$•$\mathcal{C}$ ⊢ [$*C ConstInt32 sec' k*, $*C v*, $*Store t tp a off*] : (*ts* -> *ts'*)
shows *t-sec t* = *sec*
sec' = *Public*
types-agree t v
∃ *v' v''*. *es'*= [$*C v'*, $*C v''*, $*Store t tp a off*] ∧
v' = (*ConstInt32 sec' k*) ∧
public-agree v v''
smem-ind s' i = *Some j*
j < *length* (*s.mem s'*)
memories-public-agree (*s.mem s*) (*s.mem s'*)
memory-public-agree ((*s.mem s*)!*j*) ((*s.mem s'*)!*j*)
∃ *m'*. *s.mem s'* ! *j* = (*m'*, *sec*)
⟨*proof*⟩

lemma *load-m-imp-load-m'*:
assumes *memory-public-agree* ((*s.mem s*)!*j*) ((*s.mem s'*)!*j*)

$s.mem\ s\ !\ j = (m,\ sec)$
$s.mem\ s'\ !\ j = (m',\ sec)$
$load\ m\ n\ off\ l = Some\ bs$
shows $\exists\, bs'.\ load\ m'\ n\ off\ l = Some\ bs'$
⟨*proof*⟩

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* $sx\ m'\ n\ off\ lp\ l = Some\ bs'$
⟨*proof*⟩

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*)
⟨*proof*⟩

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*)
⟨*proof*⟩

lemma *binop-i-secret-imp-binop-i-some*:
assumes *safe-binop-i iop*
shows $\exists\, c.$ *app-binop-i iop c1 c2* = *Some c*
⟨*proof*⟩

lemma *cvtop-secret-imp-cvt-some*:
assumes $\mathcal{S}{\cdot}\mathcal{C} \vdash$ [\$*C v*, \$*Cvtop t2 Convert t1 sx*] : (*ts* -> *ts′*)
is-secret-t (*typeof v*)
shows $\exists\, v'.$ *cvt t2 sx v* = *Some v′*

⟨*proof*⟩

lemma *publics-agree-imp-reduce-simple*:
assumes $(|es|)\ a \rightsquigarrow (|es\text{-}a|)$
exprs-public-agree es es'
$\mathcal{S}\cdot\mathcal{C} \vdash es : (ts \text{ -> } ts')$
trust-t $\mathcal{C}$ = *Untrusted*
shows $\exists\, a'\ es'\text{-}a.\ (|es'|)\ a' \rightsquigarrow (|es'\text{-}a|) \wedge$ *exprs-public-agree es-a es'-a* $\wedge (a \sim\text{-}a\ a')$
⟨*proof*⟩

lemma *exprs-public-agree-imp-reduce*:
assumes $(|s;vs;es|)\ a \rightsquigarrow\text{-}\ i\ (|s\text{-}a;vs\text{-}a;es\text{-}a|)$
exprs-public-agree es es'
publics-agree vs vs'
store-public-agree s s'
store-typing s $\mathcal{S}$
tvs = *map typeof vs*
i < *length* (*inst s*)
$\mathcal{C} = ((s\text{-}inst\ \mathcal{S})!i)(|trust\text{-}t := Untrusted,\ local := (local\ ((s\text{-}inst\ \mathcal{S})!i)\ @\ tvs),\ label := arb\text{-}labs,\ return := arb\text{-}return|)$
$\mathcal{S}\cdot\mathcal{C} \vdash es : (ts \text{ -> } ts')$
shows $\exists\, a'\ s'\text{-}a\ vs'\text{-}a\ es'\text{-}a.\ (|s';vs';es'|)\ a' \rightsquigarrow\text{-}\ i\ (|s'\text{-}a;vs'\text{-}a;es'\text{-}a|) \wedge$
exprs-public-agree es-a es'-a $\wedge$
publics-agree vs-a vs'-a $\wedge$
store-public-agree s-a s'-a $\wedge$
$(a \sim\text{-}a\ a')$
⟨*proof*⟩

lemma *actions-indistinguishable-secrets*:
assumes *r-actions* $(|s;vs;es|)$ *i as*
exprs-public-agree es es'
publics-agree vs vs'
store-public-agree s s'
store-typing s $\mathcal{S}$
tvs = *map typeof vs*
i < *length* (*inst s*)
$\mathcal{C} = ((s\text{-}inst\ \mathcal{S})!i)(|trust\text{-}t := Untrusted,\ local := (local\ ((s\text{-}inst\ \mathcal{S})!i)\ @\ tvs),\ label := arb\text{-}labs,\ return := arb\text{-}return|)$
$\mathcal{S}\cdot\mathcal{C} \vdash es : (ts \text{ -> } ts')$
shows $\exists\, as'.$ (*r-actions* $(|s';vs';es'|)$ *i as'*) $\wedge$ *list-all2 action-indistinguishable as as'*
⟨*proof*⟩

lemma *function-actions-indistinguishable-secrets*:
assumes *r-actions* $(|s;vs;es|)$ *i as*
exprs-public-agree es es'
publics-agree vs vs'
store-public-agree s s'
store-typing s $\mathcal{S}$

$\mathcal{S}\cdot$ *Untrusted*$\cdot$*rs* $\Vdash$*-i vs*;*es* : *ts'*
shows $\exists$ *as'*. (*r-actions* (|*s'*;*vs'*;*es'*|) *i as'*) $\wedge$ *list-all2 action-indistinguishable as as'*
⟨*proof*⟩

theorem *config-actions-indistinguishable-secrets*:
assumes $\vdash$*-i s*;*vs*;*es* : (*Untrusted*,*ts*)
r-actions (|*s*;*vs*;*es*|) *i as*
exprs-public-agree es es'
publics-agree vs vs'
store-public-agree s s'
shows $\exists$ *as'*. (*r-actions* (|*s'*;*vs'*;*es'*|) *i as'*) $\wedge$ *list-all2 action-indistinguishable as as'*
⟨*proof*⟩

lemma *config-indistinguishable-imp-reduce*:
assumes (|*s*;*vs*;*es*|) *a*$\rightsquigarrow$*- i* (|*s-a*;*vs-a*;*es-a*|)
(*s*,*vs*,*es*) $\sim$*-c* (*s'*,*vs'*,*es'*)
$\vdash$*-i s*;*vs*;*es* : (*Untrusted*,*ts*)
shows $\exists$ *a' s'-a vs'-a es'-a*. (|*s'*;*vs'*;*es'*|) *a'*$\rightsquigarrow$*- i* (|*s'-a*;*vs'-a*;*es'-a*|) $\wedge$
((*s-a*,*vs-a*,*es-a*) $\sim$*-c* (*s'-a*,*vs'-a*,*es'-a*)) $\wedge$
(*a* $\sim$*-a a'*)
⟨*proof*⟩

definition *config-bisimulation* :: ((*s* $\times$ *v list* $\times$ *e list*) $\times$ *nat*) *rel* $\Rightarrow$ *bool* **where**
config-bisimulation R $\equiv$
$\forall$ (((*s1*,*vs1*,*es1*),*i1*),((*s2*,*vs2*,*es2*),*i2*)) $\in$ *R*.
($\forall$ *s1' vs1' es1' a*. (|*s1*;*vs1*;*es1*|) *a*$\rightsquigarrow$*-i1* (|*s1'*;*vs1'*;*es1'*|) $\longrightarrow$ ($\exists$ *s2' vs2' es2' a'*. (|*s2*;*vs2*;*es2*|) *a'*$\rightsquigarrow$*-i2* (|*s2'*;*vs2'*;*es2'*|) $\wedge$ (((*s1'*,*vs1'*,*es1'*),*i1*),((*s2'*,*vs2'*,*es2'*),*i2*)) $\in$ *R* $\wedge$ (*a* $\sim$*-a a'*)))
$\wedge$ ($\forall$ *s2' vs2' es2' a*. (|*s2*;*vs2*;*es2*|) *a*$\rightsquigarrow$*-i2* (|*s2'*;*vs2'*;*es2'*|) $\longrightarrow$ ($\exists$ *s1' vs1' es1' a'*. (|*s1*;*vs1*;*es1*|) *a'*$\rightsquigarrow$*-i1* (|*s1'*;*vs1'*;*es1'*|) $\wedge$ (((*s1'*,*vs1'*,*es1'*),*i1*),((*s2'*,*vs2'*,*es2'*),*i2*)) $\in$ *R* $\wedge$ (*a* $\sim$*-a a'*)))

definition *config-bisimilar* :: ((*s* $\times$ *v list* $\times$ *e list*) $\times$ *nat*) *rel* **where**
config-bisimilar $\equiv$ $\bigcup$ { *R*. *config-bisimulation R* }

lemma *config-bisimilar-ex-config-bisimulation*:
assumes (((*s*,*vs*,*es*),*i*), ((*s'*,*vs'*,*es'*),*i'*)) $\in$ *config-bisimilar*
shows $\exists$ *R*. *config-bisimulation R* $\wedge$ (((*s*,*vs*,*es*),*i*), ((*s'*,*vs'*,*es'*),*i'*)) $\in$ *R*
⟨*proof*⟩

definition *typed-indistinguishable-pairs* :: ((*s* $\times$ *v list* $\times$ *e list*) $\times$ *nat*) *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 s*;*vs*;*es* : (*Untrusted*,*ts*))}

lemma *config-bisimulation-typed-indistinguishable-pairs1*:
assumes (((*s1*,*vs1*,*es1*),*i1*), ((*s2*,*vs2*,*es2*),*i2*)) $\in$ *typed-indistinguishable-pairs*

$(\!|s1;vs1;es1|\!)$ $a1 \leadsto$-$i1$ $(\!|s1';vs1';es1'|\!)$
shows $(\exists\, s2'\ vs2'\ es2'\ a2\ ts.\ (\!|s2;vs2;es2|\!)\ a2 \leadsto\text{-}i2\ (\!|s2';vs2';es2'|\!) \wedge$
$(((s1',vs1',es1'),i1),((s2',vs2',es2'),i2)) \in$ *typed-indistinguishable-pairs* $\wedge$
$(a1 \sim\text{-}a\ a2))$
$\langle proof \rangle$

lemma *config-bisimulation-typed-indistinguishable-pairs2*:
assumes $(((s1,vs1,es1),i1),\ ((s2,vs2,es2),i2)) \in$ *typed-indistinguishable-pairs*
$(\!|s2;vs2;es2|\!)\ a2 \leadsto\text{-}i2\ (\!|s2';vs2';es2'|\!)$
shows $(\exists\, s1'\ vs1'\ es1'\ a1\ ts.\ (\!|s1;vs1;es1|\!)\ a1 \leadsto\text{-}i1\ (\!|s1';vs1';es1'|\!) \wedge$
$(((s1',vs1',es1'),i1),((s2',vs2',es2'),i2)) \in$ *typed-indistinguishable-pairs* $\wedge$
$(a2 \sim\text{-}a\ a1))$
$\langle proof \rangle$

lemma *config-bisimulation-typed-indistinguishable-pairs*:
config-bisimulation typed-indistinguishable-pairs
$\langle proof \rangle$

theorem *config-indistinguishable-imp-config-bisimilar*:
assumes $(s,vs,es) \sim\text{-}c\ (s',vs',es')$
$\vdash\text{-}i\ s;vs;es : (Untrusted,ts)$
shows $(((s,vs,es),i),\ ((s',vs',es'),i)) \in$ *config-bisimilar*
$\langle proof \rangle$

inductive *reduce-relpowp* :: $s \Rightarrow v\ list \Rightarrow e\ list \Rightarrow action\ list \Rightarrow nat \Rightarrow s \Rightarrow v\ list \Rightarrow e\ list \Rightarrow bool$ $((\!|\text{-};\text{-};\text{-}|\!)\ \text{-}\,\hat{}\,\hat{}\!\leadsto'\text{-}\ \text{-}\ (\!|\text{-};\text{-};\text{-}|\!)\ 60)$ **where**
$(\!|s;vs;es|\!)\ []\ \hat{}\,\hat{}\!\leadsto\text{-}i\ (\!|s;vs;es|\!)$
$|\ [\![(\!|s;vs;es|\!)\ a\leadsto\text{-}i\ (\!|s'';vs'';es''|\!);\ (\!|s'';vs'';es''|\!)\ as\,\hat{}\,\hat{}\!\leadsto\text{-}i\ (\!|s';vs';es'|\!)]\!] \Longrightarrow (\!|s;vs;es|\!)\ (a\#as)\,\hat{}\,\hat{}\!\leadsto\text{-}i\ (\!|s';vs';es'|\!)$

theorem *config-indistinguishable-trace-noninterference*:
assumes $(\!|s;vs;es|\!)\ as\,\hat{}\,\hat{}\!\leadsto\text{-}i\ (\!|s\text{-}as;vs\text{-}as;es\text{-}as|\!)$
$(s,vs,es) \sim\text{-}c\ (s',vs',es')$
$\vdash\text{-}i\ s;vs;es : (Untrusted,ts)$
shows $\exists\, s'\text{-}as\ vs'\text{-}as\ es'\text{-}as\ as'.\ (\!|s';vs';es'|\!)\ as'\,\hat{}\,\hat{}\!\leadsto\text{-}i\ (\!|s'\text{-}as;vs'\text{-}as;es'\text{-}as|\!) \wedge$
list-all2 action-indistinguishable as as' $\wedge$
config-indistinguishable $(s\text{-}as,vs\text{-}as,es\text{-}as)\ (s'\text{-}as,vs'\text{-}as,es'\text{-}as)$
$\langle proof \rangle$

lemma *rep-config-untrusted-quot-typing*:
assumes $((s,vs,es),i) = (rep\text{-}config\text{-}untrusted\text{-}quot\ x)$
shows $\exists\, ts.\ \vdash\text{-}i\ s;vs;es : (Untrusted,ts)$
$\langle proof \rangle$

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 × v *list* × e *list*), *nat*, *action llist*] ⇒ *bool* **where**
base:⟦ ∀ s' vs' es' a'. ¬⦇s;vs;es⦈ a'⇝-i ⦇s';vs';es'⦈ ⟧ ⟹ *config-is-trace* (s,vs,es) i *LNil*
| *step*:⟦ ⦇s;vs;es⦈ a⇝-i ⦇s';vs';es'⦈; *config-is-trace* (s',vs',es') i tr ⟧ ⟹ *config-is-trace* (s,vs,es) i (*LCons* a tr)

definition *config-trace-set* :: [(s × v *list* × e *list*), *nat*] ⇒ (*action llist*) *set* **where**
config-trace-set ≡ λc i. *Collect* (*config-is-trace* c i)

definition *ct-prop* :: [(s × v *list* × e *list*), *nat*, *action llist*] ⇒ *bool* **where**
ct-prop c i tr ≡ ∃ tr'. *llist-all2 action-indistinguishable* tr tr' ∧ *config-is-trace* c i tr'

coinductive *P-co* :: [(s × v *list* × e *list*), *nat*, *action llist*] ⇒ *bool* **where**
base:⟦ ∀ s' vs' es' a'. ¬⦇s;vs;es⦈ a'⇝-i ⦇s';vs';es'⦈ ⟧ ⟹ *P-co* (s,vs,es) i *LNil*
| *step*:⟦ ⦇s;vs;es⦈ a'⇝-i ⦇s';vs';es'⦈; *action-indistinguishable* a a'; *P-co* (s',vs',es') i tr ⟧ ⟹ *P-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 xa i xb
and *step*:
(⋀$x1$ $x2$ $x3$.
  X $x1$ $x2$ $x3$ ⟹
  (∃ s vs es i.
    $x1$ = (s, vs, es) ∧
    $x2$ = i ∧
    $x3$ = *LNil* ∧
    (∀ s' vs' es' a'. ¬ ⦇s;vs;es⦈ a'⇝- i ⦇s';vs';es'⦈)) ∨
  (∃ s vs es a' i s' vs' es' a tr.
    $x1$ = (s, vs, es) ∧
    $x2$ = i ∧
    $x3$ = *LCons* a tr ∧
    ⦇s;vs;es⦈ a'⇝- i ⦇s';vs';es'⦈ ∧
    (a ∼-a a') ∧
    X (s', vs', es') i tr))
shows *ct-prop* xa i xb
⟨*proof*⟩

lemma *config-indistinguishable-imp-reduce2*:
assumes $(s,vs,es) \sim\text{-}c\ (s',vs',es')$
$\vdash\text{-}i\ s;vs;es : (Untrusted,ts)$
config-is-trace $(s,vs,es)\ i\ tr$
shows *ct-prop* $(s',vs',es')\ i\ tr$
⟨*proof*⟩

lemma *config-indistinguishable-imp-reduce3*:
assumes $(s,vs,es) \sim\text{-}c\ (s',vs',es')$
$\vdash\text{-}i\ s;vs;es : (Untrusted,ts)$
config-is-trace $(s,vs,es)\ i\ tr$
shows $\exists\, tr'$. *llist-all2 action-indistinguishable* $tr\ tr' \wedge$ *config-is-trace* (s',vs',es') $i\ tr'$
⟨*proof*⟩

lemma *program-actions-set2-indistinguishable-secrets-co*:
assumes $tr \in$ (*config-trace-set* $(s, vs, es)\ i)$
$(s,vs,es) \sim\text{-}c\ (s',vs',es')$
$\vdash\text{-}i\ s;vs;es : (Untrusted,ts)$
shows $\exists\, tr' \in$ (*config-trace-set* $(s', vs', es')\ i)$. *llist-all2 action-indistinguishable* $tr\ tr'$
⟨*proof*⟩

lemma *program-actions2-indistinguishable-secrets-abs-set-co*:
assumes $t \in$ (*image abs-observation* (*config-trace-set* $(s, vs, es)\ i))$
$(s,vs,es) \sim\text{-}c\ (s',vs',es')$
$\vdash\text{-}i\ s;vs;es : (Untrusted,ts)$
shows $t \in$ (*image abs-observation* (*config-trace-set* $(s', vs', es')\ i))$
⟨*proof*⟩

lemma *program-actions2-indistinguishable-secrets-abs-set-equiv-co*:
assumes $(s,vs,es) \sim\text{-}c\ (s',vs',es')$
$\vdash\text{-}i\ s;vs;es : (Untrusted,ts)$
shows (*image abs-observation* (*config-trace-set* $(s, vs, es)\ i)) =$ (*image abs-observation* (*config-trace-set* $(s', vs', es')\ i))$
⟨*proof*⟩

lift-definition *config-obs-set* :: $((s \times v\ list \times e\ list) \times nat) \Rightarrow observation\ set$ **is** $(\lambda(c,i).$ (*config-trace-set* $c\ i))$ ⟨*proof*⟩

lift-definition *config-untrusted-quot-obs-set* :: *config-untrusted-quot* $\Rightarrow$ *observation set* **is** $(\lambda c.$ *config-obs-set* $c)$
⟨*proof*⟩

definition *constant-time* :: $((s \times v\ list \times e\ list) \times nat) \Rightarrow bool$ **where**
constant-time $= (\lambda(c, i). \forall\, c'. (c \sim\text{-}c\ c') \longrightarrow (($*config-obs-set* $(c,i)) = ($*config-obs-set* $(c',i))))$

theorem *config-untrusted-constant-time*:

assumes $\vdash$-*i* *s*;*vs*;*es* : (*Untrusted*,*ts*)
shows *constant-time* ((*s*,*vs*,*es*),*i*)
⟨*proof*⟩

lift-definition *config-untrusted-quot-constant-time* :: *config-untrusted-quot* ⇒ *bool*
is *constant-time*
⟨*proof*⟩

lemma *config-untrusted-quot-constant-time-trivial*:
config-untrusted-quot-constant-time = (λ*x*. *True*)
⟨*proof*⟩

definition *trace-set-equiv* = *rel-set* (*llist-all2 action-indistinguishable*)

definition *constant-time-traces* :: ((*s* × *v list* × *e list*) × *nat*) ⇒ *bool* **where**
constant-time-traces = (λ(*c*, *i*). ∀ *c'*. (*c* ∼-*c* *c'*) ⟶ *trace-set-equiv* (*config-trace-set* *c* *i*) (*config-trace-set* *c'* *i*))

lemma *config-untrusted-constant-time-traces*:
assumes $\vdash$-*i* *s*;*vs*;*es* : (*Untrusted*,*ts*)
shows *constant-time-traces* ((*s*,*vs*,*es*),*i*)
⟨*proof*⟩
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*] ⇒ *bool* (*p'-actions* (|-;-;-|) - - *60*) **where**
config-actions s vs es i as ≡ (∃ *s'* *vs'* *es'*. (|*s*;*vs*;*es*|) *as*^^⇝-*i* (|*s'*;*vs'*;*es'*|))

definition *config-trace-set-ind* :: [(*s* × *v list* × *e list*), *nat*] ⇒ (*action list*) *set*
where
config-trace-set-ind ≡ λ(*s*,*vs*,*es*) *i*. *Collect* (*config-actions s vs es i*)

lemma *config-actions-indistinguishable-secrets-ind*:
assumes (*p-actions* (|*s*;*vs*;*es*|) *i as*)
(*s*,*vs*,*es*) ∼-*c* (*s'*,*vs'*,*es'*)
$\vdash$-*i* *s*;*vs*;*es* : (*Untrusted*,*ts*)
shows ∃ *as'*. (*p-actions* (|*s'*;*vs'*;*es'*|) *i as'*) ∧ *list-all2 action-indistinguishable as as'*
⟨*proof*⟩

lemma *config-actions-indistinguishable-secrets-abs-ind*:
assumes (*p-actions* (|*s*;*vs*;*es*|) *i as*)
(*s*,*vs*,*es*) ∼-*c* (*s'*,*vs'*,*es'*)
$\vdash$-*i* *s*;*vs*;*es* : (*Untrusted*,*ts*)
shows ∃ *as'*. (*p-actions* (|*s'*;*vs'*;*es'*|) *i as'*) ∧ ($$\mathcal{A}$ *as*) = ($$\mathcal{A}$ *as'*)

⟨*proof*⟩

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 $\exists\, as' \in (config\text{-}trace\text{-}set\text{-}ind\ (s',vs',es')\ i).\ list\text{-}all2\ action\text{-}indistinguishable\ as\ as'$
⟨*proof*⟩

lemma *config-actions-indistinguishable-secrets-abs-set-ind*:
assumes $t \in (image\ abs\text{-}obs\ (config\text{-}trace\text{-}set\text{-}ind\ (s,vs,es)\ i))$
$(s,vs,es) \sim\text{-}c\ (s',vs',es')$
$\vdash\text{-}i\ s;vs;es : (Untrusted,ts)$
shows $t \in (image\ abs\text{-}obs\ (config\text{-}trace\text{-}set\text{-}ind\ (s',vs',es')\ i))$
⟨*proof*⟩

lemma *config-actions-indistinguishable-secrets-abs-set-equiv-ind*:
assumes $(s,vs,es) \sim\text{-}c\ (s',vs',es')$
$\vdash\text{-}i\ s;vs;es : (Untrusted,ts)$
shows $(image\ abs\text{-}obs\ (config\text{-}trace\text{-}set\text{-}ind\ (s,vs,es)\ i)) = (image\ abs\text{-}obs\ (config\text{-}trace\text{-}set\text{-}ind\ (s',vs',es')\ i))$
⟨*proof*⟩

lift-definition *config-obs-set-ind* :: $((s \times v\ list \times e\ list) \times nat) \Rightarrow observation\ set$
is $(\lambda(c,i).\ (config\text{-}trace\text{-}set\text{-}ind\ c\ i))$ ⟨*proof*⟩

lift-definition *config-untrusted-quot-obs-set-ind* :: $config\text{-}untrusted\text{-}quot \Rightarrow observation\ set$ **is** $(\lambda c.\ config\text{-}obs\text{-}set\text{-}ind\ c)$
⟨*proof*⟩

definition *constant-time-ind* :: $((s \times v\ list \times e\ list) \times nat) \Rightarrow bool$ **where**
$constant\text{-}time\text{-}ind = (\lambda(c,\ i).\ \forall\, c'.\ (c \sim\text{-}c\ c') \longrightarrow ((config\text{-}obs\text{-}set\text{-}ind\ (c,i)) = (config\text{-}obs\text{-}set\text{-}ind\ (c',i))))$

theorem *config-untrusted-constant-time-ind*:
assumes $\vdash\text{-}i\ s;vs;es : (Untrusted,ts)$
shows $constant\text{-}time\text{-}ind\ ((s,vs,es),i)$
⟨*proof*⟩

lift-definition *config-untrusted-quot-constant-time-ind* :: $config\text{-}untrusted\text{-}quot \Rightarrow bool$ **is** *constant-time-ind*
⟨*proof*⟩

lemma *config-untrusted-quot-constant-time-trivial-ind*:
$config\text{-}untrusted\text{-}quot\text{-}constant\text{-}time\text{-}ind = (\lambda x.\ True)$
⟨*proof*⟩

definition $trace\text{-}set\text{-}equiv = rel\text{-}set\ (list\text{-}all2\ action\text{-}indistinguishable)$

definition *constant-time-traces-ind* :: $((s \times v\ list \times e\ list) \times nat) \Rightarrow bool$ **where**
constant-time-traces-ind = $(\lambda(c, i).\ \forall c'.\ (c \sim\text{-}c\ c') \longrightarrow$ *trace-set-equiv* (*config-trace-set-ind* $c\ i$) (*config-trace-set-ind* $c'\ i$))

lemma *config-untrusted-constant-time-traces*:
assumes $\vdash\text{-}i\ s;vs;es : (Untrusted,ts)$
shows *constant-time-traces-ind* $((s,vs,es),i)$
⟨*proof*⟩
end

15 Set Based Leakage Model (sketch)

theory *Wasm-Leakage* **imports** *Wasm-Secret* **begin**

datatype *arith-leakage* =
Unop-i32-leakage unop-i
| *Unop-i64-leakage unop-i*
| *Unop-f32-leakage unop-f f32*
| *Unop-f64-leakage 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* ⇒ *leakage* **where**
action-leakage a =
  (*case a of*
  Unop-i32-action op′ ⇒ *Arith-leakage* (*Unop-i32-leakage op′*)
| *Unop-i64-action op′* ⇒ *Arith-leakage* (*Unop-i64-leakage op′*)
| *Unop-f32-action op′ c* ⇒ *Arith-leakage* (*Unop-f32-leakage op′ c*)
| *Unop-f64-action op′ c* ⇒ *Arith-leakage* (*Unop-f64-leakage op′ c*)
| *Binop-i32-Some-action op′ c1 c2* ⇒ *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* ⇒ *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* ⇒ *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* ⇒*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* ⇒ *Arith-leakage* (*Binop-f32-Some-leakage op′ c1 c2*)
| *Binop-f32-None-action op′ c1 c2* ⇒ *Arith-leakage* (*Binop-f32-None-leakage op′ c1 c2*)
| *Binop-f64-Some-action op′ c1 c2* ⇒ *Arith-leakage* (*Binop-f64-Some-leakage op′ c1 c2*)

| Binop-f64-None-action op′ c1 c2 ⇒ Arith-leakage (Binop-f64-None-leakage op′ c1 c2)
| Testop-i32-action op′ ⇒ Arith-leakage (Testop-i32-leakage op′)
| Testop-i64-action op′ ⇒ Arith-leakage (Testop-i64-leakage op′)
| Relop-i32-action op′ ⇒ Arith-leakage (Relop-i32-leakage op′)
| Relop-i64-action op′ ⇒ Arith-leakage (Relop-i64-leakage op′)
| Relop-f32-action op′ c1 c2 ⇒ Arith-leakage (Relop-f32-leakage op′ c1 c2)
| Relop-f64-action op′ c1 c2 ⇒ Arith-leakage (Relop-f64-leakage op′ c1 c2)
| Convert-Some-action t1 t2 c ⇒ (if is-int-t t1 ∧ is-int-t t2
then Convert-Some-int-leakage t1 t2
else Convert-Some-leakage t1 t2 c)
| Convert-None-action t1 t2 c ⇒ (if is-int-t t1 ∧ is-int-t t2
then Convert-None-int-leakage t1 t2
else Convert-None-leakage t1 t2 c)
| Reinterpret-action ⇒ Empty-leakage
| Classify-action ⇒ Empty-leakage
| Declassify-action ⇒ Empty-leakage
| Unreachable-action ⇒ Empty-leakage
| Nop-action ⇒ Empty-leakage
| Drop-action ⇒ Empty-leakage
| Select-action sec c ⇒ if (sec = Secret) then Select-leakage None else Select-leakage (Some c)
| Block-action ⇒ Empty-leakage
| Loop-action ⇒ Empty-leakage
| If-false-action c ⇒ If-false-leakage c
| If-true-action c ⇒ If-true-leakage c
| Label-const-action ⇒ Empty-leakage
| Label-trap-action ⇒ Empty-leakage
| Br-action ⇒ Empty-leakage
| Br-if-false-action c ⇒ Br-if-false-leakage c
| Br-if-true-action c ⇒ Br-if-true-leakage c
| Br-table-action c ⇒ Br-table-leakage c
| Br-table-length-action c ⇒ Br-table-length-leakage c
| Local-const-action ⇒ Empty-leakage
| Local-trap-action ⇒ Empty-leakage
| Return-action ⇒ Empty-leakage
| Tee-local-action ⇒ Empty-leakage
| Trap-action ⇒ Empty-leakage
| Call-action ⇒ Empty-leakage
| Call-indirect-Some-action c ⇒ Call-indirect-Some-leakage c
| Call-indirect-None-action c ⇒ Call-indirect-None-leakage c
| Callcl-native-action n ⇒ Callcl-native-leakage n
| Callcl-host-Some-action s args s′ out tr tf host hs ⇒ Host-leakage (Callcl-host-Some-leakage (map fst (filter (λ(m,sec). sec = Public) (mem s))))
| Callcl-host-None-action s args tr tf host hs ⇒ Host-leakage (Callcl-host-Some-leakage (map fst (filter (λ(m,sec). sec = Public) (mem s))))
| Get-local-action ⇒ Empty-leakage
| Set-local-action ⇒ Empty-leakage
| Get-global-action ⇒ Empty-leakage

| *Set-global-action* ⇒ *Empty-leakage*
| *Load-Some-action t n a off* ⇒ *Load-Some-leakage t n a off*
| *Load-None-action t n a off* ⇒ *Load-None-leakage t n a off*
| *Load-packed-Some-action tp sx n a off* ⇒ *Load-packed-Some-leakage tp sx n a off*
| *Load-packed-None-action tp sx n a off* ⇒ *Load-packed-None-leakage tp sx n a off*
| *Store-Some-action t n a off* ⇒ *Store-Some-leakage t n a off*
| *Store-None-action t n a off* ⇒ *Store-None-leakage t n a off*
| *Store-packed-Some-action t tp n a off* ⇒ *Store-packed-Some-leakage t tp n a off*
| *Store-packed-None-action t tp n a off* ⇒ *Store-packed-None-leakage t tp n a off*
| *Current-memory-action l* ⇒ *Current-memory-leakage l*
| *Grow-memory-Some-action l c* ⇒ *Grow-memory-Some-leakage l c*
| *Grow-memory-None-action l c* ⇒ *Grow-memory-None-leakage l c*
| *Label-action* ⇒ *Empty-leakage*
| *Local-action* ⇒ *Empty-leakage*)

lemma *memory-agree-filter*:
assumes *memory-public-agree m m′*
shows (λ(*m*,*sec*). *sec* = *Public*) *m* = (λ(*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.