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* **..**
setup-lifting *type-definition-bytes*
declare *Quotient-bytes[transfer-rule]*

lift-definition *bytes-takefill* :: *byte $\Rightarrow$ nat $\Rightarrow$ bytes $\Rightarrow$ bytes* **is** ($\lambda a\ n\ as.\ takefill$ *(Abs-uint8 a) n as)* **.**
lift-definition *bytes-replicate* :: *nat $\Rightarrow$ byte $\Rightarrow$ bytes* **is** ($\lambda n\ b.\ replicate\ n\ (Abs\text{-}uint8$ *b))* **.**
definition *msbyte* :: *bytes $\Rightarrow$ byte* **where**
  msbyte bs = last (Rep-bytes bs)

typedef *mem = UNIV* :: *(byte list) set* **..**
setup-lifting *type-definition-mem*
declare *Quotient-mem[transfer-rule]*

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

— host
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* :: $'a \Rightarrow bool$

fixes *int-eq* :: $'a \Rightarrow 'a \Rightarrow bool$
fixes *int-lt-u* :: $'a \Rightarrow 'a \Rightarrow bool$
fixes *int-lt-s* :: $'a \Rightarrow 'a \Rightarrow bool$
fixes *int-gt-u* :: $'a \Rightarrow 'a \Rightarrow bool$
fixes *int-gt-s* :: $'a \Rightarrow 'a \Rightarrow bool$
fixes *int-le-u* :: $'a \Rightarrow 'a \Rightarrow bool$
fixes *int-le-s* :: $'a \Rightarrow 'a \Rightarrow bool$
fixes *int-ge-u* :: $'a \Rightarrow 'a \Rightarrow bool$
fixes *int-ge-s* :: $'a \Rightarrow 'a \Rightarrow bool$

fixes *int-of-nat* :: $nat \Rightarrow 'a$
fixes *nat-of-int* :: $'a \Rightarrow nat$
begin
  abbreviation (*input*)
  int-ne **where**
    int-ne x y $\equiv \neg$ (*int-eq x y*)
end

class *wasm-float* = *wasm-base* +

  fixes *float-neg*   :: $'a \Rightarrow 'a$
  fixes *float-abs*   :: $'a \Rightarrow 'a$
  fixes *float-ceil*   :: $'a \Rightarrow 'a$
  fixes *float-floor*   :: $'a \Rightarrow 'a$
  fixes *float-trunc*   :: $'a \Rightarrow 'a$
  fixes *float-nearest* :: $'a \Rightarrow 'a$
  fixes *float-sqrt*   :: $'a \Rightarrow 'a$

  fixes *float-add* :: $'a \Rightarrow 'a \Rightarrow 'a$
  fixes *float-sub* :: $'a \Rightarrow 'a \Rightarrow 'a$
  fixes *float-mul* :: $'a \Rightarrow 'a \Rightarrow 'a$
  fixes *float-div* :: $'a \Rightarrow 'a \Rightarrow 'a$
  fixes *float-min* :: $'a \Rightarrow 'a \Rightarrow 'a$
  fixes *float-max* :: $'a \Rightarrow 'a \Rightarrow 'a$
  fixes *float-copysign* :: $'a \Rightarrow 'a \Rightarrow 'a$

  fixes *float-eq* :: $'a \Rightarrow 'a \Rightarrow bool$
  fixes *float-lt* :: $'a \Rightarrow 'a \Rightarrow bool$
  fixes *float-gt* :: $'a \Rightarrow 'a \Rightarrow bool$
  fixes *float-le* :: $'a \Rightarrow 'a \Rightarrow bool$
  fixes *float-ge* :: $'a \Rightarrow 'a \Rightarrow bool$
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 .. end**
instantiation *i64* :: *wasm-int* **begin instance .. end**
instantiation *f32* :: *wasm-float* **begin instance .. end**
instantiation *f64* :: *wasm-float* **begin instance .. 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

$$| \ T\text{-}f32 \ \Rightarrow \ T\text{-}f32$$
$$| \ T\text{-}f64 \ \Rightarrow \ T\text{-}f64 \,)$$

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 $\equiv$ ((*t-sec t*) = *Public*)

abbreviation *is-secret-t* :: $t \Rightarrow bool$ **where**
  is-secret-t t $\equiv$ ((*t-sec t*) = *Secret*)

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

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

definition *is-mut* :: $tg \Rightarrow bool$ **where**
  is-mut tg = (*tg-mut tg* = *T-mut*)

definition *safe-binop-i* :: $binop\text{-}i \Rightarrow bool$ **where**
  safe-binop-i bop =
    (*case bop of*
       Div - $\Rightarrow$ *False*
     | *Rem -* $\Rightarrow$ *False*
     | *-* $\Rightarrow$ *True*)

definition *app-unop-i* :: $unop\text{-}i \Rightarrow \prime i{::}wasm\text{-}int \Rightarrow \prime i{::}wasm\text{-}int$ **where**
  app-unop-i iop c =
    (*case iop of*
     Ctz $\Rightarrow$ *int-ctz c*
   | *Clz* $\Rightarrow$ *int-clz c*
   | *Popcnt* $\Rightarrow$ *int-popcnt c*)

definition *app-unop-f* :: $unop\text{-}f \Rightarrow \prime f{::}wasm\text{-}float \Rightarrow \prime f{::}wasm\text{-}float$ **where**
  app-unop-f fop c =
          (*case fop of*
            Neg $\Rightarrow$ *float-neg c*
          | *Abs* $\Rightarrow$ *float-abs c*
          | *Ceil* $\Rightarrow$ *float-ceil c*
          | *Floor* $\Rightarrow$ *float-floor c*
          | *Trunc* $\Rightarrow$ *float-trunc c*
          | *Nearest* $\Rightarrow$ *float-nearest c*
          | *Sqrt* $\Rightarrow$ *float-sqrt c*)

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

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

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

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

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

$$| \; Nef \Rightarrow \textit{float-ne c1 c2}$$
$$| \; Ltf \Rightarrow \textit{float-lt c1 c2}$$
$$| \; Gtf \Rightarrow \textit{float-gt c1 c2}$$
$$| \; Lef \Rightarrow \textit{float-le c1 c2}$$
$$| \; Gef \Rightarrow \textit{float-ge c1 c2})$$

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* $\wedge$ *t-length v-t* = *t-length t*)

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

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

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

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

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

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

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

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

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

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

definition *stab* :: $s \Rightarrow nat \Rightarrow nat \Rightarrow cl\ option$ **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* :: $s \Rightarrow nat \Rightarrow v \Rightarrow s$ **where**

14

$$supdate\text{-}glob\text{-}s\ s\ k\ v = s(\!|globs := (globs\ s)[k{:=}((globs\ s)!k)(\!|g\text{-}val := v|\!)]|\!)$$

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

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

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

inductive *store-extension* :: $s \Rightarrow s \Rightarrow bool$ **where**
$[\![insts = insts';\ fs = fs';\ tclss = tclss';\ list\text{-}all2\ (\lambda(bs,sec)\ (bs',sec').\ mem\text{-}size\ bs$
$\leq mem\text{-}size\ bs' \wedge sec = sec')\ bss\ bss';\ gs = gs']\!] \Longrightarrow$
  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* :: $b\text{-}e\ list \Rightarrow e\ list$ ($\$* - 60$) **where**
  to-e-list b-es $\equiv$ map Basic b-es

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

inductive *Lfilled* :: $nat \Rightarrow Lholed \Rightarrow e\ list \Rightarrow e\ list \Rightarrow bool$ **where**

  $L0:[\![const\text{-}list\ vs;\ lholed = (LBase\ vs\ es')]\!] \Longrightarrow Lfilled\ 0\ lholed\ es\ (vs\ @\ es\ @\ es')$

| $LN:[\![const\text{-}list\ vs;\ lholed = (LRec\ vs\ n\ es'\ l\ es'');\ Lfilled\ k\ l\ es\ lfilledk]\!] \Longrightarrow Lfilled$
$(k{+}1)\ lholed\ es\ (vs\ @\ [Label\ n\ es'\ lfilledk]\ @\ es'')$

inductive *Lfilled-exact* :: $nat \Rightarrow Lholed \Rightarrow e\ list \Rightarrow e\ list \Rightarrow bool$ **where**

  $L0:[\![lholed = (LBase\ []\ [])]\!] \Longrightarrow Lfilled\text{-}exact\ 0\ lholed\ es\ es$

| $LN:[\![const\text{-}list\ vs;\ lholed = (LRec\ vs\ n\ es'\ l\ es'');\ Lfilled\text{-}exact\ k\ l\ es\ lfilledk]\!] \Longrightarrow$
Lfilled-exact $(k{+}1)\ lholed\ es\ (vs\ @\ [Label\ n\ es'\ lfilledk]\ @\ es'')$

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

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

$(snd\ x\ =\ Secret) \land (snd\ y\ =\ Secret)))$

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 $= (y = x \lor ((typeof\ y) = (typeof\ x) \land is\text{-}secret\text{-}t\ (typeof\ x)))$

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 $= (g\text{-}mut\ x = g\text{-}mut\ y \land public\text{-}agree\ (g\text{-}val\ x)\ (g\text{-}val\ y))$

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′ $= (inst\ s = inst\ s′\ \land$
                   funcs s = funcs s′ $\land$
                   tab s = tab s′ $\land$
                   memories-public-agree (mem s) (mem s′) $\land$
                   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)*
  using *assms*
  by (*cases t*) (*auto simp add*: *is-int-t-def*)

lemma *is-float-t-exists*:
  assumes *is-float-t t*
  shows *∃ sec. t = T-f32 ∨ t = T-f64*
  using *assms*
  by (*cases t*) (*auto simp add*: *is-float-t-def*)

lemma *int-float-disjoint*: *is-int-t t = −(is-float-t t)*
  by *simp* (*metis is-float-t-def is-int-t-def t.exhaust t.simps(15−18)*)

lemma *types-agree-imp-types-agree-insecure*:
  assumes *types-agree t v*
  shows *types-agree-insecure t v*
  using *assms*
  unfolding *types-agree-def types-agree-insecure-def*
  by *simp*

lemma *stab-unfold*:
  assumes *stab s i j = Some cl*
  shows *∃ k. inst.tab ((inst s)!i) = Some k ∧ length ((tab s)!k) > j ∧((tab s)!k)!j*
= Some cl
proof −
  obtain *k* **where** *have-k*:(*inst.tab ((inst s)!i)*) = *Some k*
    using *assms*
    unfolding *stab-def*
    by *fastforce*
  hence *s-o*:*stab s i j = stab-s s k j*
    using *assms*
    unfolding *stab-def*
    by *simp*
  then obtain *stabinst* **where** *stabinst-def*:*stabinst = ((tab s)!k)*
    by *blast*
  hence *stab-s s k j = (stabinst!j) ∧ (length stabinst > j)*
    using *assms s-o*
    unfolding *stab-s-def*
    by (*cases (length stabinst > j), auto*)
  thus *?thesis*
    using *have-k stabinst-def assms s-o*
    by *auto*
qed

lemma *inj-basic*: *inj Basic*
  by (*meson e.inject(1) injI*)

lemma *inj-basic-econst*: *inj (λv. $C v)*
  by (*simp add: inj-def*)

lemma *to-e-list-1*:[$ a] = $∗ [a]
  by *simp*

lemma *to-e-list-2*:[$ a, $ b] = $∗ [a, b]
  by *simp*

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

lemma *v-exists-b-e*:∃ *ves*. ($$∗*vs*) = ($∗*ves*)
proof (*induction vs*)
  case (*Cons a vs*)
  thus *?case*
  by (*metis list.simps*(*9*))
qed *auto*

lemma *Lfilled-exact-imp-Lfilled*:
  assumes *Lfilled-exact n lholed es LI*
  shows *Lfilled n lholed es LI*
  using *assms*
proof (*induction rule*: *Lfilled-exact.induct*)
  case (*L0 lholed es*)
  thus *?case*
    using *const-list-def Lfilled.intros*(*1*)
    by *fastforce*
next
  case (*LN vs lholed n es′ l es″ k es lfilledk*)
  thus *?case*
    using *Lfilled.intros*(*2*)
    by *fastforce*
qed

lemma *Lfilled-exact-app-imp-exists-Lfilled*:
  assumes *const-list ves*
      Lfilled-exact n lholed (*ves@es*) *LI*
  shows ∃ *lholed′*. *Lfilled n lholed′ es LI*
  using *assms*(*2,1*)
proof (*induction* (*ves@es*) *LI rule*: *Lfilled-exact.induct*)
  case (*L0 lholed*)
  show *?case*
    using *Lfilled.intros*(*1*)[*OF L0*(*2*), *of* - []]
    by *fastforce*
next
  case (*LN vs lholed n es′ l es″ k lfilledk*)
  thus *?case*
    using *Lfilled.intros*(*2*)
    by *fastforce*
qed

lemma *Lfilled-imp-exists-Lfilled-exact*:
  assumes *Lfilled n lholed es LI*
  shows ∃ *lholed′ ves es-c*. *const-list ves* ∧ *Lfilled-exact n lholed′* (*ves@es@es-c*) *LI*
  using *assms Lfilled-exact.intros*
  by (*induction rule*: *Lfilled.induct*) *fastforce+*

lemma *n-zeros-typeof*:
  n-zeros ts = vs ⟹ (*ts = map typeof vs*)

proof (*induction ts arbitrary*: *vs*)
  case *Nil*
  thus *?case*
    unfolding *n-zeros-def*
    by *simp*
next
  case (*Cons t ts*)
  obtain *vs'* **where** *n-zeros ts = vs'*
    using *n-zeros-def*
    by *blast*
  moreover
  have *typeof* (*bitzero t*) = *t*
    unfolding *typeof-def bitzero-def*
    by (*cases t*, *simp-all*)
  ultimately
  show *?case*
    using *Cons*
    unfolding *n-zeros-def*
    by *auto*
qed

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

inductive *b-e-typing* :: [*t-context*, *b-e list*, *tf*] $\Rightarrow$ *bool* (- $\vdash$ - : - *60*) **where**
  — *num ops*
  const:$\mathcal{C} \vdash [C\ v]$: ([]   -> [(*typeof v*)])
| *unop-i*:*is-int-t t* $\Longrightarrow \mathcal{C} \vdash [Unop-i\ t\ \text{-}]$: ([*t*]   -> [*t*])
| *unop-f*:*is-float-t t* $\Longrightarrow \mathcal{C} \vdash [Unop-f\ t\ \text{-}]$: ([*t*]   -> [*t*])
| *binop-i*:$[\![$*is-int-t t*; (*is-secret-t t* $\longrightarrow$ *safe-binop-i iop*)$]\!]$ $\Longrightarrow \mathcal{C} \vdash [Binop-i\ t\ iop]$: ([*t,t*] -> [*t*])
| *binop-f*:*is-float-t t* $\Longrightarrow \mathcal{C} \vdash [Binop-f\ t\ \text{-}]$: ([*t,t*] -> [*t*])
| *testop*:*is-int-t t* $\Longrightarrow \mathcal{C} \vdash [Testop\ t\ \text{-}]$: ([*t*]   -> [(*T-i32* (*t-sec t*))])
| *relop-i*:*is-int-t t* $\Longrightarrow \mathcal{C} \vdash [Relop-i\ t\ \text{-}]$: ([*t,t*] -> [(*T-i32* (*t-sec t*))])
| *relop-f*:*is-float-t t* $\Longrightarrow \mathcal{C} \vdash [Relop-f\ t\ \text{-}]$: ([*t,t*] -> [(*T-i32* (*t-sec t*))])
  — *convert*
| *convert*:$[\![$(*t1* $\neq$ *t2*); *t-sec t1* = *t-sec t2*; (*sx* = *None*) = ((*is-float-t t1* $\wedge$ *is-float-t t2*) $\vee$ (*is-int-t t1* $\wedge$ *is-int-t t2* $\wedge$ (*t-length t1* < *t-length t2*)))$]\!] \Longrightarrow \mathcal{C} \vdash [Cvtop\ t1\ Convert\ t2\ sx]$: ([*t2*] -> [*t1*])
  — *reinterpret*
| *reinterpret*:$[\![$(*t1* $\neq$ *t2*); *t-sec t1* = *t-sec t2*; *t-length t1* = *t-length t2*$]\!] \Longrightarrow \mathcal{C} \vdash [Cvtop\ t1\ Reinterpret\ t2\ None]$: ([*t2*] -> [*t1*])
  — *classify*
| *classify*:$[\![$*is-int-t t2*; *is-public-t t2*; *classify-t t2* = *t1*$]\!] \Longrightarrow \mathcal{C} \vdash [Cvtop\ t1\ Classify\ t2\ None]$: ([*t2*] -> [*t1*])
  — *declassify*
| *declassify*:$[\![$(*trust-t* $\mathcal{C}$) = *Trusted*; *is-int-t t2*; *is-secret-t t2*; *declassify-t t2* = *t1*$]\!] \Longrightarrow \mathcal{C} \vdash [Cvtop\ t1\ Declassify\ t2\ None]$: ([*t2*] -> [*t1*])

— unreachable, nop, drop, select
| *unreachable:$\mathcal{C} \vdash [Unreachable]$: (ts -> ts')*
| *nop:$\mathcal{C} \vdash [Nop]$: ([] -> [])*
| *drop:$\mathcal{C} \vdash [Drop]$: ([t] -> [])*
| *select:$\llbracket sec = Secret \longrightarrow is\text{-}secret\text{-}t \; t \rrbracket \implies \mathcal{C} \vdash [Select \; sec]$: ([$t,t,(T\text{-}i32 \; sec)$] ->*
[t])*
— block
| *block:$\llbracket tf = (tn$ -> $tm); \mathcal{C}\langle\!| label := ([tm] @ (label \; \mathcal{C}))|\!\rangle \vdash es : (tn$ -> $tm) \rrbracket \implies \mathcal{C}$*
$\vdash [Block \; tf \; es]$: (tn -> tm)*
— loop
| *loop:$\llbracket tf = (tn$ -> $tm); \mathcal{C}\langle\!| label := ([tn] @ (label \; \mathcal{C}))|\!\rangle \vdash es : (tn$ -> $tm) \rrbracket \implies \mathcal{C} \vdash$*
[$Loop \; tf \; es$] : (tn -> tm)*
— if then else
| *if-wasm:$\llbracket tf = (tn$ -> $tm); \mathcal{C}\langle\!| label := ([tm] @ (label \; \mathcal{C}))|\!\rangle \vdash es1 : (tn$ -> $tm);$*
$\mathcal{C}\langle\!| label := ([tm] @ (label \; \mathcal{C}))|\!\rangle \vdash es2 : (tn$ -> $tm) \rrbracket \implies \mathcal{C} \vdash [If \; tf \; es1 \; es2]$: (tn @
[($T\text{-}i32 \; Public$)] -> tm)*
— br
| *br:$\llbracket i < length(label \; \mathcal{C}); (label \; \mathcal{C})!i = ts \rrbracket \implies \mathcal{C} \vdash [Br \; i]$: ($t1s$ @ ts -> $t2s$)*
— br-if
| *br-if:$\llbracket i < length(label \; \mathcal{C}); (label \; \mathcal{C})!i = ts \rrbracket \implies \mathcal{C} \vdash [Br\text{-}if \; i]$: (ts @ [($T\text{-}i32$*
$Public$)] -> ts)*
— br-table
| *br-table:$\llbracket list\text{-}all \; (\lambda i. \; i < length(label \; \mathcal{C}) \wedge (label \; \mathcal{C})!i = ts) \; (is@[i]) \rrbracket \implies \mathcal{C} \vdash$*
[$Br\text{-}table \; is \; i$] : ($t1s$ @ ts @ [($T\text{-}i32 \; Public$)] -> $t2s$)*
— return
| *return:$\llbracket (return \; \mathcal{C}) = Some \; ts \rrbracket \implies \mathcal{C} \vdash [Return]$: ($t1s$ @ ts -> $t2s$)*
— call
| *call:$\llbracket trust\text{-}compat \; (trust\text{-}t \; \mathcal{C}) \; tr; \; i < length(func\text{-}t \; \mathcal{C}); (func\text{-}t \; \mathcal{C})!i = (tr,tf) \rrbracket \implies$*
$\mathcal{C} \vdash [Call \; i]$: tf*
— call-indirect
| *call-indirect:$\llbracket trust\text{-}compat \; (trust\text{-}t \; \mathcal{C}) \; tr; \; i < length(types\text{-}t \; \mathcal{C}); (types\text{-}t \; \mathcal{C})!i =$*
$(tr,(t1s$ -> $t2s)); (table \; \mathcal{C}) \neq None \rrbracket \implies \mathcal{C} \vdash [Call\text{-}indirect \; i]$: ($t1s$ @ [($T\text{-}i32$*
$Public$)] -> $t2s$)*
— get-local
| *get-local:$\llbracket i < length(local \; \mathcal{C}); (local \; \mathcal{C})!i = t \rrbracket \implies \mathcal{C} \vdash [Get\text{-}local \; i]$: ([] -> [t])*
— set-local
| *set-local:$\llbracket i < length(local \; \mathcal{C}); (local \; \mathcal{C})!i = t \rrbracket \implies \mathcal{C} \vdash [Set\text{-}local \; i]$: ([t] -> [])*
— tee-local
| *tee-local:$\llbracket i < length(local \; \mathcal{C}); (local \; \mathcal{C})!i = t \rrbracket \implies \mathcal{C} \vdash [Tee\text{-}local \; i]$: ([t] -> [t])*
— get-global
| *get-global:$\llbracket i < length(global \; \mathcal{C}); tg\text{-}t \; ((global \; \mathcal{C})!i) = t \rrbracket \implies \mathcal{C} \vdash [Get\text{-}global \; i]$:*
([] -> [t])*
— set-global
| *set-global:$\llbracket i < length(global \; \mathcal{C}); tg\text{-}t \; ((global \; \mathcal{C})!i) = t; \; is\text{-}mut \; ((global \; \mathcal{C})!i) \rrbracket \implies$*
$\mathcal{C} \vdash [Set\text{-}global \; i]$: ([t] -> [])*
— load
| *load:$\llbracket (memory \; \mathcal{C}) = Some \; (n, sec); \; t\text{-}sec \; t = sec; \; load\text{-}store\text{-}t\text{-}bounds \; a \; (option\text{-}projl$*
$tp\text{-}sx) \; t \rrbracket \implies \mathcal{C} \vdash [Load \; t \; tp\text{-}sx \; a \; off]$: ([($T\text{-}i32 \; Public$)] -> [t])*
— store

| *store*:⟦(*memory* $\mathcal{C}$) = *Some* (*n, sec*); *t-sec t = sec*; *load-store-t-bounds a tp t*⟧ ⟹ $\mathcal{C}$ ⊢ [*Store t tp a off*] : ([(*T-i32 Public*),*t*] -> [])
— *current-memory*
| *current-memory*:(*memory* $\mathcal{C}$) = *Some* (*n, sec*) ⟹ $\mathcal{C}$ ⊢ [*Current-memory*] : ([] -> [(*T-i32 Public*)])
— *Grow-memory*
| *grow-memory*:(*memory* $\mathcal{C}$) = *Some* (*n, sec*) ⟹ $\mathcal{C}$ ⊢ [*Grow-memory*] : ([(*T-i32 Public*)] -> [(*T-i32 Public*)])
— *empty program*
| *empty*:$\mathcal{C}$ ⊢ [] : ([] -> [])
— *composition*
| *composition*:⟦$\mathcal{C}$ ⊢ *es* : (*t1s* -> *t2s*); $\mathcal{C}$ ⊢ [*e*] : (*t2s* -> *t3s*)⟧ ⟹ $\mathcal{C}$ ⊢ *es* @ [*e*] : (*t1s* -> *t3s*)
— *weakening*
| *weakening*:$\mathcal{C}$ ⊢ *es* : (*t1s* -> *t2s*) ⟹ $\mathcal{C}$ ⊢ *es* : (*ts* @ *t1s* -> *ts* @ *t2s*)

inductive *cl-typing* :: [*s-context, cl, tf-t*] ⇒ *bool* **where**
⟦*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*⦈ ⊢ *es* : ([] -> *t2s*)⟧ ⟹ *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*] ⇒ *bool* (-·- ⊢ - : - *60*)
and *s-typing* :: [*s-context, trust, (t list) option, nat, v list, e list, t list*] ⇒ *bool* (-·-·- ⊩′- - -;- : - *60*) **where**

$\mathcal{C}$ ⊢ *b-es* : *tf* ⟹ $\mathcal{S}$·$\mathcal{C}$ ⊢ \$*b-es : *tf*

| ⟦$\mathcal{S}$·$\mathcal{C}$ ⊢ *es* : (*t1s* -> *t2s*); $\mathcal{S}$·$\mathcal{C}$ ⊢ [*e*] : (*t2s* -> *t3s*)⟧ ⟹ $\mathcal{S}$·$\mathcal{C}$ ⊢ *es* @ [*e*] : (*t1s* -> *t3s*)

| $\mathcal{S}$·$\mathcal{C}$ ⊢ *es* : (*t1s* -> *t2s*) ⟹$\mathcal{S}$·$\mathcal{C}$ ⊢ *es* : (*ts* @ *t1s* -> *ts* @ *t2s*)

| $\mathcal{S}$·$\mathcal{C}$ ⊢ [*Trap*] : *tf*

| ⟦$\mathcal{S}$·(*trust-t* $\mathcal{C}$)·*Some ts* ⊩-*i vs*;*es* : *ts*; *length ts = n*⟧ ⟹ $\mathcal{S}$·$\mathcal{C}$ ⊢ [*Local n i vs es*] : ([] -> *ts*)

| ⟦*trust-compat* (*trust-t* $\mathcal{C}$) *tr*; *cl-typing* $\mathcal{S}$ *cl* (*tr,tf*)⟧ ⟹ $\mathcal{S}$·$\mathcal{C}$ ⊢ [*Callcl cl*] : *tf*

| ⟦$\mathcal{S}$·$\mathcal{C}$ ⊢ *e0s* : (*ts* -> *t2s*); $\mathcal{S}$·$\mathcal{C}$⦇*label* := ([*ts*] @ (*label* $\mathcal{C}$))⦈ ⊢ *es* : ([] -> *t2s*); *length ts = n*⟧ ⟹ $\mathcal{S}$·$\mathcal{C}$ ⊢ [*Label n e0s es*] : ([] -> *t2s*)

| ⟦*i* < (*length* (*s-inst* $\mathcal{S}$)); *tvs = map typeof vs*; $\mathcal{C}$ =((*s-inst* $\mathcal{S}$)!*i*)⦇*trust-t* := *tr*, *local* := (*local* ((*s-inst* $\mathcal{S}$)!*i*) @ *tvs*), *return* := *rs*⦈; $\mathcal{S}$·$\mathcal{C}$ ⊢ *es* : ([] -> *ts*); (*rs = Some ts*) ∨ *rs = None*⟧ ⟹ $\mathcal{S}$·*tr*·*rs* ⊩-*i vs*;*es* : *ts*

definition *globi-agree gs n g = (n < length gs ∧ gs!n = g)*

definition *memi-agree sm j m = ((∃ j′ m′. j = Some j′ ∧ j′ < length sm ∧ m = Some m′ ∧ sm!j′ = m′) ∨ j = None ∧ m = None)*

definition *funci-agree fs n f = (n < length fs ∧ fs!n = f)*

inductive *inst-typing :: [s-context, inst, t-context] ⇒ bool* **where**
⟦*list-all2 (funci-agree (s-funcs S)) fs tfs; list-all2 (globi-agree (s-globs S)) gs tgs;
(i = Some i′ ∧ i′ < length (s-tab S) ∧ (s-tab S)!i′ = (the n)) ∨ (i = None ∧ n = None); memi-agree (s-mem S) j m*⟧ ⟹ *inst-typing S* (|*types = ts, funcs = fs, tab = i, mem = j, globs = gs*|) (|*trust-t = tr, types-t = ts, func-t = tfs, global = tgs, table = n, memory = m, local = [], label = [], return = None*|)

definition *glob-agree g tg = (tg-mut tg = g-mut g ∧ tg-t tg = typeof (g-val g))*

definition *tab-agree S tcl = (case tcl of None ⇒ True | Some cl ⇒ ∃ tf. cl-typing S cl tf)*

definition *mem-agree bs m = (λ (bs,sec) (m,sec′). m ≤ mem-size bs ∧ sec = sec′) bs m*

inductive *store-typing :: [s, s-context] ⇒ bool* **where**
⟦*S =* (|*s-inst = Cs, s-funcs = tfs, s-tab = ns, s-mem = ms, s-globs = tgs*|); *list-all2 (inst-typing S) insts Cs; 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* $\implies$ (|[\$(C v), \$(*Cvtop t2 Reinterpret t1 None*)]|)
(*Reinterpret-action*)$\rightsquigarrow$ (|[\$(C (*wasm-deserialise* (*bits v*) *t2*))]|)

— classify

| *classify*:*types-agree-insecure t1 v* $\implies$ (|[\$(C v), \$(*Cvtop t2 Classify t1 None*)]|)
(*Classify-action*)$\rightsquigarrow$ (|[\$(C (*classify v*))]|)

— declassify

| *declassify*:*types-agree-insecure t1 v* $\implies$ (|[\$(C v), \$(*Cvtop t2 Declassify t1 None*)]|)
(*Declassify-action*)$\rightsquigarrow$ (|[\$(C (*declassify v*))]|)

— unreachable

| *unreachable*:(|[\$ *Unreachable*]|) (*Unreachable-action*)$\rightsquigarrow$ (|[*Trap*]|)

— nop

| *nop*:(|[\$ *Nop*]|) (*Nop-action*)$\rightsquigarrow$ (|[]|)

— drop

| *drop*:(|[\$(C v), (\$ *Drop*)]|) (*Drop-action*)$\rightsquigarrow$ (|[]|)

— select

| *select-false*:*int-eq n 0* $\implies$ (|[\$(C $v1$), \$(C $v2$), \$$C$ (*ConstInt32 sec n*), (\$ *Select*
sec')]|) (*Select-action sec' n*)$\rightsquigarrow$ (|[\$(C $v2$)]|)

| *select-true*:*int-ne n 0* $\implies$ (|[\$(C $v1$), \$(C $v2$), \$$C$ (*ConstInt32 sec n*), (\$ *Select*
sec')]|) (*Select-action sec' n*)$\rightsquigarrow$ (|[\$(C $v1$)]|)

— block

| *block*:[[*const-list vs*; *length vs = n*; *length t1s = n*; *length t2s = m*]] $\implies$ (|*vs* @
[\$(*Block* (*t1s -> t2s*) *es*)]|) (*Block-action*)$\rightsquigarrow$ (|[*Label m* [] (*vs* @ (\$* *es*))]|)

— loop

| *loop*:[[*const-list vs*; *length vs = n*; *length t1s = n*; *length t2s = m*]] $\implies$ (|*vs* @
[\$(*Loop* (*t1s -> t2s*) *es*)]|) (*Loop-action*)$\rightsquigarrow$ (|[*Label n* [\$(*Loop* (*t1s -> t2s*) *es*)] (*vs*
@ (\$* *es*))]|)

— if

| *if-false*:*int-eq n 0* $\implies$ (|[\$$C$ (*ConstInt32 sec n*), \$(*If tf e1s e2s*)]|) (*If-false-action*
n)$\rightsquigarrow$ (|[\$(*Block tf e2s*)]|)

| *if-true*:*int-ne n 0* $\implies$ (|[\$$C$ (*ConstInt32 sec n*), \$(*If tf e1s e2s*)]|) (*If-true-action*
n)$\rightsquigarrow$ (|[\$(*Block tf e1s*)]|)

— label

| *label-const*:*const-list vs* $\implies$ (|[*Label n es vs*]|) (*Label-const-action*)$\rightsquigarrow$ (|*vs*|)

| *label-trap*:(|[*Label n es* [*Trap*]]|) (*Label-trap-action*)$\rightsquigarrow$ (|[*Trap*]|)

— br

| *br*:[[*const-list vs*; *length vs = n*; *Lfilled i lholed* (*vs* @ [\$(*Br i*)]) *LI*]] $\implies$ (|[*Label*
n es LI]|) (*Br-action*)$\rightsquigarrow$ (|*vs* @ *es*|)

— br-if

| *br-if-false*:*int-eq n 0* $\implies$ (|[\$$C$ (*ConstInt32 sec n*), \$(*Br-if i*)]|) (*Br-if-false-action*
n)$\rightsquigarrow$ (|[]|)

| *br-if-true*:*int-ne n 0* $\implies$ (|[\$$C$ (*ConstInt32 sec n*), \$(*Br-if i*)]|) (*Br-if-true-action*
n)$\rightsquigarrow$ (|[\$(*Br i*)]|)

— br-table

| *br-table*:[[*length is > (nat-of-int c*)]] $\implies$ (|[\$$C$ (*ConstInt32 sec c*), \$(*Br-table is*
i)]|) (*Br-table-action c*)$\rightsquigarrow$ (|[\$(*Br* (*is*!(*nat-of-int c*)))]|)

| *br-table-length*:[[*length is $\leq$ (nat-of-int c*)]] $\implies$ (|[\$$C$ (*ConstInt32 sec c*), \$(*Br-table*
is i)]|) (*Br-table-length-action c*)$\rightsquigarrow$ (|[\$(*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*;[]|)

27

— 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′*
  using *assms Abs-mem-inverse Abs-bytes-inverse*
  unfolding *mem-grow-def mem-size-def mem-append-def bytes-replicate-def*
  by *auto*

lemma *load-size*:
  (load m n off l = None) = (mem-size m < (off + n + l))
  unfolding *load-def*
  by *(cases n + off + l ≤ mem-size m) auto*

lemma *load-packed-size*:
  (load-packed sx m n off lp l = None) = (mem-size m < (off + n + lp))
  using *load-size*
  unfolding *load-packed-def*
  by *(cases n + off + l ≤ mem-size m) auto*

lemma *store-size1*:
  (store m n off v l = None) = (mem-size m < (off + n + l))
  unfolding *store-def*
  by *(cases n + off + l ≤ mem-size m) auto*

lemma *store-size*:
  assumes *(store m n off v l = Some m′)*
  shows *mem-size m = mem-size m′*
  using *assms Abs-mem-inverse Abs-bytes-inverse*
  unfolding *store-def write-bytes-def bytes-takefill-def*
  by *(cases n + off + l ≤ mem-size m) (auto simp add: mem-size-def)*

lemma *store-packed-size1*:
  (store-packed m n off v l = None) = (mem-size m < (off + n + l))
  using *store-size1*
  unfolding *store-packed-def*

by *simp*

lemma *store-packed-size*:
  assumes (*store-packed m n off v l = Some m′*)
  shows *mem-size m = mem-size m′*
  using *assms store-size*
  unfolding *store-packed-def*
  by *simp*

axiomatization where
  wasm-deserialise-type:*typeof (wasm-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*
  using *assms*
  unfolding *is-float-t-def t-sec-def*
  by (*cases t*) *auto*

lemma *is-secret-int-t*:
  assumes *is-secret-t t*
  shows *is-int-t t*
  using *assms*
  unfolding *is-int-t-def t-sec-def*
  by (*cases t*) *auto*

lemma *typeof-i32*:
  assumes *typeof v = (T-i32 sec)*
  shows $\exists$ *c. v = ConstInt32 sec c*

using *assms*
unfolding *typeof-def*
by (*cases v*) *auto*

lemma *typeof-i64*:
  assumes *typeof v* = (*T-i64 sec*)
  shows ∃ *c*. *v* = *ConstInt64 sec c*
  using *assms*
  unfolding *typeof-def*
  by (*cases v*) *auto*

lemma *typeof-f32*:
  assumes *typeof v* = *T-f32*
  shows ∃ *c*. *v* = *ConstFloat32 c*
  using *assms*
  unfolding *typeof-def*
  by (*cases v*) *auto*

lemma *typeof-f64*:
  assumes *typeof v* = *T-f64*
  shows ∃ *c*. *v* = *ConstFloat64 c*
  using *assms*
  unfolding *typeof-def*
  by (*cases v*) *auto*

lemma *is-int-t-classify-t*:
  assumes *is-int-t t*
          is-public-t t
  shows *is-int-t* (*classify-t t*)
  using *assms*
  unfolding *is-int-t-def classify-t-def t-sec-def*
  by (*cases t*) *auto*

lemma *classify-t-classify-typeof*:
  assumes *types-agree-insecure t v*
  shows (*classify-t t*) = *typeof* (*classify v*)
  using *assms*
  by (*cases t*; *cases v*) (*auto simp add*: *types-agree-insecure-def*
                                   classify-def
                                   typeof-def
                                   classify-t-def
                                   t-length-def
                                   is-int-t-def)

lemma *declassify-t-declassify-typeof*:
  assumes *types-agree-insecure t v*
  shows (*declassify-t t*) = *typeof* (*declassify v*)
  using *assms*
  by (*cases t*; *cases v*) (*auto simp add*: *types-agree-insecure-def*

declassify-def
typeof-def
declassify-t-def
t-length-def
is-int-t-def)

lemma *exists-v-typeof*: $\exists\,v\,v.\ typeof\ v = t$
proof (*cases t*)
  case (*T-i32 sec*)
  fix *v*
  have *typeof* (*ConstInt32 sec v*) = *t*
    using *T-i32*
    unfolding *typeof-def*
    by *simp*
  thus *?thesis*
    using *T-i32*
    by *fastforce*
next
  case (*T-i64 sec*)
  fix *v*
  have *typeof* (*ConstInt64 sec v*) = *t*
    using *T-i64*
    unfolding *typeof-def*
    by *simp*
  thus *?thesis*
    using *T-i64*
    by *fastforce*
next
  case *T-f32*
  fix *v*
  have *typeof* (*ConstFloat32 v*) = *t*
    using *T-f32*
    unfolding *typeof-def*
    by *simp*
  thus *?thesis*
    using *T-f32*
    by *fastforce*
next
  case *T-f64*
  fix *v*
  have *typeof* (*ConstFloat64 v*) = *t*
    using *T-f64*
    unfolding *typeof-def*
    by *simp*
  thus *?thesis*
    using *T-f64*
    by *fastforce*
qed

lemma *lfilled-collapse1*:
  assumes *Lfilled n lholed* (*vs@es*) *LI*
        const-list vs
        length vs ≥ l
  shows ∃ *lholed'*. *Lfilled n lholed'* ((*drop* (*length vs − l*) *vs*)*@es*) *LI*
  using *assms*(*1*)
proof (*induction vs@es LI rule*: *Lfilled.induct*)
  case (*L0 vs' lholed es'*)
  obtain *vs1 vs2* **where** *vs = vs1@vs2 length vs2 = l*
    using *assms*(*3*)
    by (*metis append-take-drop-id diff-diff-cancel length-drop*)
  moreover
  hence *const-list* (*vs'@vs1*)
    using *L0*(*1*) *assms*(*2*)
    unfolding *const-list-def*
    by *simp*
  ultimately
  show *?case*
    using *Lfilled.intros*(*1*)[*of vs'@vs1 - es' vs2@es*]
      by *fastforce*
next
  case (*LN vs lholed n es' l es'' k lfilledk*)
  thus *?case*
    using *Lfilled.intros*(*2*)
    by *fastforce*
qed

lemma *lfilled-collapse2*:
  assumes *Lfilled n lholed* (*es@es'*) *LI*
  shows ∃ *lholed' vs'*. *Lfilled n lholed' es LI*
  using *assms*
proof (*induction es@es' LI rule*: *Lfilled.induct*)
  case (*L0 vs lholed es'*)
  thus *?case*
    using *Lfilled.intros*(*1*)
    by *fastforce*
next
  case (*LN vs lholed n es' l es'' k lfilledk*)
  thus *?case*
    using *Lfilled.intros*(*2*)
    by *fastforce*
qed

lemma *lfilled-collapse3*:
  assumes *Lfilled k lholed* [*Label n les es*] *LI*
  shows ∃ *lholed'*. *Lfilled* (*Suc k*) *lholed' es LI*
  using *assms*
proof (*induction* [*Label n les es*] *LI rule*: *Lfilled.induct*)
  case (*L0 vs lholed es'*)

have *Lfilled 0* (*LBase [] []*) *es es*
  using *Lfilled.intros*(*1*)
  unfolding *const-list-def*
  by (*metis append.left-neutral append-Nil2 list-all-simps*(*2*))
thus *?case*
  using *Lfilled.intros*(*2*) *L0*
  by *fastforce*
next
  case (*LN vs lholed n es′ l es″ k lfilledk*)
  thus *?case*
    using *Lfilled.intros*(*2*)
    by *fastforce*
qed

lemma *unlift-b-e*: **assumes** $\mathcal{S}{\cdot}\mathcal{C} \vdash \$*b\text{-}es : tf$ **shows** $\mathcal{C} \vdash b\text{-}es : tf$
using *assms* **proof** (*induction* $\mathcal{S}$ $\mathcal{C}$ (*\$∗b-es*) *tf arbitrary: b-es*)
  case (*1* $\mathcal{C}$ *b-es tf* $\mathcal{S}$)
  then show *?case*
    using *inj-basic map-injective*
    by *auto*
next
  case (*2* $\mathcal{S}$ $\mathcal{C}$ *es t1s t2s e t3s*)
  obtain *es′ e′* **where** *es′ @ [e′] = b-es*
    using *2*(*5*)
    by (*simp add: snoc-eq-iff-butlast*)
  then show *?case* **using** *2*
    using *b-e-typing.composition*
    by *fastforce*
next
  case (*3* $\mathcal{S}$ $\mathcal{C}$ *t1s t2s ts*)
  then show *?case*
    using *b-e-typing.weakening*
    by *blast*
qed *auto*

lemma *store-typing-imp-inst-length-eq*:
  assumes *store-typing s* $\mathcal{S}$
  shows *length* (*inst s*) = *length* (*s-inst* $\mathcal{S}$)
  using *assms list-all2-lengthD*
  unfolding *store-typing.simps*
  by *fastforce*

lemma *store-typing-imp-func-length-eq*:
  assumes *store-typing s* $\mathcal{S}$
  shows *length* (*funcs s*) = *length* (*s-funcs* $\mathcal{S}$)
  using *assms list-all2-lengthD*
  unfolding *store-typing.simps*
  by *fastforce*

lemma *store-typing-imp-mem-length-eq*:
  assumes *store-typing s S*
  shows *length (s.mem s) = length (s-mem S)*
  using *assms list-all2-lengthD*
  unfolding *store-typing.simps*
  by *fastforce*

lemma *store-typing-imp-glob-length-eq*:
  assumes *store-typing s S*
  shows *length (globs s) = length (s-globs S)*
  using *assms list-all2-lengthD*
  unfolding *store-typing.simps*
  by *fastforce*

lemma *store-typing-imp-inst-typing*:
  assumes *store-typing s S*
        i < length (inst s)
  shows *inst-typing S ((inst s)!i) ((s-inst S)!i)*
  using *assms*
  unfolding *list-all2-conv-all-nth store-typing.simps*
  by *fastforce*

lemma *stab-typed-some-imp-member*:
  assumes *stab s i c = Some cl*
        store-typing s S
        i < length (inst s)
  shows *Some cl ∈ set (concat (s.tab s))*
proof −
  obtain *k′* **where** *k-def*:*inst.tab ((inst s)!i) = Some k′*
                    length ((s.tab s)!k′) > c
                    ((s.tab s)!k′)!c = Some cl
    using *stab-unfold assms(1,3)*
    by *fastforce*
  hence *Some cl ∈ set ((s.tab s)!k′)*
    using *nth-mem*
    by *fastforce*
  moreover
  have *inst-typing S ((inst s)!i) ((s-inst S)!i)*
    using *assms(2,3) store-typing-imp-inst-typing*
    by *blast*
  hence *k′ < length (s-tab S)*
    using *k-def(1)*
    unfolding *inst-typing.simps stypes-def*
    by *auto*
  hence *k′ < length (s.tab s)*
    using *assms(2) list-all2-lengthD*
    unfolding *store-typing.simps*
    by *fastforce*

ultimately
show *?thesis*
    using *k-def*
    by *auto*
qed

lemma *stab-typed-some-imp-cl-typed*:
  assumes *stab s i c = Some cl*
        store-typing s $\mathcal{S}$
        i < length (inst s)
  shows $\exists$ *tf. cl-typing $\mathcal{S}$ cl tf*
proof −
  have *Some cl $\in$ set (concat (s.tab s))*
    using *assms stab-typed-some-imp-member*
    by *auto*
  moreover
  have *list-all (tab-agree $\mathcal{S}$) (concat (s.tab s))*
    using *assms(2)*
    unfolding *store-typing.simps*
    by *auto*
  ultimately
  show *?thesis*
    unfolding *in-set-conv-nth list-all-length tab-agree-def*
    by *fastforce*
qed

lemma *b-e-type-empty1* [*dest*]: **assumes** $\mathcal{C} \vdash [] : (ts\ \text{->}\ ts')$ **shows** $ts = ts'$
  using *assms*
  by *(induction* []::(*b-e list*) (*ts -> ts'*) *arbitrary: ts ts' rule: b-e-typing.induct,*
simp-all)

lemma *b-e-type-empty*: $(\mathcal{C} \vdash [] : (ts\ \text{->}\ ts')) = (ts = ts')$
proof (*safe*)
  assume $\mathcal{C} \vdash [] : (ts\ \text{->}\ ts')$
  thus $ts = ts'$
    by *blast*
next
  assume $ts = ts'$
  thus $\mathcal{C} \vdash [] : (ts'\ \text{->}\ ts')$
    using *b-e-typing.empty b-e-typing.weakening*
    by *fastforce*
qed

lemma *b-e-type-value*:
  assumes $\mathcal{C} \vdash [e] : (ts\ \text{->}\ ts')$
        e = C v
  shows $ts' = ts\ @\ [typeof\ v]$
  using *assms*
  by *(induction* [e] (*ts -> ts'*) *arbitrary: ts ts' rule: b-e-typing.induct, auto*)

36

lemma *b-e-type-load*:
  assumes $\mathcal{C} \vdash [e] : (ts \mathrel{-\!>} ts')$
        $e = Load\ t\ tp\text{-}sx\ a\ off$
  shows $\exists\,ts''\ sec\ n.\ ts = ts''@[(T\text{-}i32\ Public)] \wedge ts' = ts''@[t] \wedge (memory\ \mathcal{C}) = Some\ (n,sec) \wedge t\text{-}sec\ t = sec$
        $load\text{-}store\text{-}t\text{-}bounds\ a\ (option\text{-}projl\ tp\text{-}sx)\ t$
  using *assms*
  by ($induction\ [e]\ (ts \mathrel{-\!>} ts')\ arbitrary$: $ts\ ts'\ rule$: $b\text{-}e\text{-}typing.induct$, $auto$)

lemma *b-e-type-store*:
  assumes $\mathcal{C} \vdash [e] : (ts \mathrel{-\!>} ts')$
        $e = Store\ t\ tp\ a\ off$
    shows $ts = ts'@[(T\text{-}i32\ Public),\ t]$
        $\exists\,sec\ n.\ (memory\ \mathcal{C}) = Some\ (n,sec) \wedge t\text{-}sec\ t = sec$
        $load\text{-}store\text{-}t\text{-}bounds\ a\ tp\ t$
  using *assms*
  by ($induction\ [e]\ (ts \mathrel{-\!>} ts')\ arbitrary$: $ts\ ts'\ rule$: $b\text{-}e\text{-}typing.induct$, $auto$)

lemma *b-e-type-current-memory*:
  assumes $\mathcal{C} \vdash [e] : (ts \mathrel{-\!>} ts')$
        $e = Current\text{-}memory$
  shows $\exists\,sec\ n.\ ts' = ts\ @\ [(T\text{-}i32\ Public)] \wedge (memory\ \mathcal{C}) = Some\ (n,sec)$
  using *assms*
  by ($induction\ [e]\ (ts \mathrel{-\!>} ts')\ arbitrary$: $ts\ ts'\ rule$: $b\text{-}e\text{-}typing.induct$, $auto$)

lemma *b-e-type-grow-memory*:
  assumes $\mathcal{C} \vdash [e] : (ts \mathrel{-\!>} ts')$
        $e = Grow\text{-}memory$
  shows $\exists\,ts''.\ ts = ts''@[(T\text{-}i32\ Public)] \wedge ts = ts' \wedge (\exists\,n.\ (memory\ \mathcal{C}) = Some\ n)$
  using *assms*
  by ($induction\ [e]\ (ts \mathrel{-\!>} ts')\ arbitrary$: $ts\ ts'\ rule$: $b\text{-}e\text{-}typing.induct$) $auto$

lemma *b-e-type-nop*:
  assumes $\mathcal{C} \vdash [e] : (ts \mathrel{-\!>} ts')$
        $e = Nop$
  shows $ts = ts'$
  using *assms*
  by ($induction\ [e]\ (ts \mathrel{-\!>} ts')\ arbitrary$: $ts\ ts'\ rule$: $b\text{-}e\text{-}typing.induct$, $auto$)

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

lemma *b-e-type-binop-relop*:

37

assumes $\mathcal{C} \vdash [e] : (ts \rightarrow ts')$
  $e = Binop\text{-}i\ t\ iop \lor e = Binop\text{-}f\ t\ fop \lor e = Relop\text{-}i\ t\ irop \lor e = Relop\text{-}f$
$t\ frop$
 shows $\exists\ ts''.\ ts = ts''@[t,t] \land ts' = ts''@[arity\text{-}2\text{-}result(e)]$
  $e = Binop\text{-}i\ t\ iop \implies is\text{-}secret\text{-}t\ t \implies safe\text{-}binop\text{-}i\ iop$
  $e = Binop\text{-}f\ t\ fop \implies is\text{-}float\text{-}t\ t$
  $e = Relop\text{-}f\ t\ frop \implies is\text{-}float\text{-}t\ t$
 using *assms*
 unfolding *arity-2-result-def*
 by (*induction* $[e]$ ($ts \rightarrow ts'$) *arbitrary*: $ts\ ts'$ *rule*: *b-e-typing.induct*, *auto*)

lemma *b-e-type-testop-drop-cvt0*:
 assumes $\mathcal{C} \vdash [e] : (ts \rightarrow ts')$
  $e = Testop\ t\ testop \lor e = Drop \lor e = Cvtop\ t1\ cvtop\ t2\ sx$
 shows $ts \neq []$
 using *assms*
 by (*induction* $[e]$ $ts \rightarrow ts'$ *arbitrary*: ts' *rule*: *b-e-typing.induct*, *auto*)

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

lemma *b-e-type-unop-testop*:
 assumes $\mathcal{C} \vdash [e] : (ts \rightarrow ts')$
  $e = Unop\text{-}i\ t\ iop \lor e = Unop\text{-}f\ t\ fop \lor e = Testop\ t\ testop$
 shows $\exists\ ts''.\ ts = ts''@[t] \land ts' = ts''@[arity\text{-}1\text{-}result\ e]$
  $e = Unop\text{-}f\ t\ fop \implies is\text{-}float\text{-}t\ t$
 using *assms int-float-disjoint*
 unfolding *arity-1-result-def*
 by (*induction* $[e]$ ($ts \rightarrow ts'$) *arbitrary*: $ts\ ts'$ *rule*: *b-e-typing.induct*) *fastforce+*

lemma *b-e-type-cvtop*:
 assumes $\mathcal{C} \vdash [e] : (ts \rightarrow ts')$
  $e = Cvtop\ t1\ cvtop\ t\ sx$
 shows $\exists\ ts''.\ ts = ts''@[t] \land ts' = ts''@[arity\text{-}1\text{-}result\ e]$
  $cvtop = Convert \implies (t1 \neq t) \land t\text{-}sec\ t1 = t\text{-}sec\ t \land (sx = None) =$
$((is\text{-}float\text{-}t\ t1 \land is\text{-}float\text{-}t\ t) \lor (is\text{-}int\text{-}t\ t1 \land is\text{-}int\text{-}t\ t \land (t\text{-}length\ t1 < t\text{-}length\ t)))$
  $cvtop = Reinterpret \implies (t1 \neq t) \land t\text{-}sec\ t1 = t\text{-}sec\ t \land t\text{-}length\ t1 = t\text{-}length$
t
  $cvtop = Classify \implies is\text{-}int\text{-}t\ t \land is\text{-}public\text{-}t\ t \land classify\text{-}t\ t = t1$
  $cvtop = Declassify \implies (trust\text{-}t\ \mathcal{C}) = Trusted \land is\text{-}int\text{-}t\ t \land is\text{-}secret\text{-}t\ t \land$
$declassify\text{-}t\ t = t1$
 using *assms*

unfolding *arity-1-result-def*
  by (*induction* [*e*] (*ts -> ts′*) *arbitrary*: *ts ts′ rule*: *b-e-typing.induct*, *auto*)

lemma *b-e-type-drop*:
  assumes $\mathcal{C} \vdash$ [*e*] : (*ts -> ts′*)
      e = Drop
  shows $\exists\, t.\ ts = ts′@[t]$
  using *assms b-e-type-testop-drop-cvt0*
by (*induction* [*e*] (*ts -> ts′*) *arbitrary*: *ts ts′ rule*: *b-e-typing.induct*, *auto*)

lemma *b-e-type-select*:
  assumes $\mathcal{C} \vdash$ [*e*] : (*ts -> 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-secret-t t)
  using *assms*
  by (*induction* [*e*] (*ts -> ts′*) *arbitrary*: *ts ts′ rule*: *b-e-typing.induct*, *auto*)

lemma *b-e-type-call*:
  assumes $\mathcal{C} \vdash$ [*e*] : (*ts -> ts′*)
      e = Call i
  shows *i < length (func-t* $\mathcal{C}$*)*
      $\exists\, tr\ ts′′\ tf1\ tf2.\ trust\text{-}compat\ (trust\text{-}t\ \mathcal{C})\ tr \wedge ts = ts′′@tf1 \wedge ts′ = ts′′@tf2$
$\wedge\ (func\text{-}t\ \mathcal{C})!i = (tr,(tf1 \text{ -> } tf2))$
  using *assms*
  by (*induction* [*e*] (*ts -> ts′*) *arbitrary*: *ts ts′ rule*: *b-e-typing.induct*, *auto*)

lemma *b-e-type-call-indirect*:
  assumes $\mathcal{C} \vdash$ [*e*] : (*ts -> ts′*)
      e = Call-indirect i
  shows *i < length (types-t* $\mathcal{C}$*)*
      $\exists\, tr\ ts′′\ tf1\ tf2.\ trust\text{-}compat\ (trust\text{-}t\ \mathcal{C})\ tr \wedge ts = ts′′@tf1@[(T\text{-}i32\ Public)]$
$\wedge\ ts′ = ts′′@tf2 \wedge (types\text{-}t\ \mathcal{C})!i = (tr,(tf1 \text{ -> } tf2))$
  using *assms*
  by (*induction* [*e*] (*ts -> ts′*) *arbitrary*: *ts ts′ rule*: *b-e-typing.induct*, *auto*)

lemma *b-e-type-get-local*:
  assumes $\mathcal{C} \vdash$ [*e*] : (*ts -> ts′*)
      e = Get-local i
  shows $\exists\, t.\ ts′ = ts@[t] \wedge (local\ \mathcal{C})!i = t\ i < length(local\ \mathcal{C})$
  using *assms*
  by (*induction* [*e*] (*ts -> ts′*) *arbitrary*: *ts ts′ rule*: *b-e-typing.induct*, *auto*)

lemma *b-e-type-set-local*:
  assumes $\mathcal{C} \vdash$ [*e*] : (*ts -> ts′*)
      e = Set-local i
  shows $\exists\, t.\ ts = ts′@[t] \wedge (local\ \mathcal{C})!i = t\ i < length(local\ \mathcal{C})$
  using *assms*
  by (*induction* [*e*] (*ts -> ts′*) *arbitrary*: *ts ts′ rule*: *b-e-typing.induct*, *auto*)

lemma *b-e-type-tee-local*:
  assumes $\mathcal{C} \vdash [e] : (ts \text{ -> } ts')$
      $e = \text{Tee-local } i$
  shows $\exists\, ts''\ t.\ ts = ts''@[t] \land ts' = ts''@[t] \land (local\ \mathcal{C})!i = t\ i < length(local\ \mathcal{C})$
  using *assms*
  by $(induction\ [e]\ (ts \text{ -> } ts')\ arbitrary{:}\ ts\ ts'\ rule{:}\ b\text{-}e\text{-}typing.induct,\ auto)$

lemma *b-e-type-get-global*:
  assumes $\mathcal{C} \vdash [e] : (ts \text{ -> } ts')$
      $e = \text{Get-global } i$
  shows $\exists\, t.\ ts' = ts@[t] \land tg\text{-}t((global\ \mathcal{C})!i) = t\ i < length(global\ \mathcal{C})$
  using *assms*
  by $(induction\ [e]\ (ts \text{ -> } ts')\ arbitrary{:}\ ts\ ts'\ rule{:}\ b\text{-}e\text{-}typing.induct,\ auto)$

lemma *b-e-type-set-global*:
  assumes $\mathcal{C} \vdash [e] : (ts \text{ -> } ts')$
      $e = \text{Set-global } i$
  shows $\exists\, t.\ ts = ts'@[t] \land (global\ \mathcal{C})!i = (\!|tg\text{-}mut = T\text{-}mut,\ tg\text{-}t = t|\!) \land i < length(global\ \mathcal{C})$
  using *assms is-mut-def*
  by $(induction\ [e]\ (ts \text{ -> } ts')\ arbitrary{:}\ ts\ ts'\ rule{:}\ b\text{-}e\text{-}typing.induct)\ auto$

lemma *b-e-type-block*:
  assumes $\mathcal{C} \vdash [e] : (ts \text{ -> } ts')$
      $e = \text{Block } tf\ es$
  shows $\exists\, ts''\ tfn\ tfm.\ tf = (tfn \text{ -> } tfm) \land (ts = ts''@tfn) \land (ts' = ts''@tfm) \land$
                $(\mathcal{C}(\!|label := [tfm] @ label\ \mathcal{C}|\!) \vdash es : tf)$
  using *assms*
  by $(induction\ [e]\ (ts \text{ -> } ts')\ arbitrary{:}\ ts\ ts'\ rule{:}\ b\text{-}e\text{-}typing.induct,\ auto)$

lemma *b-e-type-loop*:
  assumes $\mathcal{C} \vdash [e] : (ts \text{ -> } ts')$
      $e = \text{Loop } tf\ es$
  shows $\exists\, ts''\ tfn\ tfm.\ tf = (tfn \text{ -> } tfm) \land (ts = ts''@tfn) \land (ts' = ts''@tfm) \land$
                $(\mathcal{C}(\!|label := [tfn] @ label\ \mathcal{C}|\!) \vdash es : tf)$
  using *assms*
  by $(induction\ [e]\ (ts \text{ -> } ts')\ arbitrary{:}\ ts\ ts'\ rule{:}\ b\text{-}e\text{-}typing.induct,\ auto)$

lemma *b-e-type-if*:
  assumes $\mathcal{C} \vdash [e] : (ts \text{ -> } ts')$
      $e = \text{If } tf\ es1\ es2$
  shows $\exists\, ts''\ tfn\ tfm.\ tf = (tfn \text{ -> } tfm) \land (ts = ts''@tfn\ @\ [(T\text{-}i32\ Public)]) \land$
$(ts' = ts''@tfm) \land$
                $(\mathcal{C}(\!|label := [tfm] @ label\ \mathcal{C}|\!) \vdash es1 : tf) \land$
                $(\mathcal{C}(\!|label := [tfm] @ label\ \mathcal{C}|\!) \vdash es2 : tf)$
  using *assms*
  by $(induction\ [e]\ (ts \text{ -> } ts')\ arbitrary{:}\ ts\ ts'\ rule{:}\ b\text{-}e\text{-}typing.induct,\ auto)$

lemma *b-e-type-br*:
  assumes $\mathcal{C} \vdash [e] : (ts \rightarrow ts')$
        $e = Br\ i$
      shows $i < length(label\ \mathcal{C})$
            $\exists ts\text{-}c\ ts''.\ ts = ts\text{-}c\ @\ ts'' \wedge (label\ \mathcal{C})!i = ts''$
  using *assms*
  by (*induction* $[e]$ $(ts \rightarrow ts')$ *arbitrary*: $ts\ ts'$ *rule*: *b-e-typing.induct*, *auto*)

lemma *b-e-type-br-if*:
  assumes $\mathcal{C} \vdash [e] : (ts \rightarrow ts')$
        $e = Br\text{-}if\ i$
      shows $i < length(label\ \mathcal{C})$
            $\exists ts\text{-}c\ ts''.\ ts = ts\text{-}c\ @\ ts''\ @\ [(T\text{-}i32\ Public)] \wedge ts' = ts\text{-}c\ @\ ts'' \wedge (label$
$\mathcal{C})!i = ts''$
  using *assms*
  by (*induction* $[e]$ $(ts \rightarrow ts')$ *arbitrary*: $ts\ ts'$ *rule*: *b-e-typing.induct*, *auto*)

lemma *b-e-type-br-table*:
  assumes $\mathcal{C} \vdash [e] : (ts \rightarrow ts')$
        $e = Br\text{-}table\ is\ i$
  shows $\exists ts\text{-}c\ ts''.$ *list-all* $(\lambda i.\ i < length(label\ \mathcal{C}) \wedge (label\ \mathcal{C})!i = ts'')\ (is@[i]) \wedge$
$ts = ts\text{-}c\ @\ ts''@[(T\text{-}i32\ Public)]$
  using *assms*
  by (*induction* $[e]$ $(ts \rightarrow ts')$ *arbitrary*: $ts\ ts'$ *rule*: *b-e-typing.induct*, *fastforce+*)

lemma *b-e-type-return*:
  assumes $\mathcal{C} \vdash [e] : (ts \rightarrow ts')$
        $e = Return$
      shows $\exists ts\text{-}c\ ts''.\ ts = ts\text{-}c\ @\ ts'' \wedge (return\ \mathcal{C}) = Some\ ts''$
  using *assms*
  by (*induction* $[e]$ $(ts \rightarrow ts')$ *arbitrary*: $ts\ ts'$ *rule*: *b-e-typing.induct*, *auto*)

lemma *b-e-type-comp*:
  assumes $\mathcal{C} \vdash es@[e] : (t1s \rightarrow t4s)$
  shows $\exists ts'.\ (\mathcal{C} \vdash es : (t1s \rightarrow ts')) \wedge (\mathcal{C} \vdash [e] : (ts' \rightarrow t4s))$
proof (*cases es rule*: *List.rev-cases*)
  case *Nil*
  then show *?thesis*
    using *assms b-e-typing.empty b-e-typing.weakening*
    by *fastforce*
next
  case (*snoc es' e'*)
  show *?thesis* **using** *assms snoc b-e-typing.weakening*
    by (*induction es@[e] (t1s -> t4s) arbitrary*: *t1s t4s*, *fastforce+*)
qed

lemma *b-e-type-comp2-unlift*:
  assumes $\mathcal{S}\cdot\mathcal{C} \vdash [\$e1,\ \$e2] : (t1s \rightarrow t2s)$

41

shows $\exists\, ts'.\ (\mathcal{C} \vdash [e1] : (t1s \rightarrow ts')) \land (\mathcal{C} \vdash [e2] : (ts' \rightarrow t2s))$
   using *assms*
      unlift-b-e[*of* $\mathcal{S}$ $\mathcal{C}$ ([*e1*, *e2*]) (*t1s* -> *t2s*)]
      b-e-type-comp[*of* $\mathcal{C}$ [*e1*] *e2* *t1s* *t2s*]
   by *simp*

lemma *b-e-type-comp2-relift*:
   assumes $\mathcal{C} \vdash [e1] : (t1s \rightarrow ts')$ $\mathcal{C} \vdash [e2] : (ts' \rightarrow t2s)$
   shows $\mathcal{S}{\cdot}\mathcal{C} \vdash [\$e1,\ \$e2] : (ts@t1s \rightarrow ts@t2s)$
   using *assms*
      b-e-typing.composition[*OF assms*]
      e-typing-s-typing.intros(*1*)[*of* $\mathcal{C}$ [*e1*, *e2*] (*t1s* -> *t2s*)]
      e-typing-s-typing.intros(*3*)[*of* $\mathcal{S}$ $\mathcal{C}$ ([$e1,$e2]) *t1s* *t2s* *ts*]
   by *simp*

lemma *b-e-type-value2*:
   assumes $\mathcal{C} \vdash [C\ v1,\ C\ v2] : (t1s \rightarrow t2s)$
   shows $t2s = t1s\ @\ [typeof\ v1,\ typeof\ v2]$
proof −
   obtain ts' **where** $ts'\text{-}def$:$\mathcal{C} \vdash [C\ v1] : (t1s \rightarrow ts')$
                       $\mathcal{C} \vdash [C\ v2] : (ts' \rightarrow t2s)$
    using *b-e-type-comp assms*
   by (*metis append-butlast-last-id butlast.simps*(*2*) *last-ConsL last-ConsR list.distinct*(*1*))
   have $ts' = t1s\ @\ [typeof\ v1]$
    using *b-e-type-value ts'-def*(*1*)
    by *fastforce*
   thus *?thesis*
    using *b-e-type-value ts'-def*(*2*)
    by *fastforce*
qed

lemma *e-type-comp*:
   assumes $\mathcal{S}{\cdot}\mathcal{C} \vdash es@[e] : (t1s \rightarrow t3s)$
   shows $\exists\, ts'.\ (\mathcal{S}{\cdot}\mathcal{C} \vdash es : (t1s \rightarrow ts')) \land (\mathcal{S}{\cdot}\mathcal{C} \vdash [e] : (ts' \rightarrow t3s))$
proof (*cases es rule*: *List.rev-cases*)
   case *Nil*
   thus *?thesis*
    using *assms e-typing-s-typing.intros*(*1*)
    by (*metis append-Nil b-e-type-empty list.simps*(*8*))
next
   case (*snoc es' e'*)
   show *?thesis* **using** *assms snoc*
   proof (*induction es@[e] (t1s -> t3s) arbitrary: t1s t3s*)
    case (*1 $\mathcal{C}$ b-es $\mathcal{S}$*)
    obtain $es''\ e''$ **where** *b-e-defs*:($\$*\ (es''\ @\ [e''])$) = ($\$*\ b\text{-}es$)
     using *1*(*1*,*2*)
     by (*metis Nil-is-map-conv append-is-Nil-conv not-Cons-self2 rev-exhaust*)
    hence ($\$*es''$) = *es* ($\$e''$) = *e*

42

    using *1*(*2*) *inj-basic map-injective*
    by *auto*
  moreover
  have $\mathcal{C} \vdash (es'' \mathbin{@} [e'']) : (t1s \mathrel{->} t3s)$ **using** *1*(*1*)
    using *inj-basic map-injective b-e-defs*
    by *blast*
  then obtain *t2s* **where** $\mathcal{C} \vdash es'' : (t1s \mathrel{->} t2s)\ \mathcal{C} \vdash [e''] : (t2s \mathrel{->} t3s)$
    using *b-e-type-comp*
    by *blast*
  ultimately
  show *?case*
    using *e-typing-s-typing.intros*(*1*)
    by *fastforce*
 next
  case (*3 S C t1s t2s ts*)
  thus *?case*
    using *e-typing-s-typing.intros*(*3*)
    by *fastforce*
 qed *auto*
qed

lemma *e-type-comp-conc*:
  assumes $\mathcal{S}{\cdot}\mathcal{C} \vdash es : (t1s \mathrel{->} t2s)$
       $\mathcal{S}{\cdot}\mathcal{C} \vdash es' : (t2s \mathrel{->} t3s)$
  shows $\mathcal{S}{\cdot}\mathcal{C} \vdash es \mathbin{@} es' : (t1s \mathrel{->} t3s)$
  using *assms*(*2*)
proof (*induction es' arbitrary*: *t3s rule*: *List.rev-induct*)
  case *Nil*
  hence *t2s = t3s*
    using *unlift-b-e*[*of - - []*] *b-e-type-empty*[*of - t2s t3s*]
    by *fastforce*
  then show *?case*
    using *Nil assms*(*1*) *e-typing-s-typing.intros*(*2*)
    by *fastforce*
next
  case (*snoc x xs*)
  then obtain *ts'* **where** $\mathcal{S}{\cdot}\mathcal{C} \vdash xs : (t2s \mathrel{->} ts')\ \mathcal{S}{\cdot}\mathcal{C} \vdash [x] : (ts' \mathrel{->} t3s)$
    using *e-type-comp*[*of - - xs x*]
    by *fastforce*
  then show *?case*
    using *snoc*(*1*)[*of ts'*] *e-typing-s-typing.intros*(*2*)[*of - - es @ xs t1s ts' x t3s*]
    by *simp*
qed

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

proof −
  fix $\mathcal{S}$
  have *1*:$\mathcal{S}\cdot\mathcal{C} \vdash \$*es : (t1s\ \text{->}\ t2s)$
    using *e-typing-s-typing.intros(1)*[*OF assms(1)*]
    by *fastforce*
  have *2*:$\mathcal{S}\cdot\mathcal{C} \vdash \$*es' : (t2s\ \text{->}\ t3s)$
    using *e-typing-s-typing.intros(1)*[*OF assms(2)*]
    by *fastforce*
  show *?thesis*
    using *e-type-comp-conc*[*OF 1 2*]
    by (*simp add*: *unlift-b-e*)
qed

lemma *e-type-comp-conc1*:
  assumes $\mathcal{S}\cdot\mathcal{C} \vdash es@es' : (ts\ \text{->}\ ts')$
  shows $\exists ts''.\ (\mathcal{S}\cdot\mathcal{C} \vdash es : (ts\ \text{->}\ ts'')) \wedge (\mathcal{S}\cdot\mathcal{C} \vdash es' : (ts''\ \text{->}\ ts'))$
  using *assms*
proof (*induction es' arbitrary*: *ts ts' rule*: *List.rev-induct*)
  case *Nil*
  thus *?case*
    using *b-e-type-empty*[*of - ts' ts'*] *e-typing-s-typing.intros(1)*
    by *fastforce*
next
  case (*snoc x xs*)
  then show *?case*
    using *e-type-comp*[*of* $\mathcal{S}$ $\mathcal{C}$ *es* @ *xs x ts ts'*] *e-typing-s-typing.intros(2)*[*of* $\mathcal{S}$ $\mathcal{C}$
xs - - x ts']
    by *fastforce*
qed

lemma *e-type-comp-conc2*:
  assumes $\mathcal{S}\cdot\mathcal{C} \vdash es@es'@es'' : (t1s\ \text{->}\ t2s)$
  shows $\exists ts'\ ts''.\ (\mathcal{S}\cdot\mathcal{C} \vdash es : (t1s\ \text{->}\ ts'))$
                $\wedge\ (\mathcal{S}\cdot\mathcal{C} \vdash es' : (ts'\ \text{->}\ ts''))$
                $\wedge\ (\mathcal{S}\cdot\mathcal{C} \vdash es'' : (ts''\ \text{->}\ t2s))$
proof −
  obtain ts' **where** $\mathcal{S}\cdot\mathcal{C} \vdash es : (t1s\ \text{->}\ ts')$ $\mathcal{S}\cdot\mathcal{C} \vdash es'@es'' : (ts'\ \text{->}\ t2s)$
    using *assms(1)* *e-type-comp-conc1*
    by *fastforce*
  moreover
  then obtain ts'' **where** $\mathcal{S}\cdot\mathcal{C} \vdash es' : (ts'\ \text{->}\ ts'')$ $\mathcal{S}\cdot\mathcal{C} \vdash es'' : (ts''\ \text{->}\ t2s)$
    using *e-type-comp-conc1*
    by *fastforce*
  ultimately
  show *?thesis*
    by *fastforce*
qed

lemma *b-e-type-value-list*:

assumes $(C \vdash es@[C\ v] : (ts\ \text{->}\ ts'@[t]))$
shows $(C \vdash es : (ts\ \text{->}\ ts'))$
     $(typeof\ v = t)$
proof $-$
  obtain ts'' **where** $(C \vdash es : (ts\ \text{->}\ ts''))\ (C \vdash [C\ v] : (ts''\ \text{->}\ ts'\ @\ [t]))$
    using *b-e-type-comp assms*
    by *blast*
  thus $(C \vdash es : (ts\ \text{->}\ ts'))\ (typeof\ v = t)$
    using *b-e-type-value*$[of\ C\ C\ v\ ts''\ ts'\ @\ [t]]$
    by *auto*
qed

lemma *e-type-label*:
  assumes $S{\cdot}C \vdash [Label\ n\ es0\ es] : (ts\ \text{->}\ ts')$
  shows $\exists\ tls\ t2s.\ (ts' = (ts@t2s))$
              $\wedge\ length\ tls = n$
              $\wedge\ (S{\cdot}C \vdash es0 : (tls\ \text{->}\ t2s))$
              $\wedge\ (S{\cdot}C(\!|label := [tls]\ @\ (label\ C)|\!) \vdash es : ([]\ \text{->}\ t2s))$
  using *assms*
proof (*induction* $S\ C\ [Label\ n\ es0\ es]\ (ts\ \text{->}\ ts')$ *arbitrary*: $ts\ ts'$)
  case ($1\ C\ b\text{-}es\ S$)
  then show *?case*
    by (*simp add*: *map-eq-Cons-conv*)
next
  case ($2\ S\ C\ es\ t1s\ t2s\ e\ t3s$)
  then show *?case*
    by (*metis append-self-conv2 b-e-type-empty last-snoc list.simps*(8) *unlift-b-e*)
next
  case ($3\ S\ C\ t1s\ t2s\ ts$)
  then show *?case*
    by *simp*
next
  case ($7\ S\ C\ t2s$)
  then show *?case*
    by *fastforce*
qed

lemma *e-type-callcl-native*:
  assumes $S{\cdot}C \vdash [Callcl\ cl] : (t1s'\ \text{->}\ t2s')$
        $cl = Func\text{-}native\ i\ (tr,tf)\ ts\ es$
  shows $\exists\ t1s\ t2s\ ts\text{-}c.\ trust\text{-}compat\ (trust\text{-}t\ C)\ tr$
                  $\wedge(t1s' = ts\text{-}c\ @\ t1s)$
                  $\wedge\ (t2s' = ts\text{-}c\ @\ t2s)$
                  $\wedge\ tf = (t1s\ \text{->}\ t2s)$
                  $\wedge\ i < length\ (s\text{-}inst\ S)$
                  $\wedge\ (((s\text{-}inst\ S)!i)(\!|trust\text{-}t := tr, local := (local\ ((s\text{-}inst\ S)!i))\ @$
$t1s\ @\ ts, label := ([t2s]\ @\ (label\ ((s\text{-}inst\ S)!i))), return := Some\ t2s|\!)\ \vdash es : ([]$
$\text{->}\ t2s))$
  using *assms*

proof (*induction* S C [*Callcl cl*] (*t1s′ -> t2s′*) *arbitrary*: *t1s′ t2s′*)
  case (*1 C b-es S*)
  thus *?case*
    by *auto*
next
  case (*2 S C es t1s t2s e t3s*)
  have $C \vdash [] : (t1s \text{ -> } t2s)$
    using *2(1,5) unlift-b-e*
    by (*metis Nil-is-map-conv append-Nil butlast-snoc*)
  thus *?case*
    using *2(4,5,6)*
    by *fastforce*
next
  case (*3 S C t1s t2s ts*)
    thus *?case*
    by *fastforce*
next
  case (*6 S C*)
  thus *?case*
    unfolding *cl-typing.simps*
    by *fastforce*
qed

lemma *e-type-callcl-host*:
  assumes $S \cdot C \vdash [Callcl\ cl] : (t1s' \text{ -> } t2s')$
       $cl = Func\text{-}host\ tf\ f$
  shows $\exists\ tr\ t1s\ t2s\ ts\text{-}c.\ trust\text{-}compat\ (trust\text{-}t\ C)\ tr$
                    $\wedge\ (t1s' = ts\text{-}c\ @\ t1s)$
                    $\wedge\ (t2s' = ts\text{-}c\ @\ t2s)$
                    $\wedge\ tf = (tr,(t1s \text{ -> } t2s))$
  using *assms*
proof (*induction S C [Callcl cl] (t1s′ -> t2s′) arbitrary: t1s′ t2s′*)
  case (*1 C b-es S*)
  thus *?case*
    by *auto*
next
  case (*2 S C es t1s t2s e t3s*)
  have $C \vdash [] : (t1s \text{ -> } t2s)$
    using *2(1,5) unlift-b-e*
    by (*metis Nil-is-map-conv append-Nil butlast-snoc*)
  thus *?case*
    using *2(4,5,6)*
    by *fastforce*
next
  case (*3 S C t1s t2s ts*)
    thus *?case*
    by *fastforce*
next
  case (*6 S C*)

thus *?case*
  unfolding *cl-typing.simps*
  by *fastforce*
qed

lemma *e-type-callcl*:
  assumes $\mathcal{S} \cdot \mathcal{C} \vdash [Callcl\ cl] : (t1s' \rightarrow t2s')$
  shows $\exists\ tr\ t1s\ t2s\ ts\text{-}c.\ \ trust\text{-}compat\ (trust\text{-}t\ \mathcal{C})\ tr$
                     $\wedge\ (t1s' = ts\text{-}c\ @\ t1s)$
                     $\wedge\ (t2s' = ts\text{-}c\ @\ t2s)$
                     $\wedge\ cl\text{-}type\ cl = (tr,(t1s \rightarrow t2s))$
proof (*cases cl*)
  case (*Func-native x11 x12 x13 x14*)
  thus *?thesis*
    using *e-type-callcl-native*[*OF assms*]
    unfolding *cl-type-def*
    by (*cases x12*) *fastforce*
next
  case (*Func-host x21 x22*)
  thus *?thesis*
    using *e-type-callcl-host*[*OF assms*]
    unfolding *cl-type-def*
    by *fastforce*
qed

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

lemma *e-type-local*:
  assumes $\mathcal{S} \cdot \mathcal{C} \vdash [Local\ n\ i\ vs\ es] : (ts \rightarrow ts')$
  shows $\exists\ tls.\ i < length\ (s\text{-}inst\ \mathcal{S})$
        $\wedge\ length\ tls = n$
        $\wedge\ (\mathcal{S} \cdot ((s\text{-}inst\ \mathcal{S})!i)(\!|trust\text{-}t := (trust\text{-}t\ \mathcal{C}),\ local := (local\ ((s\text{-}inst\ \mathcal{S})!i))$
$@\ (map\ typeof\ vs),\ return := Some\ tls|\!) \vdash es : ([] \rightarrow tls))$
        $\wedge\ ts' = ts\ @\ tls$
  using *assms*
proof (*induction $\mathcal{S}\ \mathcal{C}$ [Local n i vs es] (ts $\rightarrow$ ts') arbitrary*: *ts ts'*)
  case (*2 $\mathcal{S}\ \mathcal{C}$ es' t1s t2s e t3s*)
  have $t1s = t2s$
    using *2 unlift-b-e*
    by *force*
  thus *?case*
    using *2*

47

by *simp*
qed (*auto simp add*: *unlift-b-e s-typing.simps*)

lemma *e-type-local-shallow*:
  assumes $\mathcal{S} \cdot \mathcal{C} \vdash [Local\ n\ i\ vs\ es] : (ts \text{ -> } ts')$
  shows $\exists\ tls.\ length\ tls = n \wedge ts' = ts@tls \wedge (\mathcal{S} \cdot (trust\text{-}t\ \mathcal{C}) \cdot (Some\ tls) \Vdash\text{-}i\ vs;es$
: *tls*)
  using *assms*
proof (*induction* $\mathcal{S}\ \mathcal{C}\ [Local\ n\ i\ vs\ es]\ (ts \text{ -> } ts')$ *arbitrary*: $ts\ ts'$)
  case (*1* $\mathcal{C}$ *b-es* $\mathcal{S}$)
  thus *?case*
  by (*metis e.distinct*(*7*) *map-eq-Cons-D*)
next
  case (*2* $\mathcal{S}\ \mathcal{C}$ *es t1s t2s e t3s*)
  thus *?case*
  by *simp* (*metis append-Nil append-eq-append-conv e-type-comp-conc e-type-local*)
qed *simp-all*

lemma *e-type-const-unwrap*:
  assumes *is-const e*
  shows $\exists\ v.\ e = \$C\ v$
  using *assms*
proof (*cases e*)
  case (*Basic x1*)
  then show *?thesis*
    using *assms*
  proof (*cases x1*)
    case (*EConst x23*)
      thus *?thesis*
        using *Basic e-typing-s-typing.intros*(*1*,*3*)
        by *fastforce*
  qed (*simp-all add*: *is-const-def*)
qed (*simp-all add*: *is-const-def*)

lemma *is-const-list1*:
  assumes $ves = map\ (Basic \circ EConst)\ vs$
  shows *const-list ves*
  using *assms*
proof (*induction vs arbitrary*: *ves*)
  case *Nil*
  then show *?case*
    unfolding *const-list-def*
    by *simp*
next
  case (*Cons a vs*)
  then obtain ves' **where** $ves' = map\ (Basic \circ EConst)\ vs$
    by *blast*
  moreover

48

have *is-const* ((*Basic* ∘ *EConst*) *a*)
  unfolding *is-const-def*
  by *simp*
ultimately
show *?case*
  using *Cons*
  unfolding *const-list-def*

  by *auto*
qed

lemma *is-const-list*:
  assumes *ves* = $$* *vs*
  shows *const-list ves*
  using *assms is-const-list1*
  unfolding *comp-def*
  by *auto*

lemma *const-list-cons-last*:
  assumes *const-list* (*es*@[*e*])
  shows *const-list es*
     is-const e
  using *assms list-all-append*[*of is-const es* [*e*]]
  unfolding *const-list-def*
  by *auto*

lemma *e-type-const1*:
  assumes *is-const e*
  shows $\exists\, t.\ (\mathcal{S}\cdot\mathcal{C} \vdash [e] : (ts\ \text{->}\ ts@[t]))$
  using *assms*
proof (*cases e*)
  case (*Basic x1*)
  then show *?thesis*
    using *assms*
  proof (*cases x1*)
    case (*EConst x23*)
      hence $\mathcal{C} \vdash [x1] : ([]\ \text{->}\ [typeof\ x23])$
        by (*simp add*: *b-e-typing.intros*(*1*))
      thus *?thesis*
        using *Basic e-typing-s-typing.intros*(*1*,*3*)
        by (*metis append-Nil2 to-e-list-1*)
  qed (*simp-all add*: *is-const-def*)
qed (*simp-all add*: *is-const-def*)

lemma *e-type-const*:
  assumes *is-const e*
      $\mathcal{S}\cdot\mathcal{C} \vdash [e] : (ts\ \text{->}\ ts')$
  shows $\exists\, t.\ (ts' = ts@[t]) \land (\mathcal{S}\cdot\mathcal{C}' \vdash [e] : ([]\ \text{->}\ [t]))$
  using *assms*

proof (*cases e*)
  case (*Basic x1*)
  then show *?thesis*
    using *assms*
  proof (*cases x1*)
    case (*EConst x23*)
      then have *ts′ = ts @ [typeof x23]*
      by (*metis (no-types) Basic assms(2) b-e-type-value list.simps(8,9) unlift-b-e*)
      moreover
      have $\mathcal{S}{\cdot}\mathcal{C}' \vdash [e] : ([] \rightarrow [typeof\ x23])$
        using *Basic EConst b-e-typing.intros(1) e-typing-s-typing.intros(1)*
        by *fastforce*
      ultimately
      show *?thesis*
        by *simp*
  qed (*simp-all add: is-const-def*)
qed (*simp-all add: is-const-def*)

lemma *const-typeof*:
  assumes $\mathcal{S}{\cdot}\mathcal{C} \vdash [\$C\ v] : ([] \rightarrow [t])$
  shows *typeof v = t*
  using *assms*
proof −
  have $\mathcal{C} \vdash [C\ v] : ([] \rightarrow [t])$
    using *unlift-b-e assms*
    by *fastforce*
  thus *?thesis*
    by (*induction [C v] ([] -> [t]) rule: b-e-typing.induct, auto*)
qed

lemma *e-type-const-list*:
  assumes *const-list vs*
          $\mathcal{S}{\cdot}\mathcal{C} \vdash vs : (ts \rightarrow ts')$
  shows $\exists\ tvs.\ ts' = ts\ @\ tvs \land length\ vs = length\ tvs \land (\mathcal{S}{\cdot}\mathcal{C}' \vdash vs : ([] \rightarrow tvs))$
  using *assms*
proof (*induction vs arbitrary: ts ts′ rule: List.rev-induct*)
  case *Nil*
  have $\mathcal{S}{\cdot}\mathcal{C}' \vdash [] : ([] \rightarrow [])$
    using *b-e-type-empty[of $\mathcal{C}'$ [] []] e-typing-s-typing.intros(1)*
    by *fastforce*
  thus *?case*
    using *Nil*
    by (*metis append-Nil2 b-e-type-empty list.map(1) list.size(3) unlift-b-e*)
next
  case (*snoc x xs*)
  hence *v-lists:list-all is-const xs is-const x*
  unfolding *const-list-def*
  by *simp-all*
  obtain *ts″* **where** *ts″-def*:$\mathcal{S}{\cdot}\mathcal{C} \vdash xs : (ts \rightarrow ts'')\ \mathcal{S}{\cdot}\mathcal{C} \vdash [x] : (ts'' \rightarrow ts')$

using *snoc(3) e-type-comp*
by *fastforce*
then obtain *ts-b* **where** *ts-b-def*:*ts″ = ts @ ts-b length xs = length ts-b S·C′ ⊢*
xs : ([] -> *ts-b*)
using *snoc(1) v-lists(1)*
unfolding *const-list-def*
by *fastforce*
then obtain *t* **where** *t-def*:*ts′ = ts @ ts-b @ [t] S·C′ ⊢ [x]* : ([] -> [*t*])
using *e-type-const v-lists(2) ts″-def*
by *fastforce*
moreover
then have *length (ts-b@[t]) = length (xs@[x])*
using *ts-b-def(2)*
by *simp*
moreover
have *S·C′ ⊢ (xs@[x])* : ([] -> *ts-b@[t]*)
using *ts-b-def(3) t-def e-typing-s-typing.intros(2,3)*
by *fastforce*
ultimately
show *?case*
by *simp*
qed

lemma *e-type-const-list-snoc*:
  assumes *const-list vs*
       S·C ⊢ vs : ([] -> *ts@[t]*)
  shows ∃ *vs1 v2*. (*S·C ⊢ vs1* : ([] -> *ts*))
          ∧ (*S·C ⊢ [v2]* : (*ts -> ts@[t]*))
          ∧ (*vs = vs1@[v2]*)
          ∧ *const-list vs1*
          ∧ *is-const v2*
  using *assms*
proof −
  obtain *vs′ v* **where** *vs-def*:*vs = vs′@[v]*
    using *e-type-const-list[OF assms(1,2)]*
    by (*metis append-Nil append-eq-append-conv list.size(3) snoc-eq-iff-butlast*)
  hence *consts-def*:*const-list vs′ is-const v*
    using *assms(1)*
    unfolding *const-list-def*
    by *auto*
  obtain *ts′* **where** *ts′-def*:*S·C ⊢ vs′* : ([] -> *ts′*) *S·C ⊢ [v]* : (*ts′ -> ts@[t]*)
    using *vs-def assms(2) e-type-comp[of S C vs′ v [] ts@[t]]*
    by *fastforce*
  obtain *c* **where** *v = $C c*
    using *e-type-const-unwrap consts-def(2)*
    by *fastforce*
  hence *ts′ = ts*
    using *ts′-def(2) unlift-b-e[of S C [C c]] b-e-type-value*
    by *fastforce*

51

thus *?thesis* **using** *ts′-def vs-def consts-def*
    by *simp*
qed

lemma *e-type-const-list-cons*:
  assumes *const-list vs*
        $\mathcal{S}\cdot\mathcal{C} \vdash vs : ([] \; \text{->} \; (ts1@ts2))$
  shows $\exists\, vs1 \; vs2.\; (\mathcal{S}\cdot\mathcal{C} \vdash vs1 : ([] \; \text{->} \; ts1))$
                $\wedge \; (\mathcal{S}\cdot\mathcal{C} \vdash vs2 : (ts1 \; \text{->} \; (ts1@ts2)))$
                $\wedge \; vs = vs1@vs2$
                $\wedge \; const\text{-}list \; vs1$
                $\wedge \; const\text{-}list \; vs2$
  using *assms*
proof (*induction ts1@ts2 arbitrary*: *vs ts1 ts2 rule*: *List.rev-induct*)
  case *Nil*
  thus *?case*
    using *e-type-const-list*
    by *fastforce*
next
  case (*snoc t ts*)
  note *snoc-outer = snoc*
  show *?case*
  proof (*cases ts2 rule*: *List.rev-cases*)
    case *Nil*
    have $\mathcal{S}\cdot\mathcal{C} \vdash [] : (ts1 \; \text{->} \; ts1 \; @ \; [])$
      using *b-e-typing.empty b-e-typing.weakening e-typing-s-typing.intros(1)*
      by *fastforce*
    then show *?thesis*
      using *snoc(3,4) Nil*
      unfolding *const-list-def*
      by *auto*
  next
    case (*snoc ts2′ a*)
    obtain *vs1 v2* **where** *vs1-def*:$(\mathcal{S}\cdot\mathcal{C} \vdash vs1 : ([] \; \text{->} \; ts1 \; @ \; ts2′))$
                      $(\mathcal{S}\cdot\mathcal{C} \vdash [v2] : (ts1 \; @ \; ts2′ \; \text{->} \; ts1 \; @ \; ts2′@[t]))$
                      $(vs = vs1@[v2])$
                      const-list vs1
                      is-const v2
                      $ts = ts1 \; @ \; ts2′$
      using *e-type-const-list-snoc*[*OF snoc-outer(3), of* $\mathcal{S}$ $\mathcal{C}$ *ts1@ts2′ t*]
          snoc-outer(2,4) snoc
      by *fastforce*
    show *?thesis*
      using *snoc-outer(1)*[*OF vs1-def(6,4,1)*] *snoc-outer(2) vs1-def(3,5)*
          e-typing-s-typing.intros(2)[*OF - vs1-def(2), of - ts1*]
          snoc
      unfolding *const-list-def*
      by *fastforce*
  qed

qed

lemma *e-type-const-conv-vs*:
  assumes *const-list ves*
  shows $\exists vs.\ ves = \$\$\ast\ vs$
  using *assms*
proof (*induction ves*)
  case *Nil*
  thus *?case*
    by *simp*
next
  case (*Cons a ves*)
  thus *?case*
  using *e-type-const-unwrap*
  unfolding *const-list-def*
  by (*metis* (*no-types, lifting*) *list.pred-inject*(*2*) *list.simps*(*9*))
qed

lemma *types-exist-lfilled*:
  assumes *Lfilled k lholed es lfilled*
        $\mathcal{S}\cdot\mathcal{C} \vdash lfilled : (ts \text{ -> } ts')$
  shows $\exists t1s\ t2s\ \mathcal{C}'\ arb\text{-}label.\ (\mathcal{S}\cdot\mathcal{C}(\!|label := arb\text{-}label@(label\ \mathcal{C})|\!) \vdash es : (t1s \text{ -> } t2s))$
  using *assms*
proof (*induction arbitrary*: $\mathcal{C}$ *ts ts' rule*: *Lfilled.induct*)
  case (*L0 vs lholed es' es*)
  hence $\mathcal{S}\cdot(\mathcal{C}(\!|label := label\ \mathcal{C}|\!)) \vdash vs\ @\ es\ @\ es' : (ts \text{ -> } ts')$
    by *simp*
  thus *?case*
    using *e-type-comp-conc2*
    by (*metis append-Nil*)
next
  case (*LN vs lholed n es' l es'' k es lfilledk*)
  obtain $ts''\ ts'''$ **where** $\mathcal{S}\cdot\mathcal{C} \vdash [Label\ n\ es'\ lfilledk] : (ts'' \text{ -> } ts''')$
    using *e-type-comp-conc2*[*OF LN*(*5*)]
    by *fastforce*
  then obtain $t1s\ t2s\ ts$ **where** $test{:}\mathcal{S}\cdot\mathcal{C}(\!|label := [ts]\ @\ (label\ \mathcal{C})|\!) \vdash lfilledk : (t1s \text{ -> } t2s)$
    using *e-type-label*
    by *metis*
  show *?case*
    using *LN*(*4*)[*OF test*(*1*)]
    by *simp* (*metis append.assoc append-Cons append-Nil*)
qed

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

⊢ *es* : (*t1s* -> *t2s*))
proof −
  have ∃ *t1s t2s C′ arb-label*. (S·C⦇*label* := *arb-label*, *return* := (*return C*)⦈) ⊢ *es* : (*t1s* -> *t2s*))
    using *types-exist-lfilled*[*OF assms*]
    by *fastforce*
  thus *?thesis*
    by *fastforce*
qed

lemma *store-typing-imp-func-agree*:
  assumes *store-typing s* S
        i < *length* (*s-inst* S)
        j < *length* (*func-t* ((*s-inst* S)!*i*))
  shows (*sfunc-ind s i j*) < *length* (*s-funcs* S)
      cl-typing S (*sfunc s i j*) ((*s-funcs* S)!(*sfunc-ind s i j*))
      ((*s-funcs* S)!(*sfunc-ind s i j*)) = (*func-t* ((*s-inst* S)!*i*))!*j*
proof −
  have *funcs-agree*:*list-all2* (*cl-typing* S) (*funcs s*) (*s-funcs* S)
    using *assms*(*1*)
    unfolding *store-typing.simps*
    by *auto*
  have *list-all2* (*funci-agree* (*s-funcs* S)) (*inst.funcs* ((*inst s*)!*i*)) (*func-t* ((*s-inst* S)!*i*))
    using *assms*(*1*,*2*) *store-typing-imp-inst-length-eq store-typing-imp-inst-typing*
    by (*fastforce simp add*: *inst-typing.simps*)
  hence *funci-agree* (*s-funcs* S) ((*inst.funcs* ((*inst s*)!*i*))!*j*) ((*func-t* ((*s-inst* S)!*i*))!*j*)
    using *assms*(*3*) *list-all2-nthD2*
    by *blast*
  thus (*sfunc-ind s i j*) < *length* (*s-funcs* S)
      ((*s-funcs* S)!(*sfunc-ind s i j*)) = (*func-t* ((*s-inst* S)!*i*))!*j*
    unfolding *funci-agree-def sfunc-ind-def*
    by *auto*
  thus *cl-typing* S (*sfunc s i j*) ((*s-funcs* S)!(*sfunc-ind s i j*))
    using *funcs-agree list-all2-nthD2*
    unfolding *sfunc-def*
    by *fastforce*
qed

lemma *store-typing-imp-glob-agree*:
  assumes *store-typing s* S
        i < *length* (*s-inst* S)
        j < *length* (*global* ((*s-inst* S)!*i*))
  shows (*sglob-ind s i j*) < *length* (*s-globs* S)
      glob-agree (*sglob s i j*) ((*s-globs* S)!(*sglob-ind s i j*))
      ((*s-globs* S)!(*sglob-ind s i j*)) = (*global* ((*s-inst* S)!*i*))!*j*
proof −
  have *globs-agree*:*list-all2 glob-agree* (*globs s*) (*s-globs* S)
    using *assms*(*1*)

54

unfolding *store-typing.simps*
by *auto*
 have *list-all2 (globi-agree (s-globs $\mathcal{S}$)) (inst.globs ((inst s)!i)) (global ((s-inst $\mathcal{S}$)!i))*
  using *assms(1,2) store-typing-imp-inst-length-eq store-typing-imp-inst-typing*
  by *(fastforce simp add: inst-typing.simps)*
 hence *globi-agree (s-globs $\mathcal{S}$) ((inst.globs ((inst s)!i))!j) ((global ((s-inst $\mathcal{S}$)!i))!j)*
  using *assms(3) list-all2-nthD2*
  by *blast*
 thus *(sglob-ind s i j) < length (s-globs $\mathcal{S}$)*
   ((s-globs $\mathcal{S}$)!(sglob-ind s i j)) = (global ((s-inst $\mathcal{S}$)!i))!j
  unfolding *globi-agree-def sglob-ind-def*
  by *auto*
 thus *glob-agree (sglob s i j) ((s-globs $\mathcal{S}$)!(sglob-ind s i j))*
  using *globs-agree list-all2-nthD2*
  unfolding *sglob-def*
  by *fastforce*
qed

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 −
 have *mems-agree:list-all2 mem-agree (mem s) (s-mem $\mathcal{S}$)*
 using *assms(1)*
 unfolding *store-typing.simps*
 by *auto*
 hence *memi-agree (s-mem $\mathcal{S}$) ((inst.mem ((inst s)!i))) ((memory ((s-inst $\mathcal{S}$)!i)))*
  using *assms(1,2) store-typing-imp-inst-length-eq store-typing-imp-inst-typing*
  by *(fastforce simp add: inst-typing.simps)*
 thus *j < length (s-mem $\mathcal{S}$)*
   $\exists x.$ ((s-mem $\mathcal{S}$)!j) = x $\wedge$ (memory ((s-inst $\mathcal{S}$)!i)) = Some x
  using *assms(3)*
  unfolding *memi-agree-def smem-ind-def*
  by *auto*
 thus *mem-agree ((mem s)!j) ((s-mem $\mathcal{S}$)!j)*
  using *mems-agree list-all2-nthD2*
  unfolding *sglob-def*
  by *fastforce*
qed

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 −
  have *mems-agree*:*list-all2 mem-agree* (*mem s*) (*s-mem* $\mathcal{S}$)
  using *assms*(*1*)
  unfolding *store-typing.simps*
  by *auto*
  hence *memi-agree* (*s-mem* $\mathcal{S}$) ((*inst.mem* ((*inst s*)!*i*))) ((*memory* ((*s-inst* $\mathcal{S}$)!*i*)))
    using *assms*(*1,2*) *store-typing-imp-inst-length-eq store-typing-imp-inst-typing*
    by (*fastforce simp add*: *inst-typing.simps*)
  thus *?thesis*
    using *assms*(*3*)
    unfolding *memi-agree-def smem-ind-def*
    by *auto*
qed

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* (λj. *snd* ((*mem s*)!*j*))
(*smem-ind s i*)
proof (*cases smem-ind s i*)
  case *None*
  thus *?thesis*
    using *assms store-typing-imp-mem-agree-None*
    unfolding *option-projr-def*
    by *fastforce*
next
  case (*Some j*)
  show *?thesis*
    using *store-typing-imp-mem-agree-Some*[*OF assms Some*] *Some*
    unfolding *option-projr-def mem-agree-def*
    by *fastforce*
qed

lemma *store-preserved-mem*:
  assumes *store-typing s* $\mathcal{S}$
      s′ = *s*(|*s.mem* := (*s.mem s*)[*i* := (*mem′*,*sec*)]|)
      mem-size mem′ ≥ *mem-size orig-mem*
      ((*s.mem s*)!*i*) = (*orig-mem, sec*)
  shows *store-typing s′* $\mathcal{S}$
proof −
  obtain *insts fs clss bss gs* **where** *s* = (|*inst* = *insts*, *funcs* = *fs*, *tab* = *clss*, *mem* = *bss*, *globs* = *gs*|)
    using *s.cases*
    by *blast*
  moreover
  obtain *insts′ fs′ clss′ bss′ gs′* **where** *s′* = (|*inst* = *insts′*, *funcs* = *fs′*, *tab* = *clss′*, *mem* = *bss′*, *globs* = *gs′*|)
    using *s.cases*

56

by *blast*
moreover
obtain $\mathcal{C}s$ *tfs ns ms tgs* **where** $\mathcal{S} = ($ *s-inst* $= \mathcal{C}s$, *s-funcs* $=$ *tfs*, *s-tab* $=$ *ns*,
s-mem $=$ *ms*, *s-globs* $=$ *tgs* $)$
  using *s-context.cases*
  by *blast*
moreover
note *s-S-defs* $=$ *calculation*
hence
insts $=$ *insts'*
fs $=$ *fs'*
clss $=$ *clss'*
gs $=$ *gs'*
  using *assms(2)*
  by *simp-all*
hence
list-all2 (*inst-typing* $\mathcal{S}$) *insts'* $\mathcal{C}s$
list-all2 (*cl-typing* $\mathcal{S}$) *fs'* *tfs*
list-all (*tab-agree* $\mathcal{S}$) (*concat clss'*)
list-all2 (λ*cls n*. $n \leq$ *length cls*) *clss'* *ns*
list-all2 glob-agree gs' *tgs*
  using *s-S-defs assms(1)*
  unfolding *store-typing.simps*
  by *auto*
moreover
have *list-all2* (λ (*bs,sec*) (*m,sec'*). $m \leq$ *mem-size bs* $\wedge$ *sec* $=$ *sec'*) *bss'* *ms*
proof $-$
  have *length bss* $=$ *length bss'*
    using *assms(2)* *s-S-defs*
    by (*simp*)
  moreover

  have *initial-mem*:*list-all2* (λ (*bs,sec*) (*m,sec'*). $m \leq$ *mem-size bs* $\wedge$ *sec* $=$ *sec'*)
bss ms
    using *assms(1)* *s-S-defs*
    unfolding *store-typing.simps mem-agree-def*
    by *blast*
  have $\bigwedge n$. $n <$ *length bss* $\Longrightarrow$ (λ (*bs,sec*) (*m,sec'*). $m \leq$ *mem-size bs* $\wedge$ *sec* $=$
sec') (*bss'*!*n*) (*ms*!*n*)
  proof $-$
    fix n
    assume *local-assms*:$n <$ *length bss*
    obtain $\mathcal{C}$-*m* **where** *cmdef*:$\mathcal{C}$-*m* $= \mathcal{C}s$! n
      by *blast*
    hence (λ (*bs,sec*) (*m,sec'*). $m \leq$ *mem-size bs* $\wedge$ *sec* $=$ *sec'*) (*bss*!*n*) (*ms*!*n*)
      using *initial-mem local-assms*
      unfolding *list-all2-conv-all-nth*
      by *simp*
    thus (λ (*bs,sec*) (*m,sec'*). $m \leq$ *mem-size bs* $\wedge$ *sec* $=$ *sec'*) (*bss'*!*n*) (*ms*!*n*)

using *assms(2,3,4) s-S-defs local-assms*
      by (*cases n=i, auto*)
  qed
  ultimately
  show *?thesis*
    by (*metis initial-mem list-all2-all-nthI list-all2-lengthD*)
  qed
  ultimately
  show *?thesis*
    unfolding *store-typing.simps mem-agree-def*
    by *simp*
qed

lemma *types-agree-imp-e-typing*:
  assumes *types-agree t v*
  shows $\mathcal{S} \cdot \mathcal{C} \vdash [\$C\ v] : ([] \rightarrow [t])$
  using *assms e-typing-s-typing.intros(1)[OF b-e-typing.intros(1)]*
  unfolding *types-agree-def*
  by *fastforce*

lemma *list-types-agree-imp-e-typing*:
  assumes *list-all2 types-agree ts vs*
  shows $\mathcal{S} \cdot \mathcal{C} \vdash \$\$* vs : ([] \rightarrow ts)$
  using *assms*
proof (*induction rule: list-all2-induct*)
  case *Nil*
  thus *?case*
    using *b-e-typing.empty e-typing-s-typing.intros(1)*
    by *fastforce*
next
  case (*Cons t ts v vs*)
  hence $\mathcal{S} \cdot \mathcal{C} \vdash [\$C\ v] : ([] \rightarrow [t])$
    using *types-agree-imp-e-typing*
    by *fastforce*
  thus *?case*
    using *e-typing-s-typing.intros(3)[OF Cons(3), of [t]] e-type-comp-conc*
    by *fastforce*
qed

lemma *b-e-typing-imp-list-types-agree*:
  assumes $\mathcal{C} \vdash (map\ (\lambda v.\ C\ v)\ vs) : (ts' \rightarrow ts'@ts)$
  shows *list-all2 types-agree ts vs*
  using *assms*
proof (*induction* (*map* ($\lambda v.\ C\ v$) *vs*) (*ts'* -> *ts'@ts*) *arbitrary*: *ts ts' vs rule*:
b-e-typing.induct)
  case (*composition $\mathcal{C}$ es t1s t2s e*)
  obtain *vs1 vs2* **where** *es-e-def:es = map EConst vs1 [e] = map EConst vs2*
vs1@vs2=vs
    using *composition(5)*

   by (*metis* (*no-types*) *last-map list.simps*(*8*,*9*) *map-butlast snoc-eq-iff-butlast*)
  have *const-list* ($*es)
   using *es-e-def*(*1*) *is-const-list1*
   by *auto*
  then obtain *tvs1* **where** *t2s* = *t1s@tvs1*
   using *e-type-const-list e-typing-s-typing.intros*(*1*)[*OF composition*(*1*)]
   by *fastforce*
  moreover
  have *const-list* ($*[*e*])
   using *es-e-def*(*2*) *is-const-list1*
   by *auto*
  then obtain *tvs2* **where** *t1s* @ *ts* = *t2s* @ *tvs2*
   using *e-type-const-list e-typing-s-typing.intros*(*1*)[*OF composition*(*3*)]
   by *fastforce*
  ultimately
  show *?case*
   using *composition*(*2*,*4*,*5*) *es-e-def*
   by (*auto simp add*: *list-all2-appendI*)
qed (*auto simp add*: *types-agree-def*)

lemma *e-typing-imp-list-types-agree*:
  assumes $\mathcal{S} \cdot \mathcal{C} \vdash$ ($\$\$*$ *vs*) : (*ts′* -> *ts′@ts*)
  shows *list-all2 types-agree ts vs*
proof −
  have ($\$\$*$ *vs*) = $\$*$ (*map* ($\lambda v.$ *C v*) *vs*)
   by *simp*
  thus *?thesis*
   using *assms unlift-b-e b-e-typing-imp-list-types-agree*
   by (*fastforce simp del*: *map-map*)
qed

lemma *store-extension-imp-store-typing*:
  assumes *store-extension s s′*
      store-typing s $\mathcal{S}$
  shows *store-typing s′* $\mathcal{S}$
proof −
  obtain *insts fs clss bss gs* **where** *s* = (|*inst* = *insts, funcs* = *fs, tab* = *clss, mem* = *bss, globs* = *gs*|)
   using *s.cases*
   by *blast*
  moreover
  obtain *insts′ fs′ clss′ bss′ gs′* **where** *s′* = (|*inst* = *insts′, funcs* = *fs′, tab* = *clss′, mem* = *bss′, globs* = *gs′*|)
   using *s.cases*
   by *blast*
  moreover
  obtain $\mathcal{C}s$ *tfs ns ms tgs* **where** $\mathcal{S}$ = (|*s-inst* = $\mathcal{C}s$, *s-funcs* = *tfs, s-tab* = *ns, s-mem* = *ms, s-globs* = *tgs*|)
   using *s-context.cases*

by *blast*
moreover
note *s-S-defs = calculation*
hence
insts = insts′
fs = fs′
clss = clss′
gs = gs′
  using *assms(1)*
  unfolding *store-extension.simps*
  by *simp-all*
hence
list-all2 (inst-typing S) insts′ Cs
list-all2 (cl-typing S) fs′ tfs
list-all (tab-agree S) (concat clss′)
list-all2 (λcls n. n ≤ length cls) clss′ ns
list-all2 glob-agree gs′ tgs
  using *s-S-defs assms(2)*
  unfolding *store-typing.simps*
  by *auto*
moreover
have *list-all2 (λ (bs,sec) (m,sec′). m ≤ mem-size bs ∧ sec = sec′) bss ms*
  using *s-S-defs(1,3) assms(2)*
  unfolding *store-typing.simps mem-agree-def*
  by *simp*
hence *list-all2 mem-agree bss′ ms*
  using *assms(1) s-S-defs(1,2)*
  unfolding *store-extension.simps list-all2-conv-all-nth mem-agree-def*
  by *fastforce*
ultimately
show *?thesis*
  using *store-typing.intros*
  by *fastforce*
qed

lemma *lfilled-deterministic*:
  assumes *Lfilled k lfilled es les*
        Lfilled k lfilled es les′
  shows *les = les′*
  using *assms*
proof (*induction arbitrary*: *les′ rule*: *Lfilled.induct*)
  case (*L0 vs lholed es′ es*)
  thus *?case*
    by (*fastforce simp add*: *Lfilled.simps*[*of 0*])
next
  case (*LN vs lholed n es′ l es″ k es lfilledk*)
  thus *?case*
    unfolding *Lfilled.simps*[*of (k + 1)*]
    by *fastforce*

60

qed

lemma *b-e-typing-trust-compat*:
  assumes $\mathcal{C} \vdash es : tf$
       trust-compat tr (*trust-t* $\mathcal{C}$)
  shows $\mathcal{C}(\!|trust\text{-}t := tr|\!) \vdash es : tf$
  using *assms*
proof (*induction rule: b-e-typing.induct*)
  case (*block tf tn tm $\mathcal{C}$ es*)
  have $\mathcal{C}(\!|label := [tm]\ @\ label\ \mathcal{C},\ trust\text{-}t := tr|\!) \vdash es : (tn \text{ -> } tm)$
    using *block(3,4)*
    by *simp*
  moreover
  have $\mathcal{C}(\!|label := [tm]\ @\ label\ \mathcal{C},\ trust\text{-}t := tr|\!) = \mathcal{C}(\!|trust\text{-}t := tr|\!)(\!|label := [tm]\ @\ label\ \mathcal{C}|\!)$
    by *simp*
  ultimately
  have $\mathcal{C}(\!|trust\text{-}t := tr|\!)(\!|label := [tm]\ @\ label\ \mathcal{C}|\!) \vdash es : (tn \text{ -> } tm)$
    by *metis*
  thus *?case*
    using *b-e-typing.block[OF block(1)]*
    by *simp*
next
  case (*loop tf tn tm $\mathcal{C}$ es*)
  have $\mathcal{C}(\!|label := [tn]\ @\ label\ \mathcal{C},\ trust\text{-}t := tr|\!) \vdash es : (tn \text{ -> } tm)$
    using *loop(3,4)*
    by *simp*
  moreover
  have $\mathcal{C}(\!|label := [tn]\ @\ label\ \mathcal{C},\ trust\text{-}t := tr|\!) = \mathcal{C}(\!|trust\text{-}t := tr|\!)(\!|label := [tn]\ @\ label\ \mathcal{C}|\!)$
    by *simp*
  ultimately
  have $\mathcal{C}(\!|trust\text{-}t := tr|\!)(\!|label := [tn]\ @\ label\ \mathcal{C}|\!) \vdash es : (tn \text{ -> } tm)$
    by *metis*
  thus *?case*
    using *b-e-typing.loop[OF loop(1)]*
    by *simp*
next
  case (*if-wasm tf tn tm $\mathcal{C}$ es1 es2*)
  have $\mathcal{C}(\!|label := [tm]\ @\ label\ \mathcal{C},\ trust\text{-}t := tr|\!) \vdash es1 : (tn \text{ -> } tm)$
    $\mathcal{C}(\!|label := [tm]\ @\ label\ \mathcal{C},\ trust\text{-}t := tr|\!) \vdash es2 : (tn \text{ -> } tm)$
    using *if-wasm(4,5,6)*
    by *simp-all*
  moreover
  have $\mathcal{C}(\!|label := [tm]\ @\ label\ \mathcal{C},\ trust\text{-}t := tr|\!) = \mathcal{C}(\!|trust\text{-}t := tr|\!)(\!|label := [tm]\ @\ label\ \mathcal{C}|\!)$
    by *simp*
  ultimately
  have $\mathcal{C}(\!|trust\text{-}t := tr|\!)(\!|label := [tm]\ @\ label\ \mathcal{C}|\!) \vdash es1 : (tn \text{ -> } tm)$

61

$\mathcal{C}(|trust\text{-}t := tr|)(|label := [tm] @ label\ \mathcal{C}|) \vdash es2 : (tn \rightarrow tm)$

by *metis+*

thus *?case*

using *b-e-typing.if-wasm*$[OF\ if\text{-}wasm(1)]$

by *simp*

qed (*auto simp add*: *b-e-typing.intros trust-compat-def*)

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

$\mathcal{S} \cdot \mathcal{C} \vdash es : tf \implies trust\text{-}compat\ tr\ (trust\text{-}t\ \mathcal{C}) \implies \mathcal{S} \cdot \mathcal{C}(|trust\text{-}t := tr|) \vdash es : tf$

$\mathcal{S} \cdot tr' \cdot r \Vdash\text{-}i\ vs; es : ts \implies trust\text{-}compat\ tr\ tr' \implies \mathcal{S} \cdot tr \cdot r \Vdash\text{-}i\ vs; es : ts$

proof (*induction rule*: *e-typing-s-typing.inducts*)

  case (*1 $\mathcal{C}$ b-es tf $\mathcal{S}$*)

  thus *?case*

    using *b-e-typing-trust-compat e-typing-s-typing.intros(1)*

    by *simp*

next

  case (*6 $\mathcal{C}$ tr $\mathcal{S}$ cl tf*)

  thus *?case*

    using *e-typing-s-typing.intros(6)*

    unfolding *trust-compat-def*

    by *fastforce*

next

  case (*7 $\mathcal{S}$ $\mathcal{C}$ e0s ts t2s es n*)

  have $\mathcal{S} \cdot \mathcal{C}(|trust\text{-}t := tr|) \vdash e0s : (ts \rightarrow t2s)$

    $\mathcal{S} \cdot \mathcal{C}\ (|label := [ts] @ label\ \mathcal{C},\ trust\text{-}t := tr|) \vdash es : ([] \rightarrow t2s)$

    using *7(4,5,6)*

    by *simp-all*

  moreover

  have $\mathcal{C}\ (|label := [ts] @ label\ \mathcal{C},\ trust\text{-}t := tr|) = \mathcal{C}(|trust\text{-}t := tr|)(|label := [ts] @ label\ (\mathcal{C}(|trust\text{-}t := tr|))|)$

    by *simp*

  ultimately

  have $\mathcal{S} \cdot \mathcal{C}(|trust\text{-}t := tr|) \vdash e0s : (ts \rightarrow t2s)$

    $\mathcal{S} \cdot \mathcal{C}(|trust\text{-}t := tr|)(|label := [ts] @ label\ (\mathcal{C}(|trust\text{-}t := tr|))|) \vdash es : ([] \rightarrow t2s)$

  by *metis+*

  thus *?case*

    using *e-typing-s-typing.intros(7) 7(3)*

    by *fastforce*

next

  case (*8 i $\mathcal{S}$ tvs vs $\mathcal{C}$ rs tr' es ts*)

  have *trust-compat tr* (*trust-t $\mathcal{C}$*)

    using *8(3,7)*

    by *simp*

  hence $\mathcal{S} \cdot \mathcal{C}(|trust\text{-}t := tr|) \vdash es : ([] \rightarrow ts)$

    using *8(6)*

    by *simp*

  moreover

  have $\mathcal{C}(|trust\text{-}t := tr|) = (s\text{-}inst\ \mathcal{S}\ !\ i)\ (|trust\text{-}t := tr,\ local := local\ (s\text{-}inst\ \mathcal{S}\ !\ i) @ tvs,\ return := rs|)$

    using *8(3)*
    by *simp*
  ultimately
  show *?case*
    using *e-typing-s-typing.intros(8)[OF 8(1,2) - - 8(5)]*
    by *fastforce*
qed (*simp-all add*: *e-typing-s-typing.intros*)

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** $t = typeof\ v'$
  using *assms*
  unfolding *cvt-def typeof-def*
  apply (*cases t*)
   apply (*simp add*: *option.case-eq-if, metis option.discI option.inject v.simps(17)*)
   apply (*simp add*: *option.case-eq-if, metis option.discI option.inject v.simps(18)*)
   apply (*simp add*: *option.case-eq-if, metis option.discI option.inject v.simps(19)*)
   apply (*simp add*: *option.case-eq-if, metis option.discI option.inject v.simps(20)*)
  done

lemma *store-preserved1*:
  assumes $(\!|s;vs;es|\!)\ a\rightsquigarrow\text{-}i\ (\!|s';vs';es'|\!)$
      store-typing s $\mathcal{S}$
      $\mathcal{S}\cdot\mathcal{C} \vdash es : (ts \to ts')$
      $\mathcal{C} = ((s\text{-}inst\ \mathcal{S})!i)(\!|trust\text{-}t := tr,\ local := local\ ((s\text{-}inst\ \mathcal{S})!i)\ @\ (map\ typeof$
vs), label := arb-label, return := arb-return$|\!)$
      $i < length\ (s\text{-}inst\ \mathcal{S})$
  shows *store-typing s' $\mathcal{S}$*
  using *assms*
proof (*induction arbitrary*: $\mathcal{C}$ *tr arb-label arb-return ts ts' rule*: *reduce.induct*)
  case (*callcl-host-Some cl tr t1s t2s f ves vcs n m s i s' vcs' vs*)
  obtain ts'' **where** $ts''\text{-}def$: $\mathcal{S}\cdot\mathcal{C} \vdash ves : (ts \to ts'')$ $\mathcal{S}\cdot\mathcal{C} \vdash [Callcl\ cl] : (ts'' \to ts')$
  using *callcl-host-Some(8) e-type-comp*
  by *fastforce*
  have *ves-c*:*const-list ves*
    using *is-const-list[OF callcl-host-Some(2)]*
    by *simp*
  then obtain *tvs* **where** *tvs-def*:$ts'' = ts\ @\ tvs$
                      length t1s = length tvs
                        $\mathcal{S}\cdot\mathcal{C} \vdash ves : ([] \to tvs)$
    using *ts''-def(1) e-type-const-list[of ves $\mathcal{S}$ $\mathcal{C}$ ts ts''] callcl-host-Some*
    by *fastforce*
  hence $ts'' = ts\ @\ t1s$

63

       ts′ = ts @ t2s
   using *e-type-callcl-host*[*OF ts″-def*(*2*) *callcl-host-Some*(*1*)]
   by *auto*
  moreover
  hence *list-all2 types-agree t1s vcs*
   using *e-typing-imp-list-types-agree*[**where** *?ts′* = []] *callcl-host-Some*(*2*) *tvs-def*(*1*,*3*)
   by *fastforce*
  thus *?case*
   using *store-extension-imp-store-typing*
       host-apply-preserve-store[*OF - callcl-host-Some*(*6*)] *callcl-host-Some*(*7*)
   by *fastforce*
next
  case (*set-global s i j v s′ vs*)
  obtain *insts fs clss bss gs* **where** *s* = (|*inst* = *insts, funcs* = *fs, tab* = *clss, mem* = *bss, globs* = *gs*|)
   using *s.cases*
   by *blast*
  moreover
  obtain *insts′ fs′ clss′ bss′ gs′* **where** *s′* = (|*inst* = *insts′, funcs* = *fs′, tab* = *clss′, mem* = *bss′, globs* = *gs′*|)
   using *s.cases*
   by *blast*
  moreover
  obtain $\mathcal{C}s$ *tfs ns ms tgs* **where** $\mathcal{S}$ = (|*s-inst* = $\mathcal{C}s$, *s-funcs* = *tfs, s-tab* = *ns, s-mem* = *ms, s-globs* = *tgs*|)
   using *s-context.cases*
   by *blast*
  moreover

  note *s-S-defs* = *calculation*

  have
  insts = *insts′*
  fs = *fs′*
  clss = *clss′*
  bss = *bss′*
   using *set-global*(*1*) *s-S-defs*(*1*,*2*)
   unfolding *supdate-glob-def supdate-glob-s-def*
   by (*metis s.ext-inject s.update-convs*(*5*))+
  hence
  list-all2 (*inst-typing* $\mathcal{S}$) *insts′* $\mathcal{C}s$
  list-all2 (*cl-typing* $\mathcal{S}$) *fs′ tfs*
  list-all (*tab-agree* $\mathcal{S}$) (*concat clss′*)
  list-all2 (λ*cls n. n* $\leq$ *length cls*) *clss′ ns*
  list-all2 mem-agree bss′ ms
   using *set-global*(*2*) *s-S-defs*
   unfolding *store-typing.simps*
   by *auto*

64

moreover
have *list-all2 glob-agree gs′ tgs*
proof −
  have *gs-agree*:*list-all2 glob-agree gs tgs*
    using *set-global(2) s-S-defs*
    unfolding *store-typing.simps*
    by *auto*
  have *length gs = length gs′*
    using *s-S-defs(1,2) set-global(1)*
    unfolding *supdate-glob-def supdate-glob-s-def*
    by (*metis length-list-update s.select-convs(5) s.update-convs(5)*)
  moreover
  obtain *k* **where** *k-def*:(*sglob-ind s i j*) *= k*
    by *blast*
  hence $\bigwedge$*j′*. ⟦*j′ ≠ k; j′ < length gs*⟧ $\Longrightarrow$ *gs*!*j′ = gs′*!*j′*
    using *s-S-defs(1,2) set-global(1)*
    unfolding *supdate-glob-def supdate-glob-s-def*
    by *auto*
  hence $\bigwedge$*j′*. ⟦*j′ ≠ k; j′ < length gs*⟧ $\Longrightarrow$ *glob-agree* (*gs′*!*j′*) (*tgs*!*j′*)
    using *gs-agree*
    by (*metis list-all2-conv-all-nth*)
  moreover
  have *glob-agree* (*gs′*!*k*) (*tgs*!*k*)
  proof −
    obtain *ts″* **where** *ts″-def*:$\mathcal{C}$ ⊢ [*C v*] : (*ts* -> *ts″*) $\mathcal{C}$ ⊢ [*Set-global j*] : (*ts″* -> *ts′*)
      by (*metis b-e-type-comp2-unlift set-global.prems(2)*)
    have *b-es*:*ts″ = ts*@[*typeof v*]
        ts = ts′
        global $\mathcal{C}$! *j* = (|*tg-mut = T-mut, tg-t = typeof v*|)
        j < length (*global* $\mathcal{C}$)
      using *b-e-type-value*[*OF ts″-def(1)*] *b-e-type-set-global*[*OF ts″-def(2)*]
      by *auto*
    hence *j < length* (*global* ((*s-inst* $\mathcal{S}$)!*i*))
      using *set-global(4)*
      by *fastforce*
    hence *globs-agree*:*k < length* (*s-globs* $\mathcal{S}$)
          glob-agree (*gs*!*k*) (*tgs*!*k*)
          (*tgs*!*k*) = (*global* $\mathcal{C}$)!*j*
      using *store-typing-imp-glob-agree*[*OF set-global(2,5)*] *b-es(4) s-S-defs(1,3)*
  k-def set-global(4)
      unfolding *sglob-def*
      by *auto*
    hence *g-mut* (*gs*!*k*) *= T-mut*
      typeof (*g-val* (*gs*!*k*)) *= typeof v*
      using *b-es(3)*
      unfolding *glob-agree-def*
      by *auto*
    hence *g-mut* (*gs′*!*k*) *= T-mut*

$typeof\ (g\text{-}val\ (gs'!k)) = typeof\ v$
        using *set-global(1) k-def globs-agree(1) store-typing-imp-glob-length-eq[OF set-global(2)] s-S-defs(1,2)*
        unfolding *supdate-glob-def supdate-glob-s-def*
        by *auto*
      thus *?thesis*
        using *globs-agree(3) b-es(3)*
        unfolding *glob-agree-def*
        by *fastforce*
    qed
    ultimately
    show *?thesis*
      using *gs-agree*
      unfolding *list-all2-conv-all-nth*
      by *fastforce*
  qed
  ultimately
  show *?case*
    using *store-typing.intros*
    by *simp*
next

  case (*store-Some t v s i j m sec' k off mem' vs sec a*)
  show *?case*
    using *store-preserved-mem[OF store-Some(5) - - store-Some(3)] store-size[OF store-Some(4)]*
    by *fastforce*
next
  case (*store-packed-Some t v tp s i m sec k off mem' vs a*)
  thus *?case*
    using *store-preserved-mem[OF store-packed-Some(5) - - store-packed-Some(3)] store-packed-size[OF store-packed-Some(4)]*
    by *simp*
next
  case (*grow-memory s i n mem sec c mem' vs*)
  show *?case*
   using *store-preserved-mem[OF grow-memory(5)- - grow-memory(2)] mem-grow-size[OF grow-memory(4)]*
    by *simp*
next
  case (*label s vs es a i s' vs' es' k lholed les les'*)
  obtain $\mathcal{C}'$ *t1s t2s arb-label' arb-return'* **where** *es-def*:$\mathcal{C}' = \mathcal{C}(\![label := arb\text{-}label',$
$return := arb\text{-}return'\!])$ $\mathcal{S} \cdot \mathcal{C}' \vdash es : (t1s \rightarrow t2s)$
    using *types-exist-lfilled-weak[OF label(2,6)]*
    by *fastforce*
  thus *?case*
    using *label(4)[OF label(5) es-def(2) - label(8)] label(7)*
    by *fastforce*
next

case (*local s vs es a i s′ vs′ es′ v0s n j*)
  obtain *tls* **where** *t-local*:($\mathcal{S}$·((s-*inst* $\mathcal{S}$)!i)($trust$-t:= *trust-t* $\mathcal{C}$, *local* := (*local* ((s-*inst* $\mathcal{S}$)!i)) @ (*map typeof vs*), *return* := *Some tls*) ⊢ *es* : ([] -> *tls*))
                    ts′ = *ts* @ *tls i* < *length* (s-*inst* $\mathcal{S}$)
    using *e-type-local*[*OF local*(4)]
    by *blast*+
  show *?case*
    using *local*(2)[*OF local*(3) *t-local*(1) - *t-local*(3), *of* (*Some tls*) *label* ((s-*inst* $\mathcal{S}$)!i)]
    by *fastforce*
qed (*simp-all*)

lemma *store-preserved*:
  assumes (|s;vs;es|) $a\rightsquigarrow$-i (|$s′$;$vs′$;$es′$|)
          store-typing s $\mathcal{S}$
          $\mathcal{S}$·tr·*None* ⊩-i *vs*;*es* : *ts*
  shows *store-typing s′* $\mathcal{S}$
proof −
  show *?thesis*
    using *store-preserved1*[*OF assms*(1,2), *of* - [] *ts None label* (s-*inst* $\mathcal{S}$!i)]
        s-type-unfold[*OF assms*(3)]
    by *fastforce*
qed

lemma *typeof-unop-testop*:
  assumes $\mathcal{S}$·$\mathcal{C}$ ⊢ [\$$C$ v, \$$e$] : (*ts* -> *ts′*)
        (e = (*Unop-i t iop*)) ∨ (e = (*Unop-f t fop*)) ∨ (e = (*Testop t testop*))
  shows (*typeof v*) = *t*
        e = (*Unop-f t fop*) ⟹ *is-float-t t*
proof −
  have $\mathcal{C}$ ⊢ [C v, e] : (*ts* -> *ts′*)
    using *unlift-b-e assms*(1)
    by *simp*
  then obtain *ts″* **where** *ts″-def*:$\mathcal{C}$ ⊢ [C v] : (*ts* -> *ts″*) $\mathcal{C}$ ⊢ [e] : (*ts″* -> *ts′*)
    using *b-e-type-comp*[**where** *?e* = *e* **and** *?es* = [C v]]
    by *fastforce*
  show (*typeof v*) = *t* e = (*Unop-f t fop*) ⟹ *is-float-t t*
   using *b-e-type-value*[*OF ts″-def*(1)] *assms*(2) *b-e-type-unop-testop*[*OF ts″-def*(2)]
    by *simp-all*
qed

lemma *typeof-cvtop*:
  assumes $\mathcal{S}$·$\mathcal{C}$ ⊢ [\$$C$ v, \$$e$] : (*ts* -> *ts′*)
        e = *Cvtop t1 cvtop t sx*
  shows (*typeof v*) = *t*
        cvtop = *Convert* ⟹ (*t1* ≠ *t*) ∧ (*t-sec t1* = *t-sec t*) ∧ ((*sx* = *None*) = ((*is-float-t t1* ∧ *is-float-t t*) ∨ (*is-int-t t1* ∧ *is-int-t t* ∧ (*t-length t1* < *t-length t*))))
        cvtop = *Reinterpret* ⟹ (*t1* ≠ *t*) ∧ *t-sec t1* = *t-sec t* ∧ *t-length t1* = *t-length t*

$cvtop = Classify \implies is\text{-}int\text{-}t\ t \land is\text{-}public\text{-}t\ t \land classify\text{-}t\ t = t1$

$cvtop = Declassify \implies (trust\text{-}t\ \mathcal{C}) = Trusted \land is\text{-}int\text{-}t\ t \land is\text{-}secret\text{-}t\ t \land$
$declassify\text{-}t\ t = t1$

proof $-$

  have $\mathcal{C} \vdash [C\ v,\ e] : (ts \text{ -> } ts')$

    using *unlift-b-e assms(1)*

    by *simp*

  then obtain ts'' **where** $ts''\text{-}def$:$\mathcal{C} \vdash [C\ v] : (ts \text{ -> } ts'')\ \mathcal{C} \vdash [e] : (ts'' \text{ -> } ts')$

    using *b-e-type-comp*[**where** *?e = e* **and** *?es = [C v]*]

    by *fastforce*

  show $(typeof\ v) = t$

    $cvtop = Convert \implies (t1 \neq t) \land t\text{-}sec\ t1 = t\text{-}sec\ t \land (sx = None) = ((is\text{-}float\text{-}t$
$t1 \land is\text{-}float\text{-}t\ t) \lor (is\text{-}int\text{-}t\ t1 \land is\text{-}int\text{-}t\ t \land (t\text{-}length\ t1 < t\text{-}length\ t)))$

    $cvtop = Reinterpret \implies (t1 \neq t) \land t\text{-}sec\ t1 = t\text{-}sec\ t \land t\text{-}length\ t1 = t\text{-}length$
t

    $cvtop = Classify \implies is\text{-}int\text{-}t\ t \land is\text{-}public\text{-}t\ t \land classify\text{-}t\ t = t1$

    $cvtop = Declassify \implies (trust\text{-}t\ \mathcal{C}) = Trusted \land is\text{-}int\text{-}t\ t \land is\text{-}secret\text{-}t\ t \land$
$declassify\text{-}t\ t = t1$

    using *b-e-type-value*[*OF ts''-def(1)*] *b-e-type-cvtop*[*OF ts''-def(2) assms(2)*]

    by *simp-all*

qed

lemma *typeof-callcl-host*:

  assumes $\mathcal{S}\cdot\mathcal{C} \vdash (\$\$* \ vs)\ @\ [e] : (ts \text{ -> } ts')$

      $e = Callcl\ cl$

      $cl = Func\text{-}host\ (tr,tf)\ f$

  shows *trust-compat* $(trust\text{-}t\ \mathcal{C})\ tr$

proof $-$

    obtain ts'' **where** $ts'\text{-}def$:$\mathcal{S}\cdot\mathcal{C} \vdash \$\$* \ vs: (ts \text{ -> } ts'')\ \mathcal{S}\cdot\mathcal{C} \vdash [e] : (ts'' \text{ -> } ts')$

      using *e-type-comp*[*OF assms(1)*]

      by *fastforce*

    thus *?thesis*

      using *assms(2,3) e-type-callcl*

      unfolding *cl-type-def*

      by *fastforce*

qed

lemma *types-preserved-unop-testop-cvtop*:

  assumes $(\![\$C\ v,\ \$e]\!)\ a \rightsquigarrow (\![\$C\ v']\!)$

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

      $(e = (Unop\text{-}i\ t\ iop)) \lor (e = (Unop\text{-}f\ t\ fop)) \lor (e = (Testop\ t\ testop)) \lor$
$(e = (Cvtop\ t2\ cvtop\ t\ sx))$

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

proof $-$

  have $\mathcal{C} \vdash [C\ v,\ e] : (ts \text{ -> } ts')$

    using *unlift-b-e assms(2)*

    by *simp*

  then obtain ts'' **where** $ts''\text{-}def$:$\mathcal{C} \vdash [C\ v] : (ts \text{ -> } ts'')\ \mathcal{C} \vdash [e] : (ts'' \text{ -> } ts')$

    using *b-e-type-comp*[**where** *?e = e* **and** *?es = [C v]*]

    by *fastforce*
  have *ts*@[*arity-1-result e*] = *ts′* (*typeof v*) = *t*
   using *b-e-type-value*[*OF ts″-def*(*1*)] *assms*(*3*) *b-e-type-unop-testop*(*1*)[*OF ts″-def*(*2*)]
        b-e-type-cvtop(*1*)[*OF ts″-def*(*2*)]
   by (*metis butlast-snoc, metis last-snoc*)
  moreover
  have *arity-1-result e* = *typeof* (*v′*)
   using *assms*(*1,3*)
   apply (*cases rule*: *reduce-simple.cases*)
       apply (*simp-all add*: *arity-1-result-def wasm-deserialise-type t-cvt*)
        apply (*auto simp add*: *typeof-def t-sec-def classify-t-classify-typeof*
declassify-t-declassify-typeof)
   done
  hence $\mathcal{C} \vdash [C\ v′] : ([]$ -> [*arity-1-result e*])
   using *b-e-typing.const*
   by *metis*
  ultimately
  show $S{\cdot}\mathcal{C} \vdash [\$C\ v′] : (ts$ -> $ts′)$
   using *e-typing-s-typing.intros*(*1*)
      b-e-typing.weakening[*of* $\mathcal{C}$ [*C v′*] [] [*arity-1-result e*] *ts*]
   by *fastforce*
qed

lemma *typeof-binop-relop*:
  assumes $S{\cdot}\mathcal{C} \vdash [\$C\ v1, \$C\ v2, \$e] : (ts$ -> $ts′)$
      $e = Binop\text{-}i\ t\ iop \lor e = Binop\text{-}f\ t\ fop \lor e = Relop\text{-}i\ t\ irop \lor e = Relop\text{-}f$
$t\ frop$
  shows *typeof v1* = *t*
     typeof v2 = *t*
     $e = Binop\text{-}i\ t\ iop \implies is\text{-}secret\text{-}t\ t \implies safe\text{-}binop\text{-}i\ iop$
     $e = Binop\text{-}f\ t\ fop \implies is\text{-}float\text{-}t\ t$
     $e = Relop\text{-}f\ t\ frop \implies is\text{-}float\text{-}t\ t$
proof −
  have $\mathcal{C} \vdash [C\ v1,\ C\ v2,\ e] : (ts$ -> $ts′)$
   using *unlift-b-e assms*(*1*)
   by *simp*
  then obtain *ts″* **where** *ts″-def*:$\mathcal{C} \vdash [C\ v1,\ C\ v2] : (ts$ -> $ts″)\ \mathcal{C} \vdash [e] : (ts″$ ->
$ts′)$
   using *b-e-type-comp*[**where** *?e* = *e* **and** *?es* = [*C v1, C v2*]]
   by *fastforce*
  then obtain *ts-id* **where** *ts-id-def*:*ts-id*@[*t,t*] = *ts″ ts′* = *ts-id* @ [*arity-2-result*
e]
                       $e = Binop\text{-}i\ t\ iop \implies is\text{-}secret\text{-}t\ t \implies safe\text{-}binop\text{-}i\ iop$
                       $e = Binop\text{-}f\ t\ fop \implies is\text{-}float\text{-}t\ t$
                       $e = Relop\text{-}f\ t\ frop \implies is\text{-}float\text{-}t\ t$
   using *assms*(*2*) *b-e-type-binop-relop*[*of* $\mathcal{C}$ *e ts″ ts′ t*]
   by *blast*
  thus *typeof v1* = *t*
     typeof v2 = *t*

$e = \text{Binop-i } t \text{ iop} \implies \text{is-secret-t } t \implies \text{safe-binop-i iop}$
$e = \text{Binop-f } t \text{ fop} \implies \text{is-float-t } t$
$e = \text{Relop-f } t \text{ frop} \implies \text{is-float-t } t$
  using *ts''-def b-e-type-comp*[*of* $\mathcal{C}$ [*C v1*] *C v2 ts ts''*] *b-e-type-value2*
  by *fastforce+*
qed

lemma *types-preserved-binop-relop*:
 assumes $([\$C\ v1,\ \$C\ v2,\ \$e]) \ a \rightsquigarrow ([\$C\ v'])$
   $\mathcal{S} \cdot \mathcal{C} \vdash [\$C\ v1,\ \$C\ v2,\ \$e] : (ts \to ts')$
   $e = \text{Binop-i } t \text{ iop} \lor e = \text{Binop-f } t \text{ fop} \lor e = \text{Relop-i } t \text{ irop} \lor e = \text{Relop-f}$
$t \text{ frop}$
 shows $\mathcal{S} \cdot \mathcal{C} \vdash [\$C\ v'] : (ts \to ts')$
proof −
 have $\mathcal{C} \vdash [C\ v1,\ C\ v2,\ e] : (ts \to ts')$
  using *unlift-b-e assms(2)*
  by *simp*
 then obtain ts'' **where** *ts''-def*:$\mathcal{C} \vdash [C\ v1,\ C\ v2] : (ts \to ts'')\ \mathcal{C} \vdash [e] : (ts'' \to$
$ts')$
  using *b-e-type-comp*[**where** *?e = e* **and** *?es = [C v1, C v2]*]
  by *fastforce*
 then obtain *ts-id* **where** *ts-id-def*:*ts-id@[t,t]* = ts'' ts' = *ts-id @ [arity-2-result*
$e]$
  using *assms(3) b-e-type-binop-relop*[*of* $\mathcal{C}$ *e ts'' ts' t*]
  by *blast*
 hence $\mathcal{C} \vdash [C\ v1] : (ts \to ts\text{-}id@[t])$
  using *ts''-def b-e-type-comp*[*of* $\mathcal{C}$ [*C v1*] *C v2 ts ts''*] *b-e-type-value*
   by *fastforce*
 hence *ts@[arity-2-result e]* = ts'
  using *b-e-type-value ts-id-def(2)*
  by *fastforce*
 moreover
 have *arity-2-result e* = *typeof* (v')
  using *assms(1,3)*
  by (*cases rule: reduce-simple.cases*) (*auto simp add: arity-2-result-def typeof-def*
t-sec-def)
 hence $\mathcal{C} \vdash [C\ v'] : ([] \to [arity\text{-}2\text{-}result\ e])$
  using *b-e-typing.const*
  by *metis*
 ultimately show *?thesis*
  using *e-typing-s-typing.intros(1)*
   b-e-typing.weakening[*of* $\mathcal{C}$ [*C v'*] [] [*arity-2-result e*] *ts*]
  by *fastforce*
qed

lemma *types-preserved-drop*:
 assumes $([\$C\ v,\ \$e]) \ a \rightsquigarrow ([[]])$
   $\mathcal{S} \cdot \mathcal{C} \vdash [\$C\ v,\ \$e] : (ts \to ts')$
   $(e = (Drop))$

shows $S \cdot C \vdash [] : (ts \rightarrow ts')$
proof −
　have $C \vdash [C\ v,\ e] : (ts \rightarrow ts')$
　　using *unlift-b-e assms(2)*
　　by *simp*
　then obtain ts'' **where** $ts''\text{-}def : C \vdash [C\ v] : (ts \rightarrow ts'')\ C \vdash [e] : (ts'' \rightarrow ts')$
　　using *b-e-type-comp*[**where** $?e = e$ **and** $?es = [C\ v]$]
　　by *fastforce*
　hence $ts'' = ts@[typeof\ v]$
　　using *b-e-type-value*
　　by *blast*
　hence $ts = ts'$
　　using $ts''\text{-}def$ *assms(3) b-e-type-drop*
　　by *blast*
　hence $C \vdash [] : (ts \rightarrow ts')$
　　using *b-e-type-empty*
　　by *simp*
　thus *?thesis*
　　using *e-typing-s-typing.intros(1)*
　　by *fastforce*
qed

lemma *typeof-select*:
　assumes $S \cdot C \vdash [\$C\ v1,\ \$C\ v2,\ \$C\ vn,\ \$e] : (ts \rightarrow ts')$
　　　　$(e = Select\ sec)$
　shows *t-sec* $(typeof\ vn) = sec$
　　　　$typeof\ v1 = typeof\ v2$
　　　　$sec = Secret \longrightarrow$ *is-secret-t* $(typeof\ v1)$
　　　　$sec = Secret \longrightarrow$ *is-secret-t* $(typeof\ v2)$
proof −
　have $C \vdash [C\ v1,\ C\ v2,\ C\ vn,\ e] : (ts \rightarrow ts')$
　　using *unlift-b-e assms(1)*
　　by *simp*
　then obtain $t1s$ **where** $t1s\text{-}def : C \vdash [C\ v1,\ C\ v2,\ C\ vn] : (ts \rightarrow t1s)\ C \vdash [e] :$
$(t1s \rightarrow ts')$
　　using *b-e-type-comp*[**where** $?e = e$ **and** $?es = [C\ v1,\ C\ v2,\ C\ vn]$]
　　by *fastforce*
　then obtain $t2s\ t$ **where** $t2s\text{-}def : t1s = t2s\ @\ [t,\ t,\ (T\text{-}i32\ sec)]$
　　　　　　　　$ts' = t2s@[t]$
　　　　　　　　$(sec = Secret \longrightarrow$ *is-secret-t* $t)$
　　using *b-e-type-select*[*of* $C\ e\ t1s$] *assms*
　　by *fastforce*
　thus $sec = Secret \longrightarrow$ *is-secret-t* $(typeof\ v1)$
　　　　t-sec $(typeof\ vn) = sec$
　　　　$typeof\ v1 = typeof\ v2$
　　　　$sec = Secret \longrightarrow$ *is-secret-t* $(typeof\ v2)$
　　　using $t1s\text{-}def\ t2s\text{-}def$ *b-e-type-value-list*[*of* $C\ [C\ v1,\ C\ v2]\ vn\ ts\ t2s@[t,t]$]
t-sec-def
　　　　　b-e-type-value2[*of* $C\ v1\ v2$]

by *fastforce+*

qed

lemma *types-preserved-select*:
  assumes (|[$C v1, $C v2, $C vn, $e]|) a⤳ (|[$C v3]|)
       S·C ⊢ [$C v1, $C v2, $C vn, $e] : (ts -> ts′)
       (e = Select sec)
  shows S·C ⊢ [$C v3] : (ts -> ts′)
proof −
  have C ⊢ [C v1, C v2, C vn, e] : (ts -> ts′)
    using *unlift-b-e assms(2)*
    by *simp*
  then obtain *t1s* **where** *t1s-def*:C ⊢ [C v1, C v2, C vn] : (ts -> t1s) C ⊢ [e] :
(t1s -> ts′)
    using *b-e-type-comp*[**where** *?e = e* **and** *?es = [C v1, C v2, C vn]*]
    by *fastforce*
  then obtain *t2s t sec* **where** *t2s-def*:t1s = t2s @ [t, t, (T-i32 sec)] ts′ = t2s@[t]
    using *b-e-type-select*[*of C e t1s*] *assms*
    by *fastforce*
  hence C ⊢ [C v1, C v2] : (ts -> t2s@[t,t])
    using *t1s-def t2s-def b-e-type-value-list*[*of C [C v1, C v2] vn ts t2s@[t,t]*]
    by *fastforce*
  hence *v2-t-def*:C ⊢ [C v1] : (ts -> t2s@[t]) typeof v2 = t
    using *t1s-def t2s-def b-e-type-value-list*[*of C [C v1] v2 ts t2s@[t]*]
    by *fastforce+*
  hence *v1-t-def*:ts = t2s typeof v1 = t
    using *b-e-type-value*
    by *fastforce+*
  have *typeof v3 = t*
    using *assms(1) v2-t-def(2) v1-t-def(2)*
    by (*cases rule: reduce-simple.cases, simp-all*)
  hence C ⊢ [C v3] : (ts -> ts′)
    using *b-e-typing.const b-e-typing.weakening t2s-def(2) v1-t-def(1)*
    by *fastforce*
  thus *?thesis*
    using *e-typing-s-typing.intros(1)*
    by *fastforce*
qed

lemma *types-preserved-block*:
  assumes (|vs @ [$Block (tn -> tm) es]|) a⤳ (|[Label m [] (vs @ ($∗ es))]|)
       S·C ⊢ vs @ [$Block (tn -> tm) es] : (ts -> ts′)
       const-list vs
       length vs = n
       length tn = n
       length tm = m
  shows S·C ⊢ [Label m [] (vs @ ($∗ es))] : (ts -> ts′)
proof −

obtain $\mathcal{C}'$ **where** *c-def*:$\mathcal{C}' = \mathcal{C}(\!|label := [tm] @ label \; \mathcal{C}|\!)$ **by** *blast*
  obtain *ts''* **where** *ts''-def*:$\mathcal{S}\cdot\mathcal{C} \vdash vs : (ts \rightarrow ts'')$ $\mathcal{S}\cdot\mathcal{C} \vdash [\$Block \; (tn \rightarrow tm) \; es] :$
$(ts'' \rightarrow ts')$
    using *assms(2)* *e-type-comp*[*of* $\mathcal{S}$ $\mathcal{C}$ *vs* $\$Block \; (tn \rightarrow tm) \; es \; ts \; ts'$]
    by *fastforce*
  hence $\mathcal{C} \vdash [Block \; (tn \rightarrow tm) \; es] : (ts'' \rightarrow ts')$
    using *unlift-b-e*
    by *auto*
   then obtain *ts-c* *tfn* *tfm* **where** *ts-c-def*:$(tn \rightarrow tm) = (tfn \rightarrow tfm)$ $ts'' =$
$ts\text{-}c@tfn$ $ts' = ts\text{-}c@tfm$ $(\mathcal{C}(\!|label := [tfm] @ label \; \mathcal{C}|\!) \vdash es : (tn \rightarrow tm))$
    using *b-e-type-block*[*of* $\mathcal{C}$ *Block* $(tn \rightarrow tm) \; es \; ts'' \; ts' \; (tn \rightarrow tm) \; es$]
    by *fastforce*
  hence *tfn-l*:*length tfn = n*
    using *assms(5)*
    by *simp*
  obtain *tvs'* **where** *tvs'-def*:*ts''* $= ts@tvs'$ *length tvs' = n* $\mathcal{S}\cdot\mathcal{C}' \vdash vs : ([] \rightarrow tvs')$
    using *e-type-const-list* *assms(3,4)* *ts''-def(1)*
    by *fastforce*
  hence $\mathcal{S}\cdot\mathcal{C}' \vdash vs : ([] \rightarrow tn)$ $\mathcal{S}\cdot\mathcal{C}' \vdash \$*es : (tn \rightarrow tm)$
    using *ts-c-def* *tvs'-def* *tfn-l* *ts''-def* *c-def* *e-typing-s-typing.intros(1)*
    by *simp-all*
  hence $\mathcal{S}\cdot\mathcal{C}' \vdash (vs @ (\$* es)) : ([] \rightarrow tm)$ **using** *e-type-comp-conc*
    by *simp*
  moreover
  have $\mathcal{S}\cdot\mathcal{C} \vdash [] : (tm \rightarrow tm)$
    using *b-e-type-empty*[*of* $\mathcal{C}$ [] []]
        e-typing-s-typing.intros(1)[**where** *?b-es = []*]
        e-typing-s-typing.intros(3)[*of* $\mathcal{S}$ $\mathcal{C}$ [] [] [] *tm*]
    by *fastforce*
  ultimately
  show *?thesis*
    using *e-typing-s-typing.intros(7)*[*of* $\mathcal{S}$ $\mathcal{C}$ [] *tm - vs @ (\$* es) m*]
        ts-c-def *tvs'-def* *assms(5,6)* *e-typing-s-typing.intros(3)* *c-def*
    by *fastforce*
qed

lemma *typeof-if*:
  assumes $\mathcal{S}\cdot\mathcal{C} \vdash [\$C \; ConstInt32 \; sec \; n, \$If \; tf \; e1s \; e2s] : (ts \rightarrow ts')$
  shows *sec = Public*
proof $-$
  have $\mathcal{C} \vdash [C \; ConstInt32 \; sec \; n, If \; tf \; e1s \; e2s] : (ts \rightarrow ts')$
    using *unlift-b-e* *assms*
    by *fastforce*
  then obtain *ts-i* **where** *ts-i-def*:$\mathcal{C} \vdash [C \; ConstInt32 \; sec \; n] : (ts \rightarrow ts\text{-}i)$ $\mathcal{C} \vdash [If$
$tf \; e1s \; e2s] : (ts\text{-}i \rightarrow ts')$
    using *b-e-type-comp*
    by (*metis append-Cons append-Nil*)
  thus *?thesis*
    using *b-e-type-if*[*OF ts-i-def(2)*] *b-e-type-value*[*OF ts-i-def(1)*]

    unfolding *typeof-def*
    by *fastforce*
qed

lemma *types-preserved-if*:
  assumes $(\![\$C\ ConstInt32\ sec\ n,\ \$If\ tf\ e1s\ e2s]\!)\ a\leadsto (\![\$Block\ tf\ es'\,]\!)$
       $\mathcal{S} \cdot \mathcal{C} \vdash [\$C\ ConstInt32\ sec\ n,\ \$If\ tf\ e1s\ e2s] : (ts \rightarrow ts')$
  shows $\mathcal{S} \cdot \mathcal{C} \vdash [\$Block\ tf\ es'\,] : (ts \rightarrow ts')$
proof −
  have $\mathcal{C} \vdash [C\ ConstInt32\ sec\ n,\ If\ tf\ e1s\ e2s] : (ts \rightarrow ts')$
    using *unlift-b-e assms(2)*
    by *fastforce*
  then obtain *ts-i* **where** *ts-i-def*:$\mathcal{C} \vdash [C\ ConstInt32\ sec\ n] : (ts \rightarrow ts\text{-}i)\ \mathcal{C} \vdash [If$
tf e1s e2s$] : (ts\text{-}i \rightarrow ts')$
    using *b-e-type-comp*
    by (*metis append-Cons append-Nil*)
  then obtain ts'' *tfn tfm* **where** *ts-def*:$tf = (tfn \rightarrow tfm)$
                       $ts\text{-}i = ts''@tfn\ @\ [(T\text{-}i32\ Public)]$
                       $ts' = ts''@tfm$
                       $(\mathcal{C}(\![label := [tfm]\ @\ label\ \mathcal{C}]\!) \vdash e1s : tf)$
                       $(\mathcal{C}(\![label := [tfm]\ @\ label\ \mathcal{C}]\!) \vdash e2s : tf)$
    using *b-e-type-if*[*of* $\mathcal{C}$ *If tf e1s e2s*]
    by *fastforce*
  have $ts\text{-}i = ts\ @\ [(T\text{-}i32\ sec)]$
    using *ts-i-def(1) b-e-type-value*
    unfolding *typeof-def*
    by *fastforce*
  moreover
  have $(\mathcal{C}(\![label := [tfm]\ @\ label\ \mathcal{C}]\!) \vdash es' : (tfn \rightarrow tfm))$
    using *assms(1) ts-def(4,5) ts-def(1)*
    by (*cases rule*: *reduce-simple.cases*, *simp-all*)
  hence $\mathcal{C} \vdash [Block\ tf\ es'\,] : (tfn \rightarrow tfm)$
    using *ts-def(1) b-e-typing.block*[*of tf tfn tfm* $\mathcal{C}$ es']
    by *simp*
  ultimately
  show *?thesis*
    using *ts-def(2,3) e-typing-s-typing.intros(1,3)*
    by *fastforce*
qed

lemma *types-preserved-tee-local*:
  assumes $(\![v,\ \$Tee\text{-}local\ i]\!)\ a\leadsto (\![v,\ v,\ \$Set\text{-}local\ i]\!)$
       $\mathcal{S} \cdot \mathcal{C} \vdash [v,\ \$Tee\text{-}local\ i] : (ts \rightarrow ts')$
       is-const v
  shows $\mathcal{S} \cdot \mathcal{C} \vdash [v,\ v,\ \$Set\text{-}local\ i] : (ts \rightarrow ts')$
proof −
  obtain *bv* **where** *bv-def*:$v = \$C\ bv$
    using *e-type-const-unwrap assms(3)*
    by *fastforce*

hence $\mathcal{C} \vdash [C\ bv,\ Tee\text{-}local\ i] : (ts\ \text{-}>\ ts')$
  using *unlift-b-e assms(2)*
  by *fastforce*
then obtain ts'' **where** $ts''\text{-}def{:}\mathcal{C} \vdash [C\ bv] : (ts\ \text{-}>\ ts'')\ \mathcal{C} \vdash [Tee\text{-}local\ i] : (ts''\ \text{-}>\ ts')$
  using *b-e-type-comp[of - [C bv] Tee-local i]*
  by *fastforce*
then obtain $ts\text{-}c\ t$ **where** $ts\text{-}c\text{-}def{:}ts'' = ts\text{-}c@[t]\ ts' = ts\text{-}c@[t]\ (local\ \mathcal{C})!i = t$
$i < length(local\ \mathcal{C})$
  using *b-e-type-tee-local[of C Tee-local i ts'' ts' i]*
  by *fastforce*
hence $t\text{-}bv{:}t = typeof\ bv\ ts = ts\text{-}c$
  using *b-e-type-value ts''-def*
  by *fastforce+*
have $\mathcal{C} \vdash [Set\text{-}local\ i] : ([t,t]\ \text{-}>\ [t])$
  using *ts-c-def(3,4) b-e-typing.set-local[of i C t]*
     b-e-typing.weakening[of C [Set-local i] [t] [] [t]]
  by *fastforce*
moreover
have $\mathcal{C} \vdash [C\ bv] : ([t]\ \text{-}>\ [t,t])$
  using *t-bv b-e-typing.const[of C bv] b-e-typing.weakening[of C [C bv] [] [t] [t]]*
  by *fastforce*
hence $\mathcal{C} \vdash [C\ bv,\ C\ bv] : ([]\ \text{-}>\ [t,t])$
  using *t-bv b-e-typing.const[of C bv] b-e-typing.composition[of C [C bv] [] [t]]*
  by *fastforce*
ultimately
have $\mathcal{C} \vdash [C\ bv,\ C\ bv,\ Set\text{-}local\ i] : (ts\ \text{-}>\ ts@[t])$
  using *b-e-typing.composition b-e-typing.weakening[of C [C bv, C bv, Set-local i]]*
  by *fastforce*
thus *?thesis*
  using *t-bv(2) ts-c-def(2) bv-def e-typing-s-typing.intros(1)*
  by *fastforce*
qed

lemma *types-preserved-loop*:
  assumes $(\!|vs\ @\ [\$Loop\ (t1s\ \text{-}>\ t2s)\ es]|\!)\ a \leadsto (\!|[Label\ n\ [\$Loop\ (t1s\ \text{-}>\ t2s)\ es]$
$(vs\ @\ (\$* \ es))]|\!)$
       $\mathcal{S}\cdot\mathcal{C} \vdash vs\ @\ [\$Loop\ (t1s\ \text{-}>\ t2s)\ es] : (ts\ \text{-}>\ ts')$
       const-list vs
       length vs = n
       length t1s = n
       length t2s = m
  shows $\mathcal{S}\cdot\mathcal{C} \vdash [Label\ n\ [\$Loop\ (t1s\ \text{-}>\ t2s)\ es]\ (vs\ @\ (\$* \ es))] : (ts\ \text{-}>\ ts')$
proof −
  obtain ts'' **where** $ts''\text{-}def{:}\mathcal{S}\cdot\mathcal{C} \vdash vs : (ts\ \text{-}>\ ts'')\ \mathcal{S}\cdot\mathcal{C} \vdash [\$Loop\ (t1s\ \text{-}>\ t2s)\ es]$
$: (ts''\ \text{-}>\ ts')$
  using *assms(2) e-type-comp*
  by *fastforce*

then have $\mathcal{C} \vdash [Loop\ (t1s\ \text{->}\ t2s)\ es] : (ts'' \text{->} ts')$
  using *unlift-b-e assms(2)*
  by *fastforce*
then obtain *ts-c tfn tfm* $\mathcal{C}'$ **where** *t-loop*:$(t1s\ \text{->}\ t2s) = (tfn\ \text{->}\ tfm)$
                                $(ts'' = ts\text{-}c@tfn)$
                                $(ts' = ts\text{-}c@tfm)$
                                $\mathcal{C}' = \mathcal{C}(\!|label := [t1s]\ @\ label\ \mathcal{C}|\!)$
                                $(\mathcal{C}' \vdash es : (tfn\ \text{->}\ tfm))$
  using *b-e-type-loop*[*of* $\mathcal{C}$ *Loop* $(t1s\ \text{->}\ t2s)\ es\ ts''\ ts'$]
  by *fastforce*
obtain *tvs* **where** *tvs-def*:$ts'' = ts\ @\ tvs\ length\ vs = length\ tvs\ \mathcal{S}\text{·}\mathcal{C}' \vdash vs : ([]$
$\text{->}\ tvs)$
  using *e-type-const-list assms(3) ts''-def(1)*
  by *fastforce*
then have *tvs-eq*:$tvs = t1s\ tfn = t1s$
  using *assms(4,5) t-loop(1,2)*
  by *simp-all*
have $\mathcal{S}\text{·}\mathcal{C} \vdash [\$Loop\ (t1s\ \text{->}\ t2s)\ es] : (t1s\ \text{->}\ t2s)$
  using *t-loop b-e-typing.loop e-typing-s-typing.intros(1)*
  by *fastforce*
moreover
have $\mathcal{S}\text{·}\mathcal{C}' \vdash \$*es : (t1s\ \text{->}\ t2s)$
  using *t-loop e-typing-s-typing.intros(1)*
  by *fastforce*
then have $\mathcal{S}\text{·}\mathcal{C}' \vdash vs@(\$*es) : ([]\ \text{->}\ t2s)$
  using *tvs-eq tvs-def(3) e-type-comp-conc*
  by *blast*
ultimately
have $\mathcal{S}\text{·}\mathcal{C} \vdash [Label\ n\ [\$Loop\ (t1s\ \text{->}\ t2s)\ es]\ (vs\ @\ (\$*\ es))] : ([]\ \text{->}\ t2s)$
  using *e-typing-s-typing.intros(7)*[*of* $\mathcal{S}$ $\mathcal{C}$ $[\$Loop\ (t1s\ \text{->}\ t2s)\ es]\ t1s\ t2s\ vs\ @$
$(\$*\ es)]$
        t-loop(4) assms(5)
  by *fastforce*
then show *?thesis*
  using *t-loop e-typing-s-typing.intros(3) tvs-def(1) tvs-eq(1)*
  by *fastforce*
qed

lemma *types-preserved-label-value*:
  assumes $(\!|[Label\ n\ es0\ vs]|\!)\ a \rightsquigarrow (\!|vs|\!)$
       $\mathcal{S}\text{·}\mathcal{C} \vdash [Label\ n\ es0\ vs] : (ts\ \text{->}\ ts')$
       const-list vs
  shows $\mathcal{S}\text{·}\mathcal{C} \vdash vs : (ts\ \text{->}\ ts')$
proof −
  obtain *tls t2s* **where** *t2s-def*:$(ts' = (ts@t2s))$
                       $(\mathcal{S}\text{·}\mathcal{C} \vdash es0 : (tls\ \text{->}\ t2s))$
                       $(\mathcal{S}\text{·}\mathcal{C}(\!|label := [tls]\ @\ (label\ \mathcal{C})|\!) \vdash vs : ([]\ \text{->}\ t2s))$
  using *assms e-type-label*
  by *fastforce*

76

thus *?thesis*
  using *e-type-const-list*[*of vs* $\mathcal{S}$ $\mathcal{C}$(|*label* := [*tls*] @ (*label* $\mathcal{C}$)|) [] *t2s*]
    assms(*3*) *e-typing-s-typing.intros*(*3*)
  by *fastforce*
qed

lemma *typeof-br-if*:
  assumes $\mathcal{S}$·$\mathcal{C}$ ⊢ [$\$C$ *ConstInt32 sec n*, $\$Br$-*if i*] : (*ts* -> *ts'*)
  shows *sec* = *Public*
proof −
  have $\mathcal{C}$ ⊢ [*C ConstInt32 sec n*, *Br-if i*] : (*ts* -> *ts'*)
    using *unlift-b-e assms*(*1*)
    by *fastforce*
  then obtain *ts''* **where** *ts''-def*:$\mathcal{C}$ ⊢ [*C ConstInt32 sec n*] : (*ts* -> *ts''*) $\mathcal{C}$ ⊢ [*Br-if i*] : (*ts''* -> *ts'*)
  using *b-e-type-comp*[*of* - [*C ConstInt32 sec n*] *Br-if i*]
  by *fastforce*
  thus *?thesis*
    using *b-e-type-br-if*[*of* $\mathcal{C}$ *Br-if i ts'' ts' i*] *b-e-type-value*[*OF ts''-def*(*1*)]
    unfolding *typeof-def*
    by *fastforce*
qed

lemma *types-preserved-br-if*:
  assumes (|[$\$C$ *ConstInt32 sec n*, $\$Br$-*if i*]|) $a \rightsquigarrow$ (|*e*|)
    $\mathcal{S}$·$\mathcal{C}$ ⊢ [$\$C$ *ConstInt32 sec n*, $\$Br$-*if i*] : (*ts* -> *ts'*)
    e = [$\$Br$ *i*] ∨ *e* = []
  shows $\mathcal{S}$·$\mathcal{C}$ ⊢ *e* : (*ts* -> *ts'*)
proof −
  have $\mathcal{C}$ ⊢ [*C ConstInt32 sec n*, *Br-if i*] : (*ts* -> *ts'*)
    using *unlift-b-e assms*(*2*)
    by *fastforce*
  then obtain *ts''* **where** *ts''-def*:$\mathcal{C}$ ⊢ [*C ConstInt32 sec n*] : (*ts* -> *ts''*) $\mathcal{C}$ ⊢ [*Br-if i*] : (*ts''* -> *ts'*)
  using *b-e-type-comp*[*of* - [*C ConstInt32 sec n*] *Br-if i*]
  by *fastforce*
  then obtain *ts-c ts-b* **where** *ts-bc-def*:*i* < *length*(*label* $\mathcal{C}$)
                                 ts'' = *ts-c* @ *ts-b* @ [(*T-i32 Public*)]
                                 ts' = *ts-c* @ *ts-b*
                                 (*label* $\mathcal{C}$)!*i* = *ts-b*
    using *b-e-type-br-if*[*of* $\mathcal{C}$ *Br-if i ts'' ts' i*]
    by *fastforce*
  hence *ts-def*:*ts* = *ts-c* @ *ts-b*
    using *ts''-def*(*1*) *b-e-type-value*
    by *fastforce*
  show *?thesis*
    using *assms*(*3*)
  proof (*rule disjE*)
    assume *e* = [$\$Br$ *i*]

77

    thus *?thesis*
      using *ts-def e-typing-s-typing.intros(1) b-e-typing.br ts-bc-def*
      by *fastforce*
  next
   assume $e = []$
   thus *?thesis*
    using *ts-def b-e-type-empty ts-bc-def(3)*
    e-typing-s-typing.intros(1)[of - [] (ts -> ts$'$)]
    by *fastforce*
  qed
qed

lemma *typeof-br-table*:
  assumes $\mathcal{S} \cdot \mathcal{C} \vdash [\$C\ ConstInt32\ sec\ c, \$Br\text{-}table\ is\ i] : (ts \text{ -> } ts')$
  shows $sec = Public$
proof −
  have $\mathcal{C} \vdash [C\ ConstInt32\ sec\ c,\ Br\text{-}table\ is\ i] : (ts \text{ -> } ts')$
   using *unlift-b-e assms(1)*
   by *fastforce*
  then obtain ts'' **where** *ts$''$-def*:$\mathcal{C} \vdash [C\ ConstInt32\ sec\ c] : (ts \text{ -> } ts'')\ \mathcal{C} \vdash$
$[Br\text{-}table\ is\ i] : (ts'' \text{ -> } ts')$
   using *b-e-type-comp[of - [C ConstInt32 sec c] Br-table is i]*
   by *fastforce*
  thus *?thesis*
   using *b-e-type-br-table[of $\mathcal{C}$ Br-table is i ts$''$ ts$'$] b-e-type-value*
   unfolding *typeof-def*
   by *fastforce*
qed

lemma *types-preserved-br-table*:
  assumes $(\![[\$C\ ConstInt32\ sec\ c, \$Br\text{-}table\ is\ i]]\!)\ a \rightsquigarrow (\![[\$Br\ i']]\!)$
     $\mathcal{S} \cdot \mathcal{C} \vdash [\$C\ ConstInt32\ sec\ c, \$Br\text{-}table\ is\ i] : (ts \text{ -> } ts')$
     $(i' = (is\ !\ nat\text{-}of\text{-}int\ c) \wedge length\ is > nat\text{-}of\text{-}int\ c) \vee i' = i$
  shows $\mathcal{S} \cdot \mathcal{C} \vdash [\$Br\ i'] : (ts \text{ -> } ts')$
proof −
  have $\mathcal{C} \vdash [C\ ConstInt32\ sec\ c,\ Br\text{-}table\ is\ i] : (ts \text{ -> } ts')$
   using *unlift-b-e assms(2)*
   by *fastforce*
  then obtain ts'' **where** *ts$''$-def*:$\mathcal{C} \vdash [C\ ConstInt32\ sec\ c] : (ts \text{ -> } ts'')\ \mathcal{C} \vdash$
$[Br\text{-}table\ is\ i] : (ts'' \text{ -> } ts')$
   using *b-e-type-comp[of - [C ConstInt32 sec c] Br-table is i]*
   by *fastforce*
  then obtain *ts-l ts-c* **where** *ts-c-def*:$list\text{-}all\ (\lambda i.\ i < length(label\ \mathcal{C}) \wedge (label\ \mathcal{C})!i$
$= ts\text{-}l)\ (is@[i])$
$$ts'' = ts\text{-}c\ @\ ts\text{-}l@[(T\text{-}i32\ Public)]$$
   using *b-e-type-br-table[of $\mathcal{C}$ Br-table is i ts$''$ ts$'$]*
   by *fastforce*
  hence *ts-def*:$ts = ts\text{-}c\ @\ ts\text{-}l$
   using *ts$''$-def(1) b-e-type-value*

78

by *fastforce*
have $\mathcal{C} \vdash [Br \; i'] : (ts \; \text{->} \; ts')$
  using *assms(3) ts-c-def(1,2) b-e-typing.br[of i' $\mathcal{C}$ ts-l ts-c ts'] ts-def*
  unfolding *list-all-length*
  by (*fastforce simp add*: *less-Suc-eq nth-append*)
thus *?thesis*
  using *e-typing-s-typing.intros(1)*
  by *fastforce*
qed

lemma *types-preserved-local-const*:
  assumes $(\![Local \; n \; i \; vs \; es]\!) \; a \rightsquigarrow (\!|es|\!)$
      $\mathcal{S} \cdot \mathcal{C} \vdash [Local \; n \; i \; vs \; es] : (ts \; \text{->} \; ts')$
      const-list es
  shows $\mathcal{S} \cdot \mathcal{C} \vdash es$: $(ts \; \text{->} \; ts')$
proof $-$
  obtain *tls* **where** $(\mathcal{S} \cdot ((s\text{-}inst \; \mathcal{S})!i)(\![trust\text{-}t := (trust\text{-}t \; \mathcal{C}), \; local := (local \; ((s\text{-}inst$
$\mathcal{S})!i)) \; @ \; (map \; typeof \; vs), \; return := Some \; tls|\!) \vdash es : ([\,] \; \text{->} \; tls))$
         $ts' = ts \; @ \; tls$
  using *e-type-local[OF assms(2)]*
  by *blast+*
  moreover
  then have $\mathcal{S} \cdot \mathcal{C} \vdash es : ([\,] \; \text{->} \; tls)$
    using *assms(3) e-type-const-list*
    by *fastforce*
  ultimately
  show *?thesis*
    using *e-typing-s-typing.intros(3)*
    by *fastforce*
qed

lemma *typing-map-typeof*:
  assumes $ves = \$\$ * \; vs$
      $\mathcal{S} \cdot \mathcal{C} \vdash ves : ([\,] \; \text{->} \; tvs)$
  shows $tvs = map \; typeof \; vs$
  using *assms*
proof (*induction ves arbitrary*: *vs tvs rule*: *List.rev-induct*)
  case *Nil*
  hence $\mathcal{C} \vdash [\,] : ([\,] \; \text{->} \; tvs)$
    using *unlift-b-e*
    by *auto*
  thus *?case*
    using *Nil*
    by *auto*
next
  case (*snoc a ves*)
  obtain $vs' \; v'$ **where** $vs'\text{-}def$: $ves \; @ \; [a] = \$\$ * \; (vs' @ [v']) \; vs = vs' @ [v']$
    using *snoc(2)*
    by (*metis Nil-is-map-conv append-is-Nil-conv list.distinct(1) rev-exhaust*)

obtain *tvs'* **where** *tvs'-def*:$\mathcal{S} \cdot \mathcal{C} \vdash ves$: ([] -> *tvs'*) $\mathcal{S} \cdot \mathcal{C} \vdash [a]$: (*tvs'* -> *tvs*)
  using *snoc(3) e-type-comp*
  by *fastforce*
hence *tvs'* = *map typeof vs'*
  using *snoc(1) vs'-def*
  by *fastforce*
moreover
have *is-const a*
  using *vs'-def*
  unfolding *is-const-def*
  by *auto*
then obtain *t* **where** *t-def*:*tvs* = *tvs'* @ [*t*] $\mathcal{S} \cdot \mathcal{C} \vdash [a]$: ([] -> [*t*])
  using *tvs'-def(2) e-type-const*[*of a* $\mathcal{S}$ $\mathcal{C}$ *tvs'* *tvs*]
  by *fastforce*
have *a* = \$ *C v'*
  using *vs'-def(1)*
  by *auto*
hence *t* = *typeof v'*
  using *t-def unlift-b-e*[*of* $\mathcal{S}$ $\mathcal{C}$ [*C v'*] ([] -> [*t*])] *b-e-type-value*[*of* $\mathcal{C}$ $\mathcal{C}$ *v'* [] [*t*] *v'*]
  by *fastforce*
ultimately
show *?case*
  using *vs'-def t-def*
  by *simp*
qed

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 −
  obtain *t1s t2s* **where** *tf-def*:*tf* = (*t1s* -> *t2s*)
    using *tf.exhaust* **by** *blast*
  obtain *ts''* **where** *ts''-def*:$\mathcal{C} \vdash [C ConstInt32 sec c]$: (*ts* -> *ts''*)
                    $\mathcal{C} \vdash [Call-indirect j]$: (*ts''* -> *ts'*)
    using *e-type-comp*[*of* $\mathcal{S}$ $\mathcal{C}$ [\$*C ConstInt32 sec c*] \$*Call-indirect j ts ts'*]
        assms(1)
        unlift-b-e[*of* $\mathcal{S}$ $\mathcal{C}$ [*C ConstInt32 sec c*]]
        unlift-b-e[*of* $\mathcal{S}$ $\mathcal{C}$ [*Call-indirect j*]]
    by *fastforce*
  hence *ts''* = *ts*@[(*T-i32 sec*)]
    using *b-e-type-value*

80

    unfolding *typeof-def*
    by *fastforce*
  moreover
  have $i' < length$ (*s-inst* $\mathcal{S}$)
    using *assms(5,6)* *store-typing-imp-inst-length-eq*
    by *fastforce*
  hence *stypes-eq:types-t* (*s-inst* $\mathcal{S}$! i') = *types* (*inst s* ! i')
   using *store-typing-imp-inst-typing*[*OF assms(5)*] *store-typing-imp-inst-length-eq*[*OF assms(5)*]
    unfolding *inst-typing.simps*
    by *fastforce*
  obtain $ts''a$ **where** $ts''a$-*def:trust-compat* (*trust-t* $\mathcal{C}$) tr'
                  $j < length$ (*types-t* $\mathcal{C}$)
                  $ts'' = ts''a$ @ $t1s$ @ $[(T\text{-}i32\ Public)]$
                  $ts' = ts''a$ @ $t2s$
                  types-t $\mathcal{C}$! $j = (tr',(t1s \text{ -> } t2s))$
    using *b-e-type-call-indirect*[*OF ts''-def(2), of j*] *tf-def assms(3,7) stypes-eq*
    unfolding *stypes-def*
    by *fastforce*
  moreover
  obtain tf' **where** tf'-*def:cl-typing* $\mathcal{S}$ *cl* tf'
    using *assms(2,5,6)* *stab-typed-some-imp-cl-typed*
    by *blast*
  hence *cl-typing* $\mathcal{S}$ *cl* (tr',tf)
    using *assms(4)*
    unfolding *cl-typing.simps cl-type-def*
    by *auto*
  hence $\mathcal{S}\cdot\mathcal{C} \vdash [Callcl\ cl] : tf$
    using *e-typing-s-typing.intros(6) assms(6,7) ts''a-def(1)*
    by *fastforce*
  ultimately
  show $\mathcal{S}\cdot\mathcal{C} \vdash [Callcl\ cl] : (ts \text{ -> } ts')$
      $sec = Public$
    using *tf-def e-typing-s-typing.intros(3)*
    by *auto*
qed

lemma *types-preserved-call-indirect-None*:
  assumes $\mathcal{S}\cdot\mathcal{C} \vdash [\$C\ ConstInt32\ sec\ c, \$Call\text{-}indirect\ j] : (ts \text{ -> } ts')$
  shows $\mathcal{S}\cdot\mathcal{C} \vdash [Trap] : (ts \text{ -> } ts')$
      $sec = Public$
proof −
  show $\mathcal{S}\cdot\mathcal{C} \vdash [Trap] : (ts \text{ -> } ts')$
    using *e-typing-s-typing.intros(4)*
    by *blast*
  obtain ts'' **where** ts''-*def:*$\mathcal{C} \vdash [C\ ConstInt32\ sec\ c] : (ts \text{ -> } ts'')$
                 $\mathcal{C} \vdash [Call\text{-}indirect\ j] : (ts'' \text{ -> } ts')$
    using *e-type-comp*[*of* $\mathcal{S}$ $\mathcal{C}$ [$\$C\ ConstInt32\ sec\ c$] $\$Call\text{-}indirect\ j\ ts\ ts'$]
        assms(1)

> > *unlift-b-e*[*of S C* [*C ConstInt32 sec c*]]
> > *unlift-b-e*[*of S C* [*Call-indirect j*]]
> > **by** *fastforce*
> **hence** $ts'' = ts@[(T\text{-}i32\ sec)]$
> > **using** *b-e-type-value*
> > **unfolding** *typeof-def*
> > **by** *fastforce*
> **thus** $sec = Public$
> > **using** *b-e-type-call-indirect*[*OF ts''-def(2)*]
> > **by** *fastforce*
> **qed**

lemma *types-preserved-callcl-native*:
> **assumes** $S{\cdot}C \vdash ves\ @\ [Callcl\ cl] : (ts \rightarrow ts')$
> > $cl = Func\text{-}native\ i\ (tr, (t1s \rightarrow t2s))\ tfs\ es$
> > $ves = \$\$*\ vs$
> > $length\ vs = n$
> > $length\ tfs = k$
> > $length\ t1s = n$
> > $length\ t2s = m$
> > $n\text{-}zeros\ tfs = zs$
> > $store\text{-}typing\ s\ S$
> **shows** $S{\cdot}C \vdash [Local\ m\ i\ (vs\ @\ zs)\ [\$Block\ ([] \rightarrow t2s)\ es]] : (ts \rightarrow ts')$
> **proof** −
> **obtain** ts'' **where** *ts''-def*:$S{\cdot}C \vdash ves : (ts \rightarrow ts'')\ S{\cdot}C \vdash [Callcl\ cl] : (ts'' \rightarrow ts')$
> > **using** *assms(1) e-type-comp*
> > **by** *fastforce*
> **have** *ves-c*:*const-list ves*
> > **using** *is-const-list*[*OF assms(3)*]
> > **by** *simp*
> **then obtain** tvs **where** *tvs-def*:$ts'' = ts\ @\ tvs$
> > > $length\ t1s = length\ tvs$
> > > $S{\cdot}C \vdash ves : ([] \rightarrow tvs)$
> > **using** *ts''-def(1) e-type-const-list*[*of ves S C ts ts''*] *assms*
> > **by** *fastforce*
> **obtain** $ts\text{-}c\ C'$ **where** *ts-c-def*:*trust-compat* $(trust\text{-}t\ C)\ tr$
> > > $(ts'' = ts\text{-}c\ @\ t1s)$
> > > $(ts' = ts\text{-}c\ @\ t2s)$
> > > $i < length\ (s\text{-}inst\ S)$
> > > $C' = ((s\text{-}inst\ S)!i)$
> > > $(C'\!(\!trust\text{-}t := tr, local := (local\ C')\ @\ t1s\ @\ tfs, label :=$
> ([t2s] @ (label C')), return := Some t2s) $\vdash es : ([] \rightarrow t2s))$
> > **using** *e-type-callcl-native*[*OF ts''-def(2) assms(2)*]
> > **by** *fastforce*
> **have** *inst-typing S (inst s ! i) (s-inst S ! i)*
> > **using** *store-typing-imp-inst-length-eq*[*OF assms(9)*] *store-typing-imp-inst-typing*[*OF*
> *assms(9)*]
> > > *ts-c-def(4)*
> > **by** *simp*

obtain $\mathcal{C}''$ **where** *c''-def*:$\mathcal{C}'' = \mathcal{C}'(\!|$*trust-t* := *tr*, *local* := (*local* $\mathcal{C}'$) @ *t1s* @ *tfs*, *return* := *Some t2s*$|\!)$
   by *blast*
  hence $\mathcal{C}''(\!|$*label* := ([*t2s*] @ (*label* $\mathcal{C}''$))$|\!)$ $= \mathcal{C}'(\!|$*trust-t* := *tr*, *local* := (*local* $\mathcal{C}'$) @ *t1s* @ *tfs*, *label* := ([*t2s*] @ (*label* $\mathcal{C}'$)), *return* := *Some t2s*$|\!)$
   by *fastforce*
  hence $\mathcal{S}\cdot\mathcal{C}'' \vdash$ [$Block ([] -> t2s) es] : ([] -> t2s)
   using *ts-c-def b-e-typing.block*[*of* ([] -> t2s) [] t2s - es] *e-typing-s-typing.intros(1)*[*of* - [*Block* ([] -> t2s) es]]
   by *fastforce*
  moreover
  have *t-eqs*:*ts* = *ts-c t1s* = *tvs*
   using *tvs-def(1,2) ts-c-def(2)*
   by *simp-all*
  have *1*:*tfs* = *map typeof zs*
   using *n-zeros-typeof assms(8)*
   by *simp*
  have *t1s* = *map typeof vs*
   using *typing-map-typeof assms(3) tvs-def t-eqs*
   by *fastforce*
  hence (*t1s* @ *tfs*) = *map typeof* (*vs* @ *zs*)
   using *1*
   by *simp*
  ultimately
  have $\mathcal{S}\cdot tr\cdot Some\ t2s \Vdash$-*i* (*vs* @ *zs*);([$Block ([] -> t2s) es]) : *t2s*
   using *e-typing-s-typing.intros(8) ts-c-def c''-def*
   by *fastforce*
  thus *?thesis*
   using *e-typing-s-typing.intros(3,5) ts-c-def t-eqs(1) assms(2,7)*
      e-typing-s-typing-trust-compat(2)
   by *fastforce*
qed

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 −
  obtain *ts''* **where** *ts''-def*:$\mathcal{S}\cdot\mathcal{C} \vdash$ *ves* : (*ts* -> *ts''*) $\mathcal{S}\cdot\mathcal{C} \vdash$ [*Callcl cl*] : (*ts''* -> *ts'*)
  using *assms(1) e-type-comp*
  by *fastforce*
  have *ves-c*:*const-list ves*
   using *is-const-list*[*OF assms(3)*]

   by *simp*
  then obtain *tvs* **where** *tvs-def*:*ts″ = ts @ tvs*
                           length t1s = length tvs
                           $\mathcal{S}{\cdot}\mathcal{C} \vdash ves : ([] \rightarrow tvs)$
    using *ts″-def(1) e-type-const-list*[*of ves $\mathcal{S}$ $\mathcal{C}$ ts ts″*] *assms*
    by *fastforce*
  hence *ts″ = ts @ t1s*
     ts′ = ts @ t2s
    using *e-type-callcl-host*[*OF ts″-def(2) assms(2)*]
    by *auto*
  moreover
  hence *list-all2 types-agree t1s vcs*
    using *e-typing-imp-list-types-agree*[**where** *?ts′ = []*] *assms(3) tvs-def(1,3)*
    by *fastforce*
  hence $\mathcal{S}{\cdot}\mathcal{C} \vdash$ *$$\ast vcs′ : ([] \rightarrow t2s)*
    using *list-types-agree-imp-e-typing host-apply-respect-type*[*OF - assms(7)*]
    by *fastforce*
  ultimately
  show *?thesis*
    using *e-typing-s-typing.intros(3)*
    by *fastforce*
qed

lemma *types-imp-concat*:
  assumes $\mathcal{S}{\cdot}\mathcal{C} \vdash es @ [e] @ es′ : (ts \rightarrow ts′)$
         $\bigwedge tes\ tes′.\ ((\mathcal{S}{\cdot}\mathcal{C} \vdash [e] : (tes \rightarrow tes′)) \Longrightarrow (\mathcal{S}{\cdot}\mathcal{C} \vdash [e′] : (tes \rightarrow tes′)))$
  shows $\mathcal{S}{\cdot}\mathcal{C} \vdash es @ [e′] @ es′ : (ts \rightarrow ts′)$
proof −
  obtain *ts″* **where** $\mathcal{S}{\cdot}\mathcal{C} \vdash es : (ts \rightarrow ts″)$
            $\mathcal{S}{\cdot}\mathcal{C} \vdash [e] @ es′ : (ts″ \rightarrow ts′)$
    using *e-type-comp-conc1*[*of - - es [e] @ es′*] *assms(1)*
    by *fastforce*
  moreover
  then obtain *ts‴* **where** $\mathcal{S}{\cdot}\mathcal{C} \vdash [e] : (ts″ \rightarrow ts‴)$ $\mathcal{S}{\cdot}\mathcal{C} \vdash es′ : (ts‴ \rightarrow ts′)$
    using *e-type-comp-conc1*[*of - - [e] es′ ts″ ts′*] *assms*
    by *fastforce*
  ultimately
  show *?thesis*
    using *assms(2) e-type-comp-conc*[*of - - es ts ts″ [e′] ts‴*]
                e-type-comp-conc[*of - - es @ [e′] ts ts‴*]
    by *fastforce*
qed

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

   shows $\mathcal{S}\cdot\mathcal{C}' \vdash vs : ([] \to tcs)$
   using *assms*
proof (*induction i arbitrary: ts ts' lholed C LI C'*)
  case *0*
  obtain *vs' es'* **where** $LI = (vs' @ (vs @ [\$Return]) @ es'$
   using *Lfilled.simps[of 0 lholed (vs @ [\$Return]) LI] 0(1)*
   by *fastforce*
  then obtain *ts'' ts'''* **where** $\mathcal{S}\cdot\mathcal{C} \vdash vs' : (ts \to ts'')$
                        $\mathcal{S}\cdot\mathcal{C} \vdash (vs @ [\$Return]) : (ts'' \to ts''')$
                        $\mathcal{S}\cdot\mathcal{C} \vdash es' : (ts''' \to ts')$
   using *e-type-comp-conc2[of $\mathcal{S}$ $\mathcal{C}$ vs' (vs @ [\$Return]) es'] 0(4)*
   by *fastforce*
  then obtain *ts-b* **where** *ts-b-def*:$\mathcal{S}\cdot\mathcal{C} \vdash vs : (ts'' \to ts\text{-}b)$ $\mathcal{S}\cdot\mathcal{C} \vdash [\$Return] : (ts\text{-}b \to ts''')$
   using *e-type-comp-conc1*
   by *fastforce*
  then obtain *ts-c* **where** *ts-c-def*:$ts\text{-}b = ts\text{-}c$ @ tcs (*return* $\mathcal{C}$) = *Some tcs*
   using *0(2) b-e-type-return[of $\mathcal{C}$] unlift-b-e[of $\mathcal{S}$ $\mathcal{C}$ [Return] ts-b $\to$ ts'']*
   by *fastforce*
  obtain *tcs'* **where** $ts\text{-}b = ts''$ @ tcs' *length vs = length tcs'* $\mathcal{S}\cdot\mathcal{C}' \vdash vs : ([] \to tcs')$
   using *ts-b-def(1) e-type-const-list 0(5)*
   by *fastforce*
  thus *?case*
   using *0(3) ts-c-def*
   by *simp*
next
  case (*Suc i*)
  obtain *vs' n l les les' LK* **where** *es-def*:$lholed = (LRec\ vs'\ n\ les\ l\ les')$
                                   Lfilled i l (vs @ [\$Return]) LK
                                   $LI = (vs' @ [Label\ n\ les\ LK] @ les')$
   using *Lfilled.simps[of (Suc i) lholed (vs @ [\$Return]) LI] Suc(2)*
   by *fastforce*
  then obtain *ts'' ts'''* **where** $\mathcal{S}\cdot\mathcal{C} \vdash [Label\ n\ les\ LK] : (ts'' \to ts''')$
   using *e-type-comp-conc2[of $\mathcal{S}$ $\mathcal{C}$ vs' [Label n les LK] les'] Suc(5)*
   by *fastforce*
  then obtain *tls t2s* **where**
     $ts''' = ts''$ @ $t2s$
     length tls = n
     $\mathcal{S}\cdot\mathcal{C} \vdash les : (tls \to t2s)$
     $\mathcal{S}\cdot\mathcal{C}(\!|label := [tls] @ label\ \mathcal{C}|\!) \vdash LK : ([] \to t2s)$
     return $(\mathcal{C}(\!|label := [tls] @ label\ \mathcal{C}|\!))$ = *Some tcs*
   using *e-type-label[of $\mathcal{S}$ $\mathcal{C}$ n les LK ts'' ts'''] Suc(3)*
   by *fastforce*
  then show *?case*
   using *Suc(1)[OF es-def(2) - assms(3) - assms(5)]*
   by *fastforce*
qed

lemma *types-preserved-return*:
  assumes $(\![Local\ n\ i\ vls\ LI]\!)\ a \leadsto (\!|ves|\!)$
        $\mathcal{S}\cdot\mathcal{C} \vdash [Local\ n\ i\ vls\ LI] : (ts \rightarrow ts')$
        const-list ves
        length ves = n
        Lfilled j lholed (ves @ [$Return]) LI
  shows $\mathcal{S}\cdot\mathcal{C} \vdash ves : (ts \rightarrow ts')$
proof −
  obtain *tls* $\mathcal{C}'$ **where** *l-def*:$i < length\ (s\text{-}inst\ \mathcal{S})$
                    $\mathcal{C}' = ((s\text{-}inst\ \mathcal{S})!i)(\!|trust\text{-}t := (trust\text{-}t\ \mathcal{C}), local := (local\ ((s\text{-}inst$
$\mathcal{S})!i)) @ (map\ typeof\ vls), return := Some\ tls|\!)$
                    $\mathcal{S}\cdot\mathcal{C}' \vdash LI : ([] \rightarrow tls)$
                    $ts' = ts\ @\ tls$
                    length tls = n
    using *e-type-local*[*OF assms(2)*]
    by *blast*
  hence $\mathcal{S}\cdot\mathcal{C} \vdash ves : ([] \rightarrow tls)$
    using *type-const-return*[*OF assms(5) - - l-def(3)*] *assms(3−5)*
    by *fastforce*
  thus *?thesis*
    using *e-typing-s-typing.intros(3) l-def(4)*
    by *fastforce*
qed

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
        $\mathcal{S}\cdot\mathcal{C} \vdash LI : (ts \rightarrow ts')$
        const-list vs
  shows $\mathcal{S}\cdot\mathcal{C}' \vdash vs : ([] \rightarrow tcs)$
  using *assms*
proof (*induction i arbitrary*: *k ts ts' lholed* $\mathcal{C}$ *LI* $\mathcal{C}'$)
  case *0*
  obtain *vs' es'* **where** *LI = (vs' @ (vs @ [$Br (0+k)]) @ es')*
    using *Lfilled.simps*[*of 0 lholed (vs @ [$Br (0 + k)]) LI*] *0(1)*
    by *fastforce*
  then obtain *ts'' ts'''* **where** $\mathcal{S}\cdot\mathcal{C} \vdash vs' : (ts \rightarrow ts'')$
                    $\mathcal{S}\cdot\mathcal{C} \vdash (vs\ @\ [$Br\ (0+k)]) : (ts'' \rightarrow ts''')$
                    $\mathcal{S}\cdot\mathcal{C} \vdash es' : (ts''' \rightarrow ts')$
    using *e-type-comp-conc2*[*of* $\mathcal{S}$ $\mathcal{C}$ *vs' (vs @ [$Br (0+k)]) es'*] *0(5)*
    by *fastforce*
  then obtain *ts-b* **where** *ts-b-def*:$\mathcal{S}\cdot\mathcal{C} \vdash vs : (ts'' \rightarrow ts\text{-}b)$ $\mathcal{S}\cdot\mathcal{C} \vdash [$Br\ (0+k)] :$
$(ts\text{-}b \rightarrow ts''')$
    using *e-type-comp-conc1*
    by *fastforce*
  then obtain *ts-c* **where** *ts-c-def*:*ts-b = ts-c @ tcs (label* $\mathcal{C}$)!*k = tcs*
    using *0(3) b-e-type-br*[*of* $\mathcal{C}$ *Br (0 + k)*] *unlift-b-e*[*of* $\mathcal{S}$ $\mathcal{C}$ *[Br (0 + k)] ts-b* ->

ts''']
  by *fastforce*
  obtain tcs' **where** ts-$b = ts'' @ tcs'$ $length\ vs = length\ tcs'$ $\mathcal{S}\cdot\mathcal{C}' \vdash vs : ([]\ ->$
$tcs')$
   using ts-b-$def(1)$ e-$type$-$const$-$list$ $0(6)$
   by *fastforce*
  thus *?case*
   using $0(4)$ ts-c-def
   by *simp*
next
  case $(Suc\ i\ k\ ts\ ts'\ lholed\ \mathcal{C}\ LI)$
  obtain $vs'\ n\ l\ les\ les'\ LK$ **where** es-def:$lholed = (LRec\ vs'\ n\ les\ l\ les')$
                                      $Lfilled\ i\ l\ (vs @ [\$Br\ (i + (Suc\ k))]) \ LK$
                                      $LI = (vs' @ [Label\ n\ les\ LK] @ les')$
   using $Lfilled.simps[of\ (Suc\ i)\ lholed\ (vs @ [\$Br\ ((Suc\ i) + k)])\ LI]$ $Suc(2)$
   by *fastforce*
  then obtain $ts''\ ts'''$ **where** $\mathcal{S}\cdot\mathcal{C} \vdash [Label\ n\ les\ LK] : (ts'' \ ->\ ts''')$
   using e-$type$-$comp$-$conc2[of\ \mathcal{S}\ \mathcal{C}\ vs'\ [Label\ n\ les\ LK]\ les']$ $Suc(6)$
   by *fastforce*
  moreover
  then obtain $lts\ \mathcal{C}''\ ts''''$ **where** $\mathcal{S}\cdot\mathcal{C}'' \vdash LK : ([]\ ->\ ts'''')$ $\mathcal{C}'' = \mathcal{C}(\!|label := [lts]$
$@\ (label\ \mathcal{C})|\!)$
                                 $length\ (label\ \mathcal{C}'') > (Suc\ k)$
                                 $(label\ \mathcal{C}'')!(Suc\ k) = tcs$
   using e-$type$-$label[of\ \mathcal{S}\ \mathcal{C}\ n\ les\ LK\ ts''\ ts''']$ $Suc(3,4)$
   by *fastforce*
  then show *?case*
   using $Suc(1)$ es-$def(2)$ $assms(4,6)$
   by *fastforce*
qed

lemma *types-preserved-br*:
  assumes $(\!|[Label\ n\ es0\ LI]|\!)$ $a \rightsquigarrow (\!|vs @ es0|\!)$
        $\mathcal{S}\cdot\mathcal{C} \vdash [Label\ n\ es0\ LI] : (ts\ ->\ ts')$
        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 $-$
  obtain $tls\ t2s\ \mathcal{C}'$ **where** l-def:$(ts' = (ts@t2s))$
                      $(\mathcal{S}\cdot\mathcal{C} \vdash es0 : (tls\ ->\ t2s))$
                      $\mathcal{C}' = \mathcal{C}(\!|label := [tls] @ (label\ \mathcal{C})|\!)$
                      $length\ (label\ \mathcal{C}') > 0$
                      $(label\ \mathcal{C}')!0 = tls$
                      $length\ tls = n$
                      $(\mathcal{S}\cdot\mathcal{C}(\!|label := [tls] @ (label\ \mathcal{C})|\!) \vdash LI : ([]\ ->\ t2s))$
   using e-$type$-$label[of\ \mathcal{S}\ \mathcal{C}\ n\ es0\ LI\ ts\ ts']$ $assms(2)$
   by *fastforce*
  hence $\mathcal{S}\cdot\mathcal{C} \vdash vs : ([]\ ->\ tls)$

    using *assms(3−5)* *type-const-br*[*of i lholed vs 0 LI C′ tls*]
    by *fastforce*
  thus *?thesis*
    using *l-def(1,2)* *e-type-comp-conc* *e-typing-s-typing.intros(3)*
    by *fastforce*
qed

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 −
  obtain *insts* **where** *inst-typ*:*list-all2* (*inst-typing* $\mathcal{S}$) *insts* (*s-inst* $\mathcal{S}$)
    using *assms(2)*
    unfolding *store-typing.simps*
    by *auto*
  thus *label* ((*s-inst* $\mathcal{S}$)!*i*) = []
    using *assms(1)*
    unfolding *inst-typing.simps List.list-all2-conv-all-nth*
    by *fastforce*
  show *local* ((*s-inst* $\mathcal{S}$)!*i*) = []
    using *assms(1) inst-typ*
    unfolding *inst-typing.simps List.list-all2-conv-all-nth*
    by *fastforce*
qed

lemma *types-preserved-b-e1*:
  assumes $(\!|es|\!)\ a\rightsquigarrow (\!|es'|\!)$
       store-typing s $\mathcal{S}$
       $\mathcal{S}\cdot\mathcal{C} \vdash es : (ts \rightarrow ts')$
  shows $\mathcal{S}\cdot\mathcal{C} \vdash es' : (ts \rightarrow ts')$
  using *assms(1)*
proof (*cases rule*: *reduce-simple.cases*)
  case (*unop-i32 c iop*)
  thus *?thesis*
    using *assms(1,3) types-preserved-unop-testop-cvtop*
    by *simp*
next
  case (*unop-i64 c iop*)
  thus *?thesis*
    using *assms(1, 3) types-preserved-unop-testop-cvtop*
    by *simp*
next
  case (*unop-f32 c fop*)
  thus *?thesis*
    using *assms(1, 3) types-preserved-unop-testop-cvtop*
    by *simp*
next
  case (*unop-f64 c fop*)

thus *?thesis*
  using *assms(1, 3) types-preserved-unop-testop-cvtop*
  by *simp*
next
 case (*binop-i32-Some v iop c1 c2*)
 thus *?thesis*
  using *assms(1, 3) types-preserved-binop-relop*
  by *simp*
next
 case (*binop-i32-None iop c1 c2*)
 thus *?thesis*
  by (*simp add: e-typing-s-typing.intros(4)*)
next
 case (*binop-i64-Some v iop c1 c2*)
 thus *?thesis*
  using *assms(1, 3) types-preserved-binop-relop*
  by *simp*
next
 case (*binop-i64-None iop c1 c2*)
 thus *?thesis*
  by (*simp add: e-typing-s-typing.intros(4)*)
next
 case (*binop-f32-Some v fop c1 c2*)
 thus *?thesis*
  using *assms(1, 3) types-preserved-binop-relop*
  by *simp*
next
 case (*binop-f32-None fop c1 c2*)
 thus *?thesis*
  by (*simp add: e-typing-s-typing.intros(4)*)
next
 case (*binop-f64-Some v fop c1 c2*)
 then show *?thesis*
  using *assms(1, 3) types-preserved-binop-relop*
  by *simp*
next
 case (*binop-f64-None fop c1 c2*)
 then show *?thesis*
  by (*simp add: e-typing-s-typing.intros(4)*)
next
 case (*testop-i32 c testop*)
 then show *?thesis*
  using *assms(1, 3) types-preserved-unop-testop-cvtop*
  by *simp*
next
 case (*testop-i64 c testop*)
 then show *?thesis*
  using *assms(1, 3) types-preserved-unop-testop-cvtop*
  by *simp*

next
　case (*relop-i32 c1 c2 iop*)
　then show *?thesis*
　　using *assms*(*1*, *3*) *types-preserved-binop-relop*
　　by *simp*
next
　case (*relop-i64 c1 c2 iop*)
　then show *?thesis*
　　using *assms*(*1*, *3*) *types-preserved-binop-relop*
　　by *simp*
next
　case (*relop-f32 c1 c2 fop*)
　then show *?thesis*
　　using *assms*(*1*, *3*) *types-preserved-binop-relop*
　　by *simp*
next
　case (*relop-f64 c1 c2 fop*)
　then show *?thesis*
　　using *assms*(*1*, *3*) *types-preserved-binop-relop*
　　by *simp*
next
　case (*convert-Some t1 v v′ t2 sx*)
　then show *?thesis*
　　using *assms*(*1*, *3*) *types-preserved-unop-testop-cvtop*
　　by *simp*
next
　case (*convert-None t1 v t2 sx*)
　then show *?thesis*
　　using *e-typing-s-typing.intros*(*4*)
　　by *simp*
next
　case (*reinterpret t1 v t2*)
　then show *?thesis*
　　using *assms*(*1*, *3*) *types-preserved-unop-testop-cvtop*
　　by *simp*
next
　case (*classify t1 v t2*)
　then show *?thesis*
　　using *assms*(*1*, *3*) *types-preserved-unop-testop-cvtop*
　　by *simp*
next
　case (*declassify t1 v t2*)
　then show *?thesis*
　　using *assms*(*1*, *3*) *types-preserved-unop-testop-cvtop*
　　by *simp*
next
　case *unreachable*
　then show *?thesis*
　　using *e-typing-s-typing.intros*(*4*)

    by *simp*
next
  case *nop*
  then have $\mathcal{C} \vdash [Nop] : (ts \to ts')$
    using *assms(3) unlift-b-e*
    by *simp*
  then show *?thesis*
    using *nop b-e-typing.empty e-typing-s-typing.intros(1,3)*
    apply (*induction* [*Nop*] *ts* -> *ts'* *arbitrary*: *ts ts'*)
     apply *simp-all*
    apply (*metis list.simps(8)*)
    apply *blast*
    done
next
  case (*drop v*)
  then show *?thesis*
    using *assms(1, 3) types-preserved-drop*
    by *simp*
next
  case (*select-false v1 v2*)
  then show *?thesis*
    using *assms(1, 3) types-preserved-select*
    by *simp*
next
  case (*select-true n v1 v2*)
  then show *?thesis*
    using *assms(1, 3) types-preserved-select*
    by *simp*
next
  case (*block vs n t1s t2s m es*)
  then show *?thesis*
    using *assms(1, 3) types-preserved-block*
    by *simp*
next
  case (*loop vs n t1s t2s m es*)
  then show *?thesis*
    using *assms(1, 3) types-preserved-loop*
    by *simp*
next
  case (*if-false tf e1s e2s*)
  then show *?thesis*
    using *assms(1, 3) types-preserved-if*
    by *simp*
next
  case (*if-true n tf e1s e2s*)
  then show *?thesis*
    using *assms(1, 3) types-preserved-if*
    by *simp*
next

case (*label-const ts es*)
  then show *?thesis*
    using *assms*(*1*, *3*) *types-preserved-label-value*
    by *simp*
next
  case (*label-trap ts es*)
  then show *?thesis*
    by (*simp add*: *e-typing-s-typing.intros*(*4*))
next
  case (*br vs n i lholed LI es*)
  then show *?thesis*
    using *assms*(*1*, *3*) *types-preserved-br*
    by *fastforce*
next
  case (*br-if-false n sec i*)
  then show *?thesis*
    using *assms*(*1*, *3*) *types-preserved-br-if*
    by *fastforce*
next
  case (*br-if-true n i*)
  then show *?thesis*
    using *assms*(*1*, *3*) *types-preserved-br-if*
    by *fastforce*
next
  case (*br-table is' c sec i'*)
  then show *?thesis*
    using *assms*(*1*, *3*) *types-preserved-br-table*
    by *fastforce*
next
  case (*br-table-length is' c sec i'*)
  then show *?thesis*
    using *assms*(*1*, *3*) *types-preserved-br-table*
    by *fastforce*
next
  case (*local-const i vs*)
  then show *?thesis*
    using *assms*(*1*, *3*) *types-preserved-local-const*
    by *fastforce*
next
  case (*local-trap i vs*)
  then show *?thesis*
    by (*simp add*: *e-typing-s-typing.intros*(*4*))
next
  case (*return n j lholed es i vls*)
  then show *?thesis*
    using *assms*(*1*, *3*) *types-preserved-return*
    by *fastforce*
next
  case (*tee-local v i*)

then show *?thesis*
    using *assms(1, 3) types-preserved-tee-local*
    by *simp*
next
  case (*trap lholed*)
  then show *?thesis*
    by (*simp add: e-typing-s-typing.intros(4)*)
qed

lemma *types-preserved-b-e*:
  assumes $(\!|es|\!)\ a \rightsquigarrow (\!|es'|\!)$
       store-typing s $\mathcal{S}$
       $\mathcal{S} \cdot tr \cdot None \Vdash \text{-}i\ vs;es : ts$
  shows $\mathcal{S} \cdot tr \cdot None \Vdash \text{-}i\ vs;es' : ts$
proof −
  have $i < (length\ (s\text{-}inst\ \mathcal{S}))$
    using *assms(3) s-typing.cases*
    by *blast*
  moreover
  obtain *tvs $\mathcal{C}$* **where** *defs*: $tvs = map\ typeof\ vs\ \mathcal{C} = ((s\text{-}inst\ \mathcal{S})!i)(\!|trust\text{-}t := tr,$
local := (local ((s-inst $\mathcal{S}$)!i) @ tvs), return := None|$\!$) $\mathcal{S} \cdot \mathcal{C} \vdash es : ([] \rightarrow ts)$
    using *assms(3)*
    unfolding *s-typing.simps*
    by *blast*
  have $\mathcal{S} \cdot \mathcal{C} \vdash es' : ([] \rightarrow ts)$
    using *assms(1,2) defs(3) types-preserved-b-e1*
    by *simp*
  ultimately show *?thesis*
    using *defs*
    unfolding *s-typing.simps*
    by *auto*
qed

lemma *types-preserved-store*:
  assumes $\mathcal{S} \cdot \mathcal{C} \vdash [\$C\ ConstInt32\ sec\ k,\ \$C\ v,\ \$Store\ t\ tp\ a\ off] : (ts \rightarrow ts')$
  shows $\mathcal{S} \cdot \mathcal{C} \vdash [] : (ts \rightarrow ts')$
      sec = Public
      types-agree t v
proof −
  obtain $ts''\ ts'''$ **where** *ts-def*:$\mathcal{S} \cdot \mathcal{C} \vdash [\$C\ ConstInt32\ sec\ k] : (ts \rightarrow ts'')$
                     $\mathcal{S} \cdot \mathcal{C} \vdash [\$C\ v] : (ts'' \rightarrow ts''')$
                     $\mathcal{S} \cdot \mathcal{C} \vdash [\$Store\ t\ tp\ a\ off] : (ts''' \rightarrow ts')$
    using *assms e-type-comp-conc2[of $\mathcal{S}$ $\mathcal{C}$ [\$C ConstInt32 sec k] [\$C v] [\$Store t*
tp a off]]
    by *fastforce*
  then have $ts'' = ts@[(T\text{-}i32\ sec)]$
    using *b-e-type-value[of $\mathcal{C}$ $\mathcal{C}$ ConstInt32 sec k ts ts'']*
      unlift-b-e[of $\mathcal{S}$ $\mathcal{C}$ [C (ConstInt32 sec k)] (ts \rightarrow ts'')]
    unfolding *typeof-def*

by *fastforce*
  hence *ts''' = ts@[(T-i32 sec), (typeof v)]*
    using *ts-def(2) b-e-type-value[of C C v ts'' ts''']*
        unlift-b-e[of S C [C v] (ts'' -> ts''')]
    by *fastforce*
  hence *ts = ts' sec = Public types-agree t v*
    using *ts-def(3) b-e-type-store[of C Store t tp a off ts''' ts']*
        unlift-b-e[of S C [Store t tp a off] (ts''' -> ts')]
    unfolding *types-agree-def*
    by *fastforce+*
  thus *S·C ⊢ [] : (ts -> ts') sec = Public types-agree t v*
    using *b-e-type-empty[of C ts ts'] e-typing-s-typing.intros(1)*
    by *fastforce+*
qed

lemma *types-preserved-current-memory*:
  assumes *S·C ⊢ [$Current-memory] : (ts -> ts')*
  shows *S·C ⊢ [$C ConstInt32 Public c] : (ts -> ts')*
proof −
  have *ts' = ts@[(T-i32 Public)]*
    using *assms b-e-type-current-memory unlift-b-e[of S C [Current-memory]]*
    by *fastforce*
  thus *?thesis*
    using *b-e-typing.const[of C ConstInt32 Public c] e-typing-s-typing.intros(1,3)*
    unfolding *typeof-def*
    by *fastforce*
qed

lemma *types-preserved-grow-memory*:
  assumes *S·C ⊢ [$C ConstInt32 sec c, $Grow-memory] : (ts -> ts')*
  shows *S·C ⊢ [$C ConstInt32 sec c'] : (ts -> ts')*
      sec = Public
proof −
  obtain *ts''* **where** *ts''-def:S·C ⊢ [$C ConstInt32 sec c] : (ts -> ts'')*
                      S·C ⊢ [$Grow-memory] : (ts'' -> ts')
    using *e-type-comp assms*
    by *(metis append-butlast-last-id butlast.simps(2) last.simps list.distinct(1))*
  have *ts'' = ts@[(T-i32 sec)]*
    using *b-e-type-value[of C C ConstInt32 sec c ts ts'']*
        unlift-b-e[of S C [C ConstInt32 sec c]] ts''-def(1)
    unfolding *typeof-def*
    by *fastforce*
  moreover
  hence *ts'' = ts'* **and** *sec = Public*
   using *ts''-def b-e-type-grow-memory[of - - ts'' ts'] unlift-b-e[of S C [Grow-memory]]*
    by *fastforce+*
  ultimately
  show *S·C ⊢ [$C ConstInt32 sec c'] : (ts -> ts') sec = Public*
    using *e-typing-s-typing.intros(1,3)*

b-e-typing.const[*of C ConstInt32 sec c'*]
    unfolding *typeof-def*
    by *fastforce+*
qed

lemma *types-preserved-set-global*:
  assumes $\mathcal{S} \cdot \mathcal{C} \vdash [\$C\ v,\ \$Set\text{-}global\ j] : (ts \rightarrow ts')$
  shows $\mathcal{S} \cdot \mathcal{C} \vdash [] : (ts \rightarrow ts')$
       tg-t $(global\ \mathcal{C}\ !\ j) = typeof\ v$
proof −
  obtain ts'' **where** $ts''\text{-}def{:}\mathcal{S} \cdot \mathcal{C} \vdash [\$C\ v] : (ts \rightarrow ts'')$
                           $\mathcal{S} \cdot \mathcal{C} \vdash [\$Set\text{-}global\ j] : (ts'' \rightarrow ts')$
    using *e-type-comp assms*
    by (*metis append-butlast-last-id butlast.simps(2) last.simps list.distinct(1)*)
  hence $ts'' = ts@[typeof\ v]$
    using *b-e-type-value unlift-b-e*[*of S C [C v]*]
    by *fastforce*
  hence $ts = ts'\ tg\text{-}t\ (global\ \mathcal{C}\ !\ j) = typeof\ v$
    using *b-e-type-set-global ts''-def(2) unlift-b-e*[*of S C [Set-global j]*]
    by *fastforce+*
  thus $\mathcal{S} \cdot \mathcal{C} \vdash [] : (ts \rightarrow ts')\ tg\text{-}t\ (global\ \mathcal{C}\ !\ j) = typeof\ v$
    using *b-e-type-empty*[*of C ts ts'*] *e-typing-s-typing.intros(1)*
    by *fastforce+*
qed

lemma *types-preserved-load*:
  assumes $\mathcal{S} \cdot \mathcal{C} \vdash [\$C\ ConstInt32\ sec\ k,\ \$Load\ t\ tp\ a\ off] : (ts \rightarrow ts')$
       typeof $v = t$
  shows $\mathcal{S} \cdot \mathcal{C} \vdash [\$C\ v] : (ts \rightarrow ts')$
       sec = Public
proof −
  obtain ts'' **where** $ts''\text{-}def{:}\mathcal{S} \cdot \mathcal{C} \vdash [\$C\ ConstInt32\ sec\ k] : (ts \rightarrow ts'')$
                           $\mathcal{S} \cdot \mathcal{C} \vdash [\$Load\ t\ tp\ a\ off] : (ts'' \rightarrow ts')$
    using *e-type-comp assms*
    by (*metis append-butlast-last-id butlast.simps(2) last.simps list.distinct(1)*)
  hence $ts'' = ts@[(T\text{-}i32\ sec)]$
    using *b-e-type-value unlift-b-e*[*of S C [C ConstInt32 sec k]*]
    unfolding *typeof-def*
    by *fastforce*
  hence *ts-def:sec = Public* $ts' = ts@[t]$ *load-store-t-bounds a (option-projl tp) t*
    using *ts''-def(2) b-e-type-load unlift-b-e*[*of S C [Load t tp a off]*]
     by (*metis last-snoc t.inject(1) to-e-list-1, metis to-e-list-1 append1-eq-conv,*
metis to-e-list-1)
  moreover
  hence $\mathcal{C} \vdash [C\ v] : (ts \rightarrow ts@[t])$
    using *assms(2) b-e-typing.const b-e-typing.weakening*
    by *fastforce*
  ultimately
  show $\mathcal{S} \cdot \mathcal{C} \vdash [\$C\ v] : (ts \rightarrow ts')\ sec = Public$

95

    using *e-typing-s-typing.intros*(1)
    by *fastforce*+
qed

lemma *types-preserved-get-local*:
  assumes $\mathcal{S} \cdot \mathcal{C} \vdash [\$Get\text{-}local\ i] : (ts \text{ -> } ts')$
      length vi = i
      $(local\ \mathcal{C}) = map\ typeof\ (vi\ @\ [v]\ @\ vs)$
  shows $\mathcal{S} \cdot \mathcal{C} \vdash [\$C\ v] : (ts \text{ -> } ts')$
proof $-$
  have $(local\ \mathcal{C})!i = typeof\ v$
    using *assms*(2,3)
   by (*metis* (*no-types*, *hide-lams*) *append-Cons length-map list.simps*(9) *map-append nth-append-length*)
  hence $ts' = ts@[typeof\ v]$
    using *assms*(1) *unlift-b-e*[*of* $\mathcal{S}$ $\mathcal{C}$ [$Get\text{-}local\ i$]] *b-e-type-get-local*
    by *fastforce*
  thus *?thesis*
    using *b-e-typing.const e-typing-s-typing.intros*(1,3)
    by *fastforce*
qed

lemma *types-preserved-set-local*:
  assumes $\mathcal{S} \cdot \mathcal{C} \vdash [\$C\ v', \$Set\text{-}local\ i] : (ts \text{ -> } ts')$
      length vi = i
      $(local\ \mathcal{C}) = map\ typeof\ (vi\ @\ [v]\ @\ vs)$
  shows $(\mathcal{S} \cdot \mathcal{C} \vdash [] : (ts \text{ -> } ts')) \wedge map\ typeof\ (vi\ @\ [v]\ @\ vs) = map\ typeof\ (vi\ @$ $[v'] @ vs)$
proof $-$
  have *v-type*:$(local\ \mathcal{C})!i = typeof\ v$
    using *assms*(2,3)
   by (*metis* (*no-types*, *hide-lams*) *append-Cons length-map list.simps*(9) *map-append nth-append-length*)
  obtain ts'' **where** *ts''-def*:$\mathcal{S} \cdot \mathcal{C} \vdash [\$C\ v'] : (ts \text{ -> } ts'')$
                  $\mathcal{S} \cdot \mathcal{C} \vdash [\$Set\text{-}local\ i] : (ts'' \text{ -> } ts')$
    using *e-type-comp assms*
    by (*metis append-butlast-last-id butlast.simps*(2) *last.simps list.distinct*(1))
  hence $ts'' = ts@[typeof\ v']$
    using *b-e-type-value unlift-b-e*[*of* $\mathcal{S}$ $\mathcal{C}$ [$C\ v'$]]
    by *fastforce*
  hence $typeof\ v = typeof\ v'\ ts' = ts$
    using *v-type b-e-type-set-local*[*of* $\mathcal{C}$ *Set-local i* ts'' ts'] *ts''-def*(2) *unlift-b-e*[*of* $\mathcal{S}$ $\mathcal{C}$ [$Set\text{-}local\ i$]]
    by *fastforce*+
  thus *?thesis*
    using *b-e-type-empty*[*of* $\mathcal{C}$ *ts* ts'] *e-typing-s-typing.intros*(1)
    by *fastforce*
qed

lemma *types-preserved-get-global*:
  assumes *typeof* (*sglob-val s i j*) = *tg-t* (*global C* ! *j*)
        $\mathcal{S}\cdot\mathcal{C} \vdash [\$ \textit{Get-global } j] : (ts \rightarrow ts')$
  shows $\mathcal{S}\cdot\mathcal{C} \vdash [\$C \textit{ sglob-val s i j}] : (ts \rightarrow ts')$
proof −
  have $ts' = ts@[\textit{tg-t } (\textit{global } C \text{ ! } j)]$
    using *b-e-type-get-global assms(2) unlift-b-e*[*of* - - [*Get-global j*]]
    by *fastforce*
  thus *?thesis*
   using *b-e-typing.const*[*of C sglob-val s i j*] *assms(1) e-typing-s-typing.intros(1,3)*
    by *fastforce*
qed

lemma *lholed-same-type*:
  assumes *Lfilled k lholed es les*
        Lfilled k lholed es' les'
        $\mathcal{S}\cdot\mathcal{C} \vdash les : (ts \rightarrow ts')$
        $\bigwedge \textit{arb-labs ts ts'}.$
          $\mathcal{S}\cdot(\mathcal{C}(\!|label := \textit{arb-labs}@(\textit{label } \mathcal{C})|\!)) \vdash es : (ts \rightarrow ts')$
            $\Longrightarrow \mathcal{S}\cdot(\mathcal{C}(\!|label := \textit{arb-labs}@(\textit{label } \mathcal{C})|\!)) \vdash es' : (ts \rightarrow ts')$
  shows $(\mathcal{S}\cdot\mathcal{C} \vdash les' : (ts \rightarrow ts'))$
  using *assms*
proof (*induction arbitrary*: *ts ts' es' C les' rule*: *Lfilled.induct*)
  case (*L0 vs lholed es' es ts ts' es''*)
  obtain $ts''\ ts'''$ **where** $\mathcal{S}\cdot\mathcal{C} \vdash vs : (ts \rightarrow ts'')$
                      $\mathcal{S}\cdot\mathcal{C} \vdash es : (ts'' \rightarrow ts''')$
                      $\mathcal{S}\cdot\mathcal{C} \vdash es' : (ts''' \rightarrow ts')$
    using *e-type-comp-conc2 L0(4)*
    by *blast*
  moreover
  hence $(\mathcal{S}\cdot\mathcal{C} \vdash es'' : (ts'' \rightarrow ts'''))$
    using *L0(5)*[*of* [] $ts''\ ts'''$]
    by *fastforce*
  ultimately
  have $(\mathcal{S}\cdot\mathcal{C} \vdash vs @ es'' @ es' : (ts \rightarrow ts'))$
    using *e-type-comp-conc*
    by *fastforce*
  thus *?case*
    using *L0(2,3) Lfilled.simps*[*of 0 lholed es'' les'*]
    by *fastforce*
next
  case (*LN vs lholed n es' l es'' k es lfilledk t1s t2s es''' C les'*)
  obtain *lfilledk'* **where** *l'-def*:*Lfilled k l es''' lfilledk' les'* = *vs* @ [*Label n es'*
lfilledk'] @ *es''*
    using *LN Lfilled.simps*[*of k+1 lholed es''' les'*]
    by *fastforce*
  obtain $ts'\ ts''$ **where** *lab-def*:$\mathcal{S}\cdot\mathcal{C} \vdash vs : (t1s \rightarrow ts')$
                      $\mathcal{S}\cdot\mathcal{C} \vdash [\textit{Label n es' lfilledk}] : (ts' \rightarrow ts'')$
                      $\mathcal{S}\cdot\mathcal{C} \vdash es'' : (ts'' \rightarrow t2s)$

using *e-type-comp-conc2* $[OF\ LN(6)]$
by *blast*
obtain *tls ts-c C-int* **where** *int-def*: $ts'' = ts'\ @\ ts\text{-}c$
$\qquad\qquad\qquad\qquad\qquad$ *length tls* $= n$
$\qquad\qquad\qquad\qquad\qquad$ $\mathcal{S}\cdot\mathcal{C} \vdash es' : (tls \text{ -> } ts\text{-}c)$
$\qquad\qquad\qquad\qquad\qquad$ $\mathcal{C}\text{-}int = \mathcal{C}(\!|label := [tls]\ @\ label\ \mathcal{C}|\!)$
$\qquad\qquad\qquad\qquad\qquad$ $\mathcal{S}\cdot\mathcal{C}\text{-}int \vdash lfilledk : ([]\text{ -> } ts\text{-}c)$
$\quad$ **using** *e-type-label* $[OF\ lab\text{-}def(2)]$
$\quad$ **by** *blast*
have $(\bigwedge \mathcal{C}'\ arb\text{-}labs'\ ts\ ts'.$
$\qquad \mathcal{C}' = \mathcal{C}\text{-}int(\!|label := arb\text{-}labs'\ @\ label\ \mathcal{C}\text{-}int|\!) \Longrightarrow$
$\qquad \mathcal{S}\cdot\mathcal{C}' \vdash es : (ts \text{ -> } ts') \Longrightarrow$
$\qquad (\mathcal{S}\cdot\mathcal{C}' \vdash es''' : (ts \text{ -> } ts')))$
proof −
$\quad$ **fix** $\mathcal{C}''\ arb\text{-}labs''\ tts\ tts'$
$\quad$ **assume** $\mathcal{C}'' = \mathcal{C}\text{-}int(\!|label := arb\text{-}labs''\ @\ label\ \mathcal{C}\text{-}int|\!)$
$\qquad\qquad \mathcal{S}\cdot\mathcal{C}'' \vdash es : (tts \text{ -> } tts')$
$\quad$ **thus** $(\mathcal{S}\cdot\mathcal{C}'' \vdash es''' : (tts \text{ -> } tts'))$
$\qquad$ **using** $LN(7)[of\ arb\text{-}labs''\ @\ [tls]\ tts\ tts']\ int\text{-}def(4)$
$\qquad$ **by** *fastforce*
qed
hence $(\mathcal{S}\cdot\mathcal{C}\text{-}int \vdash lfilledk' : ([]\text{ -> } ts\text{-}c))$
$\quad$ **using** $LN(4)[OF\ l'\text{-}def(1)\ int\text{-}def(5)]$
$\quad$ **by** *fastforce*
hence $(\mathcal{S}\cdot\mathcal{C} \vdash [Label\ n\ es'\ lfilledk'] : (ts' \text{ -> } ts''))$
$\quad$ **using** *int-def e-typing-s-typing.intros(3,7)*
$\quad$ **by** (*metis append.right-neutral*)
thus *?case*
$\quad$ **using** *lab-def e-type-comp-conc l'-def(2)*
$\quad$ **by** *blast*
qed

lemma *types-preserved-e1*:
$\quad$ **assumes** $(\!|s;vs;es|\!)\ a\leadsto\text{-}i\ (\!|s';vs';es'|\!)$
$\qquad\qquad$ *store-typing s* $\mathcal{S}$
$\qquad\qquad$ *tvs* $=$ *map typeof vs*
$\qquad\qquad$ $i < length\ (inst\ s)$
$\qquad\qquad$ $\mathcal{C} = ((s\text{-}inst\ \mathcal{S})!i)(\!|trust\text{-}t := tr,\ local := (local\ ((s\text{-}inst\ \mathcal{S})!i)\ @\ tvs),\ label$
$:= arb\text{-}labs,\ return := arb\text{-}return|\!)$
$\qquad\qquad$ $\mathcal{S}\cdot\mathcal{C} \vdash es : (ts \text{ -> } ts')$
$\quad$ **shows** $(\mathcal{S}\cdot\mathcal{C} \vdash es' : (ts \text{ -> } ts')) \land (map\ typeof\ vs = map\ typeof\ vs')$
$\quad$ **using** *assms*
proof (*induction arbitrary*: *tr tvs C ts ts' arb-labs arb-return rule*: *reduce.induct*)
$\quad$ **case** (*basic e e' s vs i*)
$\quad$ **then show** *?case*
$\quad\quad$ **using** *types-preserved-b-e1* $[OF\ basic(1,2)]$
$\quad\quad$ **by** *fastforce*
next
$\quad$ **case** (*call s vs j i*)

obtain *tr' ts'' tf1 tf2* **where** *l-func-t*: *trust-compat* (*trust-t C*) *tr'*
                                    length (*func-t C*) > *j*
                                    ts = ts''@tf1
                                    ts' = ts''@tf2
                                    ((*func-t C*)!*j*) = (*tr'*,(*tf1 -> tf2*))
  using *b-e-type-call*[*of C Call j ts ts' j*] *call*(*5*)
    unlift-b-e[*of - - [Call j] (ts -> ts')*]
  by *fastforce*
have *i < length* (*s-inst S*)
  using *call*(*3*) *store-typing-imp-inst-length-eq*[*OF call*(*1*)]
  by *simp*
moreover
have *j < length* (*func-t* (*s-inst S ! i*))
  using *l-func-t*(*2*) *call*(*4*)
  by *simp*
ultimately
have *cl-typing S* (*sfunc s i j*) (*tr'*,(*tf1 -> tf2*))
  using *store-typing-imp-func-agree*[*OF call*(*1*)] *l-func-t*(*5*) *call*(*4*)
  by *fastforce*
thus *?case*
  using *e-typing-s-typing.intros*(*3*,*6*) *l-func-t*
  by *fastforce*
next
  case (*call-indirect-Some s i' c cl j tf vs sec*)
  show *?case*
    using *types-preserved-call-indirect-Some*[*OF call-indirect-Some*(*8*,*1*)]
      call-indirect-Some(*2*,*3*,*4*,*6*,*7*)
    by *fastforce*
next
  case (*call-indirect-None c j s i vs tf*)
  thus *?case*
    using *e-typing-s-typing.intros*(*4*)
    by *blast*
next
  case (*callcl-native cl i' j tfs es s t1s t2s ves vs n k m zs i*)
  thus *?case*
    using *types-preserved-callcl-native*
    by *fastforce*
next
  case (*callcl-host-Some cl t1s t2s f ves vcs n m s i s' vcs' vs*)
  thus *?case*
    using *types-preserved-callcl-host-some*
      by *fastforce*
next
  case (*callcl-host-None cl t1s t2s f ves vcs n m s vs i*)
  thus *?case*
    using *e-typing-s-typing.intros*(*4*)
    by *blast*
next

case (*get-local vi j s v vs i*)
  hence *i < length (s-inst S)*
    unfolding *list-all2-conv-all-nth store-typing.simps*
    by *fastforce*
  then have *local C = tvs*
    using *store-local-label-empty assms(2) get-local*
    by *fastforce*
  then show *?case*
    using *types-preserved-get-local get-local*
    by *fastforce*
next
  case (*set-local vi j s v vs v′ i*)
  hence *i < length (s-inst S)*
    unfolding *list-all2-conv-all-nth store-typing.simps*
    by *fastforce*
  hence *local C = tvs*
    using *store-local-label-empty assms(2) set-local*
    by *fastforce*
  thus *?case*
    using *set-local types-preserved-set-local*
    by *simp*
next
  case (*get-global s vs j i*)
  have *length (global C) > j*
    using *b-e-type-get-global get-global(5) unlift-b-e[of - - [Get-global j] (ts -> ts′)]*
    by *fastforce*
  hence *glob-agree (sglob s i j) ((global C)!j)*
   using *get-global(3,4) store-typing-imp-glob-agree[OF get-global(1)] store-typing-imp-inst-length-eq[OF get-global(1)]*
    by *fastforce*
  hence *typeof (g-val (sglob s i j)) = tg-t (global C ! j)*
    unfolding *glob-agree-def*
    by *simp*
  thus *?case*
    using *get-global(5) types-preserved-get-global*
    unfolding *glob-agree-def sglob-val-def*
    by *fastforce*
next
  case (*set-global s i j v s′ vs*)
  then show *?case*
    using *types-preserved-set-global*
    by *fastforce*
next
  case (*load-Some s i k m sec off t v vs a*)
  then show *?case*
    using *types-preserved-load(1) wasm-deserialise-type*
    by *blast*
next
  case (*load-None s i k off t vs a*)

     then show *?case*
      using *e-typing-s-typing.intros(4)*
      by *blast*
next
  case (*load-packed-Some tp sx s i k off t v vs a*)
  then show *?case*
    using *types-preserved-load(1) wasm-deserialise-type*
    by *blast*
next
  case (*load-packed-None s i k off tp vs t sx a*)
  then show *?case*
    using *e-typing-s-typing.intros(4)*
    by *blast*
next
  case (*store-Some t v s$'$ s i k off vs a*)
  then show *?case*
    using *types-preserved-store*
    by *blast*
next
  case (*store-None t v s i k off vs a*)
  then show *?case*
    using *e-typing-s-typing.intros(4)*
    by *blast*
next
  case (*store-packed-Some t v s$'$ s i k off tp vs a*)
  then show *?case*
    using *types-preserved-store*
    by *blast*
next
  case (*store-packed-None t v s i k off tp vs a*)
  then show *?case*
    using *e-typing-s-typing.intros(4)*
    by *blast*
next
  case (*current-memory s i n c vs*)
  then show *?case*
    using *types-preserved-current-memory*
    by *fastforce*
next
  case (*grow-memory s i n$'$ c$'$ c n s$'$ vs*)
  then show *?case*
    using *types-preserved-grow-memory*
    by *fastforce*
next
  case (*grow-memory-fail s vs c i*)
  thus *?case*
    using *types-preserved-grow-memory*
    by *blast*
next

101

case (*label s vs es a i s′ vs′ es′ k lholed les les′*)

{
  fix $\mathcal{C}′$ *arb-labs′ ts ts′*
  assume *local-assms*:$\mathcal{C}′ = \mathcal{C}(\!|label := arb\text{-}labs′@(label\ \mathcal{C}), return := (return\ \mathcal{C})|\!)$
  hence $(\mathcal{S}\cdot\mathcal{C}′ \vdash es : (ts \text{ -> } ts′)) \implies (\mathcal{S}\cdot\mathcal{C}′ \vdash es′ : (ts \text{ -> } ts′)) \wedge map\ typeof\ vs =$
map typeof vs′
    using *label(4)*[*OF label(5,6,7)*] *label(8)*
    by *fastforce*
  hence $(\mathcal{S}\cdot\mathcal{C}(\!|label := arb\text{-}labs′@(label\ \mathcal{C})|\!) \vdash es : (ts \text{ -> } ts′))$
        $\implies (\mathcal{S}\cdot\mathcal{C}(\!|label := arb\text{-}labs′@(label\ \mathcal{C})|\!) \vdash es′ : (ts \text{ -> } ts′)) \wedge$
          map typeof vs = map typeof vs′
    using *local-assms*
    by *simp*
}
hence $\bigwedge arb\text{-}labs′\ ts\ ts′.\ \mathcal{S}\cdot\mathcal{C}(\!|label := arb\text{-}labs′@(label\ \mathcal{C})|\!) \vdash es : (ts \text{ -> } ts′)$
                $\implies (\mathcal{S}\cdot\mathcal{C}(\!|label := arb\text{-}labs′@(label\ \mathcal{C})|\!) \vdash es′ : (ts \text{ -> } ts′))$
  map typeof vs = map typeof vs′
  using *types-exist-lfilled*[*OF label(2,9)*]
  by *auto*
thus *?case*
  using *lholed-same-type*[*OF label(2,3,9)*]
  by *fastforce*
next
  case (*local s vls es a i s′ vs′ es′ v0s n j*)
  obtain $\mathcal{C}′$ *tls* **where** *es-def*:$i < length\ (s\text{-}inst\ \mathcal{S})$
                  length tls = n
                  $\mathcal{C}′ = (s\text{-}inst\ \mathcal{S}\ !\ i)\ (\!|trust\text{-}t := trust\text{-}t\ \mathcal{C}, local := local(s\text{-}inst$
$\mathcal{S}\ !\ i)\ @\ map\ typeof\ vls, label := label\ (s\text{-}inst\ \mathcal{S}\ !\ i), return := Some\ tls|\!)$
                  $\mathcal{S}\cdot\mathcal{C}′ \vdash es : ([] \text{ -> } tls)$
                  $ts′ = ts\ @\ tls$
    using *e-type-local*[*OF local(7)*]
    by *fastforce*
  moreover
  obtain $ts′′$ **where** $ts′ = ts@ts′′$ $(\mathcal{S}\cdot(trust\text{-}t\ \mathcal{C})\cdot(Some\ ts′′) \Vdash\text{-}i\ vls;es : ts′′)$
    using *e-type-local-shallow local(7)*
    by *fastforce*
  moreover
  have *inst-typing* $\mathcal{S}\ ((inst\ s)!i)\ ((s\text{-}inst\ \mathcal{S})!i)\ i < length\ (inst\ s)$
    using *local es-def(1)*
    unfolding *store-typing.simps list-all2-conv-all-nth*
    by *fastforce+*
  ultimately
  have $\mathcal{S}\cdot\mathcal{C}′ \vdash es′ : ([] \text{ -> } tls)\ map\ typeof\ vls = map\ typeof\ vs′$
    using *local(2)*[*OF local(3) - - - es-def(4), of map typeof vls Some tls label*
$(s\text{-}inst\ \mathcal{S}\ !\ i)$]
    by *fastforce+*
  hence $\mathcal{S}\cdot(trust\text{-}t\ \mathcal{C})\cdot(Some\ tls) \Vdash\text{-}\ i\ vs′;es′ : tls$
    using *e-typing-s-typing.intros(8) es-def(1,3)*
    by *fastforce*

thus *?case*
    using *e-typing-s-typing.intros(3,5) es-def(2,5)*
    by *fastforce*
qed

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*
  using *assms*
proof −
  have *i < (length (s-inst $\mathcal{S}$))*
    using *assms(3) s-typing.cases*
    by *blast*
  moreover
  hence *i-bound:i < length (inst s)*
    using *assms(2)*
    unfolding *list-all2-conv-all-nth store-typing.simps*
    by *fastforce*
  obtain *tvs $\mathcal{C}$* **where** *defs: tvs = map typeof vs*
                  $\mathcal{C} = ((\text{s-inst } \mathcal{S})!i)(\!|trust\text{-}t := tr, local := (local ((\text{s-inst } \mathcal{S})!i)$
@ *tvs), label := (label ((s-inst $\mathcal{S}$)!i)), return := None*$|\!)$
                  $\mathcal{S}{\cdot}\mathcal{C} \vdash es : (\![] \text{ -> } ts)$
    using *assms(3)*
    unfolding *s-typing.simps*
    by *fastforce*
  have $(\mathcal{S}{\cdot}\mathcal{C} \vdash es' : (\![] \text{ -> } ts)) \land (map\ typeof\ vs = map\ typeof\ vs')$
    using *types-preserved-e1[OF assms(1,2) defs(1) i-bound defs(2,3)]*
    by *simp*
  ultimately show *?thesis*
    using *defs*
    unfolding *s-typing.simps*
    by *auto*
qed

6.2   Progress

lemma *const-list-no-progress*:
  assumes *const-list es*
  shows $\lnot(\!|s;vs;es|\!)$ $a\leadsto$- i $(\!|s';vs';es'|\!)$
proof −
  {
    assume $(\!|s;vs;es|\!)$ $a\leadsto$- i $(\!|s';vs';es'|\!)$
    hence *False*
      using *assms*
    proof (*induction rule: reduce.induct*)
      case (*basic e a e' s vs i*)
      thus *?thesis*

103

proof (*induction rule*: *reduce-simple.induct*)
  case (*trap es lholed*)
  show *?case*
    using *trap*(*2*)
  proof (*cases rule*: *Lfilled.cases*)
    case (*L0 vs es′*)
    thus *?thesis*
      using *trap*(*3*) *list-all-append const-list-cons-last*(*2*)[*of vs Trap*]
      unfolding *const-list-def*
      by (*simp add*: *is-const-def*)
  next
    case (*LN vs n es′ l es″ k lfilledk*)
    thus *?thesis*
      by (*simp add*: *is-const-def*)
  qed
  qed (*fastforce simp add*: *const-list-cons-last*(*2*) *is-const-def const-list-def*)+
next
  case (*label s vs es a i s′ vs′ es′ k lholed les les′*)
  show *?case*
    using *label*(*2*)
  proof (*cases rule*: *Lfilled.cases*)
    case (*L0 vs es′*)
    thus *?thesis*
      using *label*(*4,5*) *list-all-append*
      unfolding *const-list-def*
      by *fastforce*
  next
    case (*LN vs n es′ l es″ k lfilledk*)
    thus *?thesis*
      using *label*(*4,5*)
      unfolding *const-list-def*
      by (*simp add*: *is-const-def*)
  qed
  qed (*fastforce simp add*: *const-list-cons-last*(*2*) *is-const-def const-list-def*)+
  }
  thus *?thesis*
    by *blast*
qed

lemma *empty-no-progress*:
  assumes *es* = []
  shows ¬⦇*s*;*vs*;*es*⦈ *a*⇝- *i* ⦇*s′*;*vs′*;*es′*⦈
proof −
  {
    assume ⦇*s*;*vs*;*es*⦈ *a*⇝- *i* ⦇*s′*;*vs′*;*es′*⦈
    hence *False*
      using *assms*
    proof (*induction rule*: *reduce.induct*)
      case (*basic e a e′ s vs i*)

```
      thus ?thesis
      proof (induction rule: reduce-simple.induct)
        case (trap es lholed)
        thus ?case
          using Lfilled.simps[of 0 lholed [Trap] es]
          by auto
      qed auto
    next
      case (label s vs es a i s' vs' es' k lholed les les')
        thus ?case
          using Lfilled.simps[of k lholed es []]
          by auto
    qed auto
  }
  thus ?thesis
    by blast
qed

lemma trap-no-progress:
  assumes es = [Trap]
  shows ¬(|s;vs;es|) a⤳- i (|s';vs';es'|)
proof −
  {
    assume (|s;vs;es|) a⤳- i (|s';vs';es'|)
    hence False
      using assms
    proof (induction rule: reduce.induct)
      case (basic e a e' s vs i)
      thus ?case
      by (induction rule: reduce-simple.induct) auto
    next
      case (label s vs es a i s' vs' es' k lholed les les')
      show ?case
        using label(2)
        proof (cases rule: Lfilled.cases)
          case (L0 vs es')
          show ?thesis
            using L0(2) label(1,4,5) empty-no-progress
            by (auto simp add: Cons-eq-append-conv)
        next
          case (LN vs n es' l es'' k' lfilledk)
          show ?thesis
            using LN(2) label(5)
            by (simp add: Cons-eq-append-conv)
        qed
    qed auto
  }
  thus ?thesis
    by blast
```

105

qed

lemma *terminal-no-progress*:
  assumes *const-list es* $\vee$ *es* = [*Trap*]
  shows $\neg(\!|s;vs;es|\!)$ $a \leadsto\text{-} i$ $(\!|s';vs';es'|\!)$
  using *const-list-no-progress trap-no-progress assms*
  by *blast*

lemma *progress-L0*:
  assumes $(\!|s;vs;es|\!)$ $a \leadsto\text{-} i$ $(\!|s';vs';es'|\!)$
       const-list cs
  shows $(\!|s;vs;cs@es@es\text{-}c|\!)$ $a \leadsto\text{-} i$ $(\!|s';vs';cs@es'@es\text{-}c|\!)$
proof $-$
  have $\bigwedge es.$ *Lfilled 0* (*LBase cs es-c*) *es* (*cs@es@es-c*)
    using *Lfilled.intros(1)*[*of cs* (*LBase cs es-c*) *es-c*] *assms(2)*
    unfolding *const-list-def*
    by *fastforce*
  thus *?thesis*
    using *reduce.intros(23) assms(1)*
    by *blast*
qed

lemma *progress-L0-left*:
  assumes $(\!|s;vs;es|\!)$ $a \leadsto\text{-} i$ $(\!|s';vs';es'|\!)$
       const-list cs
  shows $(\!|s;vs;cs@es|\!)$ $a \leadsto\text{-} i$ $(\!|s';vs';cs@es'|\!)$
  using *assms progress-L0*[**where** *?es-c* = []]
  by *fastforce*

lemma *progress-L0-trap*:
  assumes *const-list cs*
      cs $\neq$ [] $\vee$ *es* $\neq$ []
  shows $\exists\, a.$ $(\!|s;vs;cs@[Trap]@es|\!)$ $a \leadsto\text{-} i$ $(\!|s;vs;[Trap]|\!)$
proof $-$
  have *cs* @ [*Trap*] @ *es* $\neq$ [*Trap*]
    using *assms(2)*
    by (*cases cs* = []) (*auto simp add*: *append-eq-Cons-conv*)
  thus *?thesis*
    using *reduce.intros(1) assms(2) reduce-simple.trap*
        Lfilled.intros(1)[*OF assms(1), of - es* [*Trap*]]
    by *blast*
qed

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

106

$$\land \; \textit{const-list vs}$$

    using *assms*

proof (*induction* [$Br (j+k)] *es arbitrary*: *k* $\mathcal{C}$ *ts rule*: *Lfilled.induct*)

  case (*L0 vs lholed es′*)

  obtain *ts′ ts″* **where** *ts-def*:$\mathcal{S}\cdot\mathcal{C} \vdash vs : ([] \; \text{-}\!> \; ts′)$

                            $\mathcal{S}\cdot\mathcal{C} \vdash [\$Br \; k] : (ts′ \; \text{-}\!> \; ts″)$

                            $\mathcal{S}\cdot\mathcal{C} \vdash es′ : (ts″ \; \text{-}\!> \; ts)$

    using *e-type-comp-conc2*[*OF L0(3)*]

    by *fastforce*

  obtain *ts-c* **where** *ts′ = ts-c @ tvs*

    using *b-e-type-br*[*of* $\mathcal{C}$ *Br k ts′ ts″*] *L0(3,4) ts-def(2) unlift-b-e*

    by *fastforce*

  then obtain *vs1 vs2* **where** *vs-def*:$\mathcal{S}\cdot\mathcal{C} \vdash vs1 : ([] \; \text{-}\!> \; ts\text{-}c)$

                               $\mathcal{S}\cdot\mathcal{C} \vdash vs2 : (ts\text{-}c \; \text{-}\!> \; (ts\text{-}c@tvs))$

                               vs = vs1@vs2

                               const-list vs1

                               const-list vs2

    using *e-type-const-list-cons*[*OF L0(1)*] *ts-def(1)*

    by *fastforce*

  hence $\mathcal{S}\cdot\mathcal{C} \vdash vs2 : ([] \; \text{-}\!> \; tvs)$

    using *e-type-const-list* **by** *blast*

  thus *?case*

    using *Lfilled.intros(1)*[*OF vs-def(4), of - es′ vs2@*[$Br k]] *vs-def(3,5)*

    by *fastforce*

next

  case (*LN vs lholed n es′ l es″ j lfilledk*)

  obtain *t1s t2s* **where** *ts-def*:$\mathcal{S}\cdot\mathcal{C} \vdash vs : ([] \; \text{-}\!> \; t1s)$

                        $\mathcal{S}\cdot\mathcal{C} \vdash [Label \; n \; es′ \; lfilledk] : (t1s \; \text{-}\!> \; t2s)$

                        $\mathcal{S}\cdot\mathcal{C} \vdash es″ : (t2s \; \text{-}\!> \; ts)$

  using *e-type-comp-conc2*[*OF LN(5)*]

  by *fastforce*

  obtain *ts′ ts-l* **where** *ts-l-def*:$\mathcal{S}\cdot\mathcal{C}(\!|label := [ts′] \; @ \; label \; \mathcal{C}|\!) \vdash lfilledk : ([] \; \text{-}\!> \; ts\text{-}l)$

    using *e-type-label*[*OF ts-def(2)*]

    by *fastforce*

  obtain *lholed′ vs′ $\mathcal{C}'$* **where** *lfilledk-def*:*Lfilled j lholed′* (*vs′* @ [$Br (j + (1 + k))]) *lfilledk*

                             $\mathcal{S}\cdot\mathcal{C}' \vdash vs′ : ([] \; \text{-}\!> \; tvs)$

                             const-list vs′

    using *LN(4)*[*OF - ts-l-def, of 1 + k*] *LN(5,6)*

    by *fastforce*

  thus *?case*

    using *Lfilled.intros(2)*[*OF LN(1) - lfilledk-def(1)*]

    by *fastforce*

qed

lemma *progress-LN-return*:

  assumes (*Lfilled j lholed* [$Return] *es*)

      $\mathcal{S}\cdot\mathcal{C} \vdash es : ([] \; \text{-}\!> \; ts)$

      (*return* $\mathcal{C}$) *= Some tvs*

shows $\exists$ *lholed$'$ vs $\mathcal{C}'$.* (*Lfilled j lholed$'$* (*vs*@[*$Return*]) *es*)
$\qquad\qquad \land$ ($\mathcal{S}\cdot\mathcal{C}' \vdash vs : ([] \text{ -> } tvs)$)
$\qquad\qquad \land$ *const-list vs*
  using *assms*
proof (*induction* [*$Return*] *es arbitrary: k $\mathcal{C}$ ts rule: Lfilled.induct*)
  case (*L0 vs lholed es$'$*)
  obtain *ts$'$ ts$''$* **where** *ts-def*:$\mathcal{S}\cdot\mathcal{C} \vdash vs : ([] \text{ -> } ts')$
$\qquad\qquad\qquad\qquad\qquad\quad \mathcal{S}\cdot\mathcal{C} \vdash [\$Return] : (ts' \text{ -> } ts'')$
$\qquad\qquad\qquad\qquad\qquad\quad \mathcal{S}\cdot\mathcal{C} \vdash es' : (ts'' \text{ -> } ts)$
    using *e-type-comp-conc2*[*OF L0(3)*]
    by *fastforce*
  obtain *ts-c* **where** *ts$'$ = ts-c @ tvs*
    using *b-e-type-return*[*of $\mathcal{C}$ Return ts$'$ ts$''$*] *L0(3,4) ts-def(2) unlift-b-e*
    by *fastforce*
  then obtain *vs1 vs2* **where** *vs-def*:$\mathcal{S}\cdot\mathcal{C} \vdash vs1 : ([] \text{ -> } ts\text{-}c)$
$\qquad\qquad\qquad\qquad\qquad\qquad\quad \mathcal{S}\cdot\mathcal{C} \vdash vs2 : (ts\text{-}c \text{ -> } (ts\text{-}c@tvs))$
$\qquad\qquad\qquad\qquad\qquad\qquad\quad vs = vs1@vs2$
$\qquad\qquad\qquad\qquad\qquad\qquad\quad const\text{-}list\ vs1$
$\qquad\qquad\qquad\qquad\qquad\qquad\quad const\text{-}list\ vs2$
    using *e-type-const-list-cons*[*OF L0(1)*] *ts-def(1)*
    by *fastforce*
  hence $\mathcal{S}\cdot\mathcal{C} \vdash vs2 : ([] \text{ -> } tvs)$
    using *e-type-const-list* **by** *blast*
  thus *?case*
    using *Lfilled.intros(1)*[*OF vs-def(4), of - es$'$ vs2*@[*$Return*]] *vs-def(3,5)*
    by *fastforce*
next
  case (*LN vs lholed n es$'$ l es$''$ j lfilledk*)
  obtain *t1s t2s* **where** *ts-def*:$\mathcal{S}\cdot\mathcal{C} \vdash vs : ([] \text{ -> } t1s)$
$\qquad\qquad\qquad\qquad\qquad\quad \mathcal{S}\cdot\mathcal{C} \vdash [Label\ n\ es'\ lfilledk] : (t1s \text{ -> } t2s)$
$\qquad\qquad\qquad\qquad\qquad\quad \mathcal{S}\cdot\mathcal{C} \vdash es'' : (t2s \text{ -> } ts)$
  using *e-type-comp-conc2*[*OF LN(5)*]
  by *fastforce*
  obtain *ts$'$ ts-l* **where** *ts-l-def*:$\mathcal{S}\cdot\mathcal{C}(\!|label := [ts'] \text{ @ } label\ \mathcal{C}|\!) \vdash lfilledk : ([] \text{ -> } ts\text{-}l)$
    using *e-type-label*[*OF ts-def(2)*]
    by *fastforce*
  obtain *lholed$'$ vs$'$ $\mathcal{C}'$* **where** *lfilledk-def*:*Lfilled j lholed$'$* (*vs$'$* @ [*$Return*]) *lfilledk*
$\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathcal{S}\cdot\mathcal{C}' \vdash vs' : ([] \text{ -> } tvs)$
$\qquad\qquad\qquad\qquad\qquad\qquad\qquad const\text{-}list\ vs'$
    using *LN(4)*[*OF ts-l-def*] *LN(6)*
    by *fastforce*
  thus *?case*
    using *Lfilled.intros(2)*[*OF LN(1) - lfilledk-def(1)*]
    by *fastforce*
qed

lemma *progress-LN1*:
  assumes (*Lfilled j lholed* [*$Br (j+k)*] *es*)
$\qquad\qquad \mathcal{S}\cdot\mathcal{C} \vdash es : (ts \text{ -> } ts')$

shows *length* (*label* $\mathcal{C}$) $> k$
  using *assms*
proof (*induction* [*\$Br* (*j*+*k*)] *es arbitrary*: *k* $\mathcal{C}$ *ts ts' rule*: *Lfilled.induct*)
  case (*L0 vs lholed es'*)
  obtain *ts''* *ts'''* **where** *ts-def*:$\mathcal{S}\cdot\mathcal{C} \vdash vs : (ts \to ts'')$
                              $\mathcal{S}\cdot\mathcal{C} \vdash [\$Br\ k] : (ts'' \to ts''')$
                              $\mathcal{S}\cdot\mathcal{C} \vdash es' : (ts''' \to ts')$
    using *e-type-comp-conc2*[*OF L0(3)*]
    by *fastforce*
  thus *?case*
    using *b-e-type-br(1)*[*of - Br k ts'' ts'''*] *unlift-b-e*
    by *fastforce*
next
  case (*LN vs lholed n es' l es'' k' lfilledk*)
  obtain *t1s t2s* **where** *ts-def*:$\mathcal{S}\cdot\mathcal{C} \vdash vs : (ts \to t1s)$
                            $\mathcal{S}\cdot\mathcal{C} \vdash [Label\ n\ es'\ lfilledk] : (t1s \to t2s)$
                            $\mathcal{S}\cdot\mathcal{C} \vdash es'' : (t2s \to ts')$
  using *e-type-comp-conc2*[*OF LN(5)*]
  by *fastforce*
  obtain *ts'' ts-l* **where** *ts-l-def*:$\mathcal{S}\cdot\mathcal{C}(\!|label := [ts'] @ label\ \mathcal{C}|\!) \vdash lfilledk : ([] \to$
ts-l)
    using *e-type-label*[*OF ts-def(2)*]
    by *fastforce*
  thus *?case*
    using *LN(4)*[*of 1+k*]
    by *fastforce*
qed

lemma *progress-LN2*:
  assumes (*Lfilled j lholed e1s lfilled*)
  shows $\exists$ *lfilled'*. (*Lfilled j lholed e2s lfilled'*)
  using *assms*
proof (*induction rule*: *Lfilled.induct*)
  case (*L0 vs lholed es' es*)
  thus *?case*
    using *Lfilled.intros(1)*
    by *fastforce*
next
  case (*LN vs lholed n es' l es'' k es lfilledk*)
  thus *?case*
    using *Lfilled.intros(2)*
    by *fastforce*
qed

lemma *const-of-const-list*:
  assumes *length cs = 1*
        const-list cs
  shows $\exists v.\ cs = [\$C\ v]$
  using *e-type-const-unwrap assms*

109

unfolding *const-list-def list-all-length*
  by (*metis append-butlast-last-id append-self-conv2 gr-zeroI last-conv-nth length-butlast*
          length-greater-0-conv less-numeral-extra(1,4) zero-less-diff)

lemma *const-of-i32*:
  assumes *const-list cs*
          $\mathcal{S} \cdot \mathcal{C} \vdash cs : ([] \to [(T\text{-}i32\ sec)])$
  shows $\exists c.\ cs = [\$C\ ConstInt32\ sec\ c]$
proof −
  obtain v **where** $cs = [\$C\ v]$
    using *const-of-const-list assms(1) e-type-const-list[OF assms]*
    by *fastforce*
  moreover
  hence $\mathcal{C} \vdash [C\ v] : ([] \to [(T\text{-}i32\ sec)])$
    using *assms(2) unlift-b-e*
    by *fastforce*
  hence $\exists c.\ v = ConstInt32\ sec\ c$
  proof (*induction* $[C\ v]$ $([] \to [(T\text{-}i32\ sec)])$ *rule: b-e-typing.induct*)
    case (*const* $\mathcal{C}$)
    then show *?case*
      unfolding *typeof-def*
      by (*cases v, auto*)
  qed *auto*
  ultimately
  show *?thesis*
    by *fastforce*
qed

lemma *const-of-i64*:
  assumes *const-list cs*
          $\mathcal{S} \cdot \mathcal{C} \vdash cs : ([] \to [(T\text{-}i64\ sec)])$
  shows $\exists c.\ cs = [\$C\ ConstInt64\ sec\ c]$
proof −
  obtain v **where** $cs = [\$C\ v]$
    using *const-of-const-list assms(1) e-type-const-list[OF assms]*
    by *fastforce*
  moreover
  hence $\mathcal{C} \vdash [C\ v] : ([] \to [(T\text{-}i64\ sec)])$
    using *assms(2) unlift-b-e*
    by *fastforce*
  hence $\exists c.\ v = ConstInt64\ sec\ c$
  proof (*induction* $[C\ v]$ $([] \to [(T\text{-}i64\ sec)])$ *rule: b-e-typing.induct*)
    case (*const* $\mathcal{C}$)
    then show *?case*
      unfolding *typeof-def*
      by (*cases v, auto*)
  qed *auto*
  ultimately
  show *?thesis*

by *fastforce*

qed

lemma *const-of-f32*:
  assumes *const-list cs*
       $\mathcal{S} \cdot \mathcal{C} \vdash cs : ([] \rightarrow [\textit{T-f32}])$
  shows $\exists c.\ cs = [\$C\ \textit{ConstFloat32}\ c]$
proof −
  obtain v **where** $cs = [\$C\ v]$
    using *const-of-const-list assms*(*1*) *e-type-const-list*[*OF assms*]
    by *fastforce*
  moreover
  hence $\mathcal{C} \vdash [C\ v] : ([] \rightarrow [\textit{T-f32}])$
    using *assms*(*2*) *unlift-b-e*
    by *fastforce*
  hence $\exists c.\ v = \textit{ConstFloat32}\ c$
  proof (*induction* $[C\ v]$ ($[]$ -> $[\textit{T-f32}]$) *rule*: *b-e-typing.induct*)
    case (*const* $\mathcal{C}$)
    then show *?case*
      unfolding *typeof-def*
      by (*cases v*, *auto*)
  qed *auto*
  ultimately
  show *?thesis*
    by *fastforce*
qed

lemma *const-of-f64*:
  assumes *const-list cs*
       $\mathcal{S} \cdot \mathcal{C} \vdash cs : ([] \rightarrow [\textit{T-f64}])$
  shows $\exists c.\ cs = [\$C\ \textit{ConstFloat64}\ c]$
proof −
  obtain v **where** $cs = [\$C\ v]$
    using *const-of-const-list assms*(*1*) *e-type-const-list*[*OF assms*]
    by *fastforce*
  moreover
  hence $\mathcal{C} \vdash [C\ v] : ([] \rightarrow [\textit{T-f64}])$
    using *assms*(*2*) *unlift-b-e*
    by *fastforce*
  hence $\exists c.\ v = \textit{ConstFloat64}\ c$
  proof (*induction* $[C\ v]$ ($[]$ -> $[\textit{T-f64}]$) *rule*: *b-e-typing.induct*)
    case (*const* $\mathcal{C}$)
    then show *?case*
      unfolding *typeof-def*
      by (*cases v*, *auto*)
  qed *auto*
  ultimately
  show *?thesis*
    by *fastforce*

qed

lemma *progress-unop-testop-i*:
  assumes $\mathcal{S} \cdot \mathcal{C} \vdash cs : ([] \to [t])$
        is-int-t t
        const-list cs
        $e = Unop\text{-}i\ t\ iop \lor e = Testop\ t\ testop$
  shows $\exists\, a\ s'\ vs'\ es'.\ (\!|s;vs;cs@([\$e])|\!)\ a\rightsquigarrow\text{-}i\ (\!|s';vs';es'|\!)$
  using *assms*(*2*)
proof (*cases t*)
  case *T-i32*
  thus *?thesis*
    using *const-of-i32*[*OF assms*(*3*)] *assms*(*1*,*4*)
      reduce.intros(*1*)[*OF reduce-simple.intros*(*1*)] *reduce.intros*(*1*)[*OF reduce-simple.intros*(*13*)]
    by *fastforce*
next
  case *T-i64*
  thus *?thesis*
    using *const-of-i64*[*OF assms*(*3*)] *assms*(*1*,*4*)
      reduce.intros(*1*)[*OF reduce-simple.intros*(*2*)] *reduce.intros*(*1*)[*OF reduce-simple.intros*(*14*)]
    by *fastforce*
qed (*simp-all add*: *is-int-t-def*)

lemma *progress-unop-f*:
  assumes $\mathcal{S} \cdot \mathcal{C} \vdash cs : ([] \to [t])$
        is-float-t t
        const-list cs
        $e = Unop\text{-}f\ t\ iop$
  shows $\exists\, a\ s'\ vs'\ es'.\ (\!|s;vs;cs@([\$e])|\!)\ a\rightsquigarrow\text{-}i\ (\!|s';vs';es'|\!)$
  using *assms*(*2*)
proof (*cases t*)
  case *T-f32*
  thus *?thesis*
    using *const-of-f32*[*OF assms*(*3*)] *assms*(*1*,*4*)
      reduce.intros(*1*)[*OF reduce-simple.intros*(*3*)] *reduce.intros*(*1*)[*OF reduce-simple.intros*(*13*)]
    by *fastforce*
next
  case *T-f64*
  thus *?thesis*
    using *const-of-f64*[*OF assms*(*3*)] *assms*(*1*,*4*)
      reduce.intros(*1*)[*OF reduce-simple.intros*(*4*)] *reduce.intros*(*1*)[*OF reduce-simple.intros*(*14*)]
    by *fastforce*
qed (*simp-all add*: *is-float-t-def*)

lemma *const-list-split-2*:
  assumes *const-list cs*
        $\mathcal{S} \cdot \mathcal{C} \vdash cs : ([] \to [t1,\ t2])$
  shows $\exists\, c1\ c2.\ (\mathcal{S} \cdot \mathcal{C} \vdash [c1] : ([] \to [t1]))$
            $\land\ (\mathcal{S} \cdot \mathcal{C} \vdash [c2] : ([] \to [t2]))$

112

$\wedge$ *cs = [c1, c2]*
$\wedge$ *const-list [c1]*
$\wedge$ *const-list [c2]*

proof $-$
  have *l-cs:length cs = 2*
    using *assms e-type-const-list[OF assms]*
    by *simp*
  then obtain *c1 c2* **where** *cs!0 = c1 cs!1 = c2*
    by *fastforce*
  hence *cs = [c1] @ [c2]*
    using *assms e-type-const-conv-vs typing-map-typeof*
    by *fastforce*
  thus *?thesis*
    using *assms e-type-comp[of $\mathcal{S} \cdot \mathcal{C}$ [c1] c2] e-type-const[of c2 $\mathcal{S} \cdot \mathcal{C}$ - [t1,t2]]*
    unfolding *const-list-def*
    by *fastforce*
qed

lemma *const-list-split-3*:
  assumes *const-list cs*
       $\mathcal{S} \cdot \mathcal{C} \vdash cs : ([] \rightarrow [t1, t2, t3])$
  shows $\exists c1\ c2\ c3.\ (\mathcal{S} \cdot \mathcal{C} \vdash [c1] : ([] \rightarrow [t1]))$
           $\wedge\ (\mathcal{S} \cdot \mathcal{C} \vdash [c2] : ([] \rightarrow [t2]))$
           $\wedge\ (\mathcal{S} \cdot \mathcal{C} \vdash [c3] : ([] \rightarrow [t3]))$
           $\wedge\ cs = [c1,\ c2,\ c3]$
proof $-$
  have *l-cs:length cs = 3*
    using *assms e-type-const-list[OF assms]*
    by *simp*
  then obtain *c1 c2 c3* **where** *cs!0 = c1 cs!1 = c2 cs!2 = c3*
    by *fastforce*
  hence *cs = [c1] @ [c2] @ [c3]*
    using *assms e-type-const-conv-vs typing-map-typeof*
    by *fastforce*
  thus *?thesis*
    using *assms e-type-comp-conc2[of $\mathcal{S} \mathcal{C}$ [c1] [c2] [c3] [] [t1,t2,t3]]*
        e-type-const[of c1] e-type-const[of c2] e-type-const[of c3]
    unfolding *const-list-def*
    by *fastforce*
qed

lemma *progress-binop-relop-i*:
  assumes $\mathcal{S} \cdot \mathcal{C} \vdash cs : ([] \rightarrow [t, t])$
       is-int-t t
       const-list cs
       e = Binop-i t iop $\vee$ e = Relop-i t irop
  shows $\exists a\ s'\ vs'\ es'.\ (\!|s;vs;cs@([\$e])|\!)\ a \leadsto\text{-}i\ (\!|s';vs';es'|\!)$
  using *assms(2)*
proof (*cases t*)

113

case (*T-i32 sec*)
hence *cs-def*:∃ *c1 c2. cs* = [$*C ConstInt32 sec c1*,$*C ConstInt32 sec c2*]
  using *const-list-split-2*[*OF assms(3,1)*] *assms(3) const-of-i32*
  unfolding *const-list-def*
  by *blast*
show *?thesis*
proof (*cases e = Binop-i t iop*)
  case *True*
  obtain *c1 c2* **where** *cs* = [$*C ConstInt32 sec c1*,$*C ConstInt32 sec c2*]
    using *cs-def*
    by *blast*
  thus *?thesis*
    apply (*cases app-binop-i iop c1 c2*)
    apply (*metis reduce-simple.intros(6) reduce.intros(1) T-i32 True append-Cons append-Nil*)
    apply (*metis reduce-simple.intros(5) reduce.intros(1) T-i32 True append-Cons append-Nil*)
      done
  next
    case *False*
    thus *?thesis*
    using *reduce-simple.intros(15) assms(4) reduce.intros(1) cs-def T-i32*
    by *fastforce*
  qed
next
  case (*T-i64 sec*)
  hence *cs-def*:∃ *c1 c2. cs* = [$*C ConstInt64 sec c1*,$*C ConstInt64 sec c2*]
    using *const-list-split-2*[*OF assms(3,1)*] *assms(3) const-of-i64*
    unfolding *const-list-def*
    by *blast*
  show *?thesis*
  proof (*cases e = Binop-i t iop*)
    case *True*
    obtain *c1 c2* **where** *cs* = [$*C ConstInt64 sec c1*,$*C ConstInt64 sec c2*]
      using *cs-def*
      by *blast*
    thus *?thesis*
      apply (*cases app-binop-i iop c1 c2*)
      apply (*metis reduce-simple.intros(8) reduce.intros(1) T-i64 True append-Cons append-Nil*)
      apply (*metis reduce-simple.intros(7) reduce.intros(1) T-i64 True append-Cons append-Nil*)
        done
    next
      case *False*
      thus *?thesis*
      using *reduce-simple.intros(16) assms(4) reduce.intros(1) cs-def T-i64*
      by *fastforce*
    qed

qed (*simp-all add*: *is-int-t-def*)

lemma *progress-binop-relop-f*:
  assumes $\mathcal{S} \cdot \mathcal{C} \vdash cs : ([] \rightarrow [t, t])$
        is-float-t t
        const-list cs
        $e = \text{Binop-f } t \text{ fop} \lor e = \text{Relop-f } t \text{ frop}$
  shows $\exists a \ s' \ vs' \ es'. \ (\!|s;vs;cs@([\$e])|\!) \ a \leadsto\text{-}i \ (\!|s';vs';es'|\!)$
  using *assms(2)*
proof (*cases t*)
  case *T-f32*
  hence *cs-def*:$\exists c1 \ c2. \ cs = [\$C \ ConstFloat32 \ c1,\$C \ ConstFloat32 \ c2]$
    using *const-list-split-2*[*OF assms(3,1)*] *assms(3) const-of-f32*
    unfolding *const-list-def*
    by *blast*
  show *?thesis*
  proof (*cases e = Binop-f t fop*)
    case *True*
    obtain *c1 c2* **where** *cs-def*:$cs = [\$C \ ConstFloat32 \ c1,\$C \ ConstFloat32 \ c2]$
      using *cs-def*
      by *blast*
    thus *?thesis*
      apply (*cases app-binop-f fop c1 c2*)
      apply (*metis reduce-simple.intros(10) reduce.intros(1) T-f32 True append-Cons append-Nil*)
      apply (*metis reduce-simple.intros(9) reduce.intros(1) T-f32 True append-Cons append-Nil*)
    done
  next
    case *False*
    thus *?thesis*
    using *reduce-simple.intros(17) assms(4) reduce.intros(1) cs-def T-f32*
    by *fastforce*
  qed
next
  case *T-f64*
  hence *cs-def*:$\exists c1 \ c2. \ cs = [\$C \ ConstFloat64 \ c1,\$C \ ConstFloat64 \ c2]$
    using *const-list-split-2*[*OF assms(3,1)*] *assms(3) const-of-f64*
    unfolding *const-list-def*
    by *blast*
  show *?thesis*
  proof (*cases e = Binop-f t fop*)
    case *True*
    obtain *c1 c2* **where** $cs = [\$C \ ConstFloat64 \ c1,\$C \ ConstFloat64 \ c2]$
      using *cs-def*
      by *blast*
    thus *?thesis*
      apply (*cases app-binop-f fop c1 c2*)
      apply (*metis reduce-simple.intros(12) reduce.intros(1) T-f64 True append-Cons*

append-Nil)
    apply (*metis reduce-simple.intros*(*11*) *reduce.intros*(*1*) *T-f64 True append-Cons*
append-Nil)
    done
  next
    case *False*
    thus *?thesis*
      using *reduce-simple.intros*(*18*) *assms*(*4*) *reduce.intros*(*1*) *cs-def T-f64*
    by *fastforce*
  qed
qed (*simp-all add*: *is-float-t-def*)

lemma *progress-b-e*:
  assumes $\mathcal{C} \vdash$ *b-es* : (*ts -> ts′*)
      $\mathcal{S} \cdot \mathcal{C} \vdash cs$: ([] *-> ts*)
      ($\bigwedge$*lholed.* ¬(*Lfilled 0 lholed* [$*Return*] (*cs*@($\ast$*b-es*))))
      $\bigwedge$ *i lholed.* ¬(*Lfilled 0 lholed* [$*Br* (*i*)] (*cs*@($\ast$*b-es*)))
      const-list cs
      ¬ *const-list* ($\ast$ *b-es*)
      i < *length* (*s-inst* $\mathcal{S}$)
      length (*local* $\mathcal{C}$) = *length* (*vs*)
      option-projr (*memory* $\mathcal{C}$) = *map-option* ($\lambda j.$ *snd* ((*mem s*)!*j*)) (*smem-ind*
s i)
  shows $\exists$ *a s′ vs′ es′.* (|*s;vs;cs*@($\ast$*b-es*)|) *a*⤳*-i* (|*s′;vs′;es′*|)
  using *assms*
proof (*induction b-es* (*ts -> ts′*) *arbitrary*: *ts ts′ cs rule*: *b-e-typing.induct*)
  case (*const* $\mathcal{C}$ *v*)
  then show *?case*
    unfolding *const-list-def is-const-def*
    by *simp*
next
  case (*unop-i t* $\mathcal{C}$ *uu*)
  thus *?case*
    using *progress-unop-testop-i*[*OF unop-i*(*2,1*)]
    by *fastforce*
next
  case (*unop-f t* $\mathcal{C}$ *uv*)
  thus *?case*
    using *progress-unop-f*[*OF unop-f*(*2,1,5*)]
    by *fastforce*
next
  case (*binop-i t* $\mathcal{C}$ *uw*)
  thus *?case*
    using *progress-binop-relop-i*[*OF binop-i*(*3,1*)]
    by *fastforce*
next
  case (*binop-f t* $\mathcal{C}$ *ux*)
  thus *?case*
    using *progress-binop-relop-f*[*OF binop-f*(*2,1,5*)]

by *fastforce*
next
  case (*testop t C uy*)
  thus *?case*
    using *progress-unop-testop-i*[*OF testop(2,1)*]
    by *fastforce*
next
  case (*relop-i t C uz*)
  thus *?case*
    using *progress-binop-relop-i*[*OF relop-i(2,1)*]
    by *fastforce*
next
  case (*relop-f t C va*)
  thus *?case*
    using *progress-binop-relop-f*[*OF relop-f(2,1,5)*]
    by *fastforce*
next
  case (*convert t1 t2 sx C*)
  obtain *v* **where** *cs-def*:*cs = [\$ C v] typeof v = t2*
    using *const-typeof const-of-const-list*[*OF - convert(7)*] *e-type-const-list*[*OF convert(7,4)*]
    by *fastforce*
  thus *?case*
  proof (*cases cvt t1 sx v*)
    case *None*
    thus *?thesis*
      using *reduce.intros(1)*[*OF reduce-simple.convert-None*[*OF - None*]] *cs-def*
types-agree-imp-types-agree-insecure
     unfolding *types-agree-def*
     by *fastforce*
  next
    case (*Some a*)
    thus *?thesis*
      using *reduce.intros(1)*[*OF reduce-simple.convert-Some*[*OF - Some*]] *cs-def*
types-agree-imp-types-agree-insecure
     unfolding *types-agree-def*
     by *fastforce*
  qed
next
  case (*reinterpret t1 t2 C*)
  obtain *v* **where** *cs-def*:*cs = [\$ C v] typeof v = t2*
    using *const-typeof const-of-const-list*[*OF - reinterpret(7)*] *e-type-const-list*[*OF reinterpret(7,4)*]
    by *fastforce*
  thus *?case*
   using *reduce.intros(1)*[*OF reduce-simple.reinterpret*[*OF types-agree-imp-types-agree-insecure*]]
    unfolding *types-agree-def*
    by *fastforce*
next

case (*classify t2 t1 C*)

obtain *v* **where** *cs-def*:*cs* = [$ *C v*] *typeof v* = *t2*

  using *const-typeof const-of-const-list*[*OF - classify(7)*] *e-type-const-list*[*OF classify(7,4)*]

  by *fastforce*

thus *?case*

 using *reduce.intros(1)*[*OF reduce-simple.classify*[*OF types-agree-imp-types-agree-insecure*]]

  unfolding *types-agree-def*

  by *fastforce*

next

case (*declassify C t2 t1*)

obtain *v* **where** *cs-def*:*cs* = [$ *C v*] *typeof v* = *t2*

  using *const-typeof const-of-const-list*[*OF - declassify(8)*] *e-type-const-list*[*OF declassify(8,5)*]

  by *fastforce*

thus *?case*

 using *reduce.intros(1)*[*OF reduce-simple.declassify*[*OF types-agree-imp-types-agree-insecure*]]

  unfolding *types-agree-def*

  by *fastforce*

next

case (*unreachable C ts ts′*)

thus *?case*

  using *reduce.intros(1)*[*OF reduce-simple.unreachable*] *progress-L0*[*OF - unreachable(4)*]

  by *fastforce*

next

case (*nop C*)

thus *?case*

  using *reduce.intros(1)*[*OF reduce-simple.nop*] *progress-L0*[*OF - nop(4)*]

  by *fastforce*

next

case (*drop C t*)

obtain *v* **where** *cs* = [$*C v*]

  using *const-of-const-list drop(4) e-type-const-list*[*OF drop(4,1)*]

  by *fastforce*

thus *?case*

  using *reduce.intros(1)*[*OF reduce-simple.drop*] *progress-L0*[*OF - drop(4)*]

  by *fastforce*

next

case (*select sec t C*)

obtain *v1 v2 v3* **where** *cs-def*:*S·C* ⊢ [$ *C v3*] : ([] -> [(*T-i32 sec*)])

                        cs = [$*C v1*, $*C v2*, $ *C v3*]

  using *const-list-split-3*[*OF select(5,2)*] *select(5)*

  unfolding *const-list-def*

  by (*metis list-all-simps(1) e-type-const-unwrap*)

obtain *c3* **where** *c-def*:*v3* = *ConstInt32 sec c3*

  using *cs-def select(5) const-of-i32*[*OF - cs-def(1)*]

  unfolding *const-list-def*

  by *fastforce*

have $\exists a\ s'\ vs'\ es'.\ (\!|s;vs;[\$C\ v1,\ \$C\ v2,\ \$\ C\ ConstInt32\ sec\ c3,\ \$Select\ sec]|\!)$ $a{\rightsquigarrow}\text{-}i\ (\!|s';vs';es'|\!)$
  proof (*cases int-eq c3 0*)
    case *True*
    thus *?thesis*
      using *reduce.intros(1)[OF reduce-simple.select-false]*
      by *fastforce*
  next
    case *False*
    thus *?thesis*
      using *reduce.intros(1)[OF reduce-simple.select-true]*
      by *fastforce*
  qed
  thus *?case*
    using *c-def cs-def*
    by *fastforce*
next
  case (*block tf tn tm C es*)
  show *?case*
    using *reduce-simple.block[OF block(7), of - tn tm - es]*
        e-type-const-list[OF block(7,4)] reduce.intros(1) block(1)
    by *fastforce*
next
  case (*loop tf tn tm C es*)
  show *?case*
    using *reduce-simple.loop[OF loop(7), of - tn tm - es]*
        e-type-const-list[OF loop(7,4)] reduce.intros(1) loop(1)
    by *fastforce*
next
  case (*if-wasm tf tn tm C es1 es2*)
  obtain *c1s c2s* **where** *cs-def*:$\mathcal{S}\cdot\mathcal{C} \vdash c1s : ([]\ \text{->}\ tn)$
                      $\mathcal{S}\cdot\mathcal{C} \vdash c2s : ([]\ \text{->}\ [(\text{T-}i32\ Public)])$
                      const-list c1s
                      const-list c2s
                      cs = c1s @ c2s
    using *e-type-const-list-cons[OF if-wasm(9,6)] e-type-const-list*
    by *fastforce*
  obtain *c* **where** *c-def*: *c2s* = [$\$\ C\ (ConstInt32\ Public\ c)$]
    using *const-of-i32 cs-def*
    by *fastforce*
  have $\exists a\ s'\ vs'\ es'.\ (\!|s;vs;[\$\ C\ (ConstInt32\ Public\ c),\ \$\ If\ tf\ es1\ es2]|\!)\ a{\rightsquigarrow}\text{-}i$ $(\!|s';vs';es'|\!)$
  proof (*cases int-eq c 0*)
    case *True*
    thus *?thesis*
      using *reduce.intros(1)[OF reduce-simple.if-false]*
      by *fastforce*
  next
    case *False*

      thus *?thesis*
        using *reduce.intros(1)[OF reduce-simple.if-true]*
        by *fastforce*
    qed
    thus *?case*
      using *c-def cs-def progress-L0*
      by *fastforce*
next
  case (*br i C ts t1s t2s*)
  thus *?case*
    using *Lfilled.intros(1)[OF br(6), of - [] [$Br i]]*
    by *fastforce*
next
  case (*br-if j C ts*)
  obtain *cs1 cs2* **where** *cs-def*:$\mathcal{S} \cdot \mathcal{C} \vdash cs1 : ([] \to ts)$
                           $\mathcal{S} \cdot \mathcal{C} \vdash cs2 : ([] \to [(T\text{-}i32\ Public)])$
                           const-list cs1
                           const-list cs2
                           cs = cs1 @ cs2
    using *e-type-const-list-cons[OF br-if(6,3)] e-type-const-list*
    by *fastforce*
  obtain *c* **where** *c-def*:*cs2 = [$C ConstInt32 Public c]*
    using *const-of-i32[OF cs-def(4,2)]*
    by *blast*
  have $\exists a\ s'\ vs'\ es'. (\!|s;vs;cs2@(\$* [Br\text{-}if\ j])|\!)\ a \rightsquigarrow\text{-}i\ (\!|s';vs';es'|\!)$
  proof (*cases int-eq c 0*)
    case *True*
    thus *?thesis*
      using *c-def reduce.intros(1)[OF reduce-simple.br-if-false]*
      by *fastforce*
  next
    case *False*
    thus *?thesis*
      using *c-def reduce.intros(1)[OF reduce-simple.br-if-true]*
      by *fastforce*
  qed
  thus *?case*
    using *cs-def(5) progress-L0[OF - cs-def(3), of s vs cs2 @ ($* [Br-if j]) - - - -*
- []]
    by *fastforce*
next
  case (*br-table C ts is i' t1s t2s*)
  obtain *cs1 cs2* **where** *cs-def*:$\mathcal{S} \cdot \mathcal{C} \vdash cs1 : ([] \to (t1s\ @\ ts))$
                           $\mathcal{S} \cdot \mathcal{C} \vdash cs2 : ([] \to [(T\text{-}i32\ Public)])$
                           const-list cs1
                           const-list cs2
                           cs = cs1 @ cs2
    using *e-type-const-list-cons[OF br-table(5), of $\mathcal{S}$ $\mathcal{C}$ (t1s @ ts) [(T-i32 Public)]]*
        e-type-const-list[of - $\mathcal{S}$ $\mathcal{C}$ t1s @ ts (t1s @ ts) @ [(T-i32 Public)]]

      br-table(2,5)
   unfolding *const-list-def*
   by *fastforce*
  obtain *c* **where** *c-def*:*cs2* $=$ [$\$C$ *ConstInt32 Public c*]
   using *const-of-i32*[*OF cs-def(4,2)*]
   by *blast*
 have $\exists$ *a s' vs' es'.* $(\!|s;vs;[\$C$ *ConstInt32 Public c,* $\$Br$-*table is i'*]$\!|)$ $a \rightsquigarrow$-*i* $(\!|s';vs';es'\!|)$
 proof (*cases* (*nat-of-int c*) $<$ *length is*)
  case *True*
  show *?thesis*
   using *reduce.intros(1)*[*OF reduce-simple.br-table*[*OF True*]]
   by *fastforce*
 next
  case *False*
  hence *length is* $\leq$ *nat-of-int c*
   by *fastforce*
  thus *?thesis*
   using *reduce.intros(1)*[*OF reduce-simple.br-table-length*]
   by *fastforce*
 qed
 thus *?case*
  using *c-def cs-def progress-L0*
  by *fastforce*
next
 case (*return* $\mathcal{C}$ *ts t1s t2s*)
 thus *?case*
  using *Lfilled.intros(1)*[*OF return(5), of* - [] [$\$Return$]]
  by *fastforce*
next
 case (*call* $\mathcal{C}$ *tr j*)
 show *?case*
  using *progress-L0*[*OF reduce.intros(2)*[*of s vs j i*] *call(7), of* []]
  by *fastforce*
next
 case (*call-indirect* $\mathcal{C}$ *tr j t1s t2s*)
 obtain *cs1 cs2* **where** *cs-def*:$\mathcal{S} \cdot \mathcal{C} \vdash cs1 : ([]$-$> t1s)$
                     $\mathcal{S} \cdot \mathcal{C} \vdash cs2 : ([]$ -$> [(T$-*i32 Public*)])
                     const-list cs1
                     const-list cs2
                     cs $=$ *cs1* @ *cs2*
  using *e-type-const-list-cons*[*OF call-indirect(8), of* $\mathcal{S}$ $\mathcal{C}$ *t1s* [(*T-i32 Public*)]]
    e-type-const-list[*of* - $\mathcal{S}$ $\mathcal{C}$ *t1s t1s* @ [(*T-i32 Public*)]]
    call-indirect(5)
  by *fastforce*
 obtain *c* **where** *c-def*:*cs2* $=$ [$\$C$ *ConstInt32 Public c*]
  using *cs-def(2,4) const-of-i32*
  by *fastforce*
 consider
  (*1*) $\exists$ *cl tf. stab s i* (*nat-of-int c*) $=$ *Some cl* $\wedge$ *stypes s i j* $=$ *tf* $\wedge$ *cl-type cl* $=$ *tf*

| *(2)* ∃ *cl. stab s i (nat-of-int c) = Some cl* ∧ *stypes s i j* ≠ *cl-type cl*
| *(3)* *stab s i (nat-of-int c) = None*
  by *(metis option.collapse)*
 hence ∃ *a s' vs' es'.* (|*s;vs;*[*$C ConstInt32 Public c*, *$Call-indirect j*]|) *a⤳-i* (|*s';vs';es'*|)
 proof *(cases)*
  case *1*
  thus *?thesis*
   using *reduce.intros(3)*
   by *blast*
 next
  case *2*
  thus *?thesis*
   using *reduce.intros(4)*
   by *blast*
 next
  case *3*
  thus *?thesis*
   using *reduce.intros(4)*
   by *blast*
 qed
 then show *?case*
  using *c-def cs-def progress-L0*
  by *fastforce*
next
 case *(get-local j C t)*
 obtain *v vj vj'* **where** *v-def*:*v = vs ! j vj = (take j vs)* *vj' = (drop (j+1) vs)*
  by *blast*
 have *j-def*:*j < length vs*
  using *get-local(1,9)*
  by *simp*
 hence *vj-len*:*length vj = j*
  using *v-def(2)*
  by *fastforce*
 hence *vs = vj @ [v] @ vj'*
  using *v-def id-take-nth-drop j-def*
  by *fastforce*
 thus *?case*
  using *progress-L0*[*OF reduce.intros(8)*[*OF vj-len, of s v vj'*] *get-local(6)*]
  by *fastforce*
next
 case *(set-local j C t)*
 obtain *v vj vj'* **where** *v-def*:*v = vs ! j vj = (take j vs)* *vj' = (drop (j+1) vs)*
  by *blast*
 obtain *v'* **where** *cs-def*: *cs = [$C v']*
  using *const-of-const-list set-local(3,6) e-type-const-list*
  by *fastforce*
 have *j-def*:*j < length vs*
  using *set-local(1,9)*

    by *simp*
  hence *vj-len*:*length vj = j*
    using *v-def(2)*
    by *fastforce*
  hence $vs = vj @ [v] @ vj'$
    using *v-def id-take-nth-drop j-def*
    by *fastforce*
  thus *?case*
    using *reduce.intros(9)[OF vj-len, of s v vj' v' i] cs-def*
    by *fastforce*
next
  case (*tee-local i C t*)
  obtain *v* **where** $cs = [\$C\ v]$
    using *const-of-const-list tee-local(3,6) e-type-const-list*
    by *fastforce*
  thus *?case*
    using *reduce.intros(1)[OF reduce-simple.tee-local] tee-local(6)*
    unfolding *const-list-def*
    by *fastforce*
next
  case (*get-global j C t*)
  thus *?case*
    using *reduce.intros(10)[of s vs j i] progress-L0*
    by *fastforce*
next
  case (*set-global j C t*)
  obtain *v* **where** $cs = [\$C\ v]$
    using *const-of-const-list set-global(4,7) e-type-const-list*
    by *fastforce*
  thus *?case*
    using *reduce.intros(11)[of s i j v - vs]*
    by *fastforce*
next
  case (*load C n sec t a tp-sx off*)
  obtain *c* **where** *c-def*: $cs = [\$C\ ConstInt32\ Public\ c]$
    using *const-of-i32 load(4,7) e-type-const-unwrap*
    unfolding *const-list-def*
    by *fastforce*
  obtain *j* **where** *mem-some*:*smem-ind s i = Some j*
    using *load(1,11)*
    unfolding *smem-ind-def*
    by (*metis map-option-eq-Some option-projr-def*)
  hence *smem-sec*:*snd (s.mem s ! j) = sec*
    using *load(1,11)*
    unfolding *option-projr-def*
    by *simp*
  have $\exists a'\ s'\ vs'\ es'.\ (\!|s;vs;[\$C\ ConstInt32\ Public\ c,\ \$Load\ t\ tp\text{-}sx\ a\ off]|\!)\ a'\rightsquigarrow\text{-}i$
$(\!|s';vs';es'|\!)$
  proof (*cases tp-sx*)

123

case *None*
note *tp-none = None*
show *?thesis*
proof (*cases load* (*fst* ((*mem s*)!*j*)) (*nat-of-int c*) *off* (*t-length t*))
  case *None*
  show *?thesis*
   using *reduce.intros*(*13*)[*OF mem-some - None, of sec vs*] *tp-none smem-sec load*(*2*)
   by *fastforce*
next
  case (*Some a*)
  show *?thesis*
   using *reduce.intros*(*12*)[*OF mem-some - Some, of sec vs*] *tp-none smem-sec load*(*2*)
   by *fastforce*
qed
next
  case (*Some a*)
  obtain *tp sx* **where** *tp-some*:*tp-sx = Some* (*tp, sx*)
   using *Some*
   by *fastforce*
  show *?thesis*
  proof (*cases load-packed sx* (*fst* ((*mem s*)!*j*)) (*nat-of-int c*) *off* (*tp-length tp*) (*t-length t*))
   case *None*
   show *?thesis*
    using *reduce.intros*(*15*)[*OF mem-some - None, of sec vs Public*] *tp-some smem-sec load*(*2*)
   by *fastforce*
  next
   case (*Some a*)
   show *?thesis*
    using *reduce.intros*(*14*)[*OF mem-some - Some, of sec vs Public*] *tp-some smem-sec load*(*2*)
   by *fastforce*
  qed
qed
then show *?case*
  using *c-def progress-L0*
  by *fastforce*
next
 case (*store $\mathcal{C}$ n sec t a tp off*)
 obtain *cs′ v* **where** *cs-def*:$\mathcal{S} \cdot \mathcal{C} \vdash [cs′] : ([] \text{ -> } [(\text{T-i32 Public})])$
                $\mathcal{S} \cdot \mathcal{C} \vdash [\$ \text{ } C \text{ } v] : ([] \text{ -> } [t])$
                $cs = [cs′, \$ \text{ } C \text{ } v]$
  using *const-list-split-2*[*OF store*(*7,4*)] *e-type-const-unwrap*
  unfolding *const-list-def*
  by *fastforce*
 have *t-def*:*typeof v = t*

124

using *cs-def*(*2*) *b-e-type-value*[*OF unlift-b-e*[*of S C* [*C v*] ([] *-> [t]*)]]
  by *fastforce*
obtain *j* **where** *mem-some*:*smem-ind s i = Some j*
  using *store*(*1*,*11*)
  unfolding *smem-ind-def*
  by (*metis map-option-eq-Some option-projr-def*)
hence *smem-sec*:*snd* (*s.mem s ! j*) *= sec*
  using *store*(*1*,*11*)
  unfolding *option-projr-def*
  by *simp*
obtain *c* **where** *c-def*:*cs' = $C ConstInt32 Public c*
  using *const-of-i32*[*OF - cs-def*(*1*)] *cs-def*(*3*) *store*(*7*)
  unfolding *const-list-def*
  by *fastforce*
have ∃ *a' s' vs' es'*. (|*s*;*vs*;[*$C ConstInt32 Public c*, *$C v*, *$Store t tp a off*]|)
a'⤳-i (|*s'*;*vs'*;*es'*|)
proof (*cases tp*)
  case *None*
  note *tp-none = None*
  show *?thesis*
  proof (*cases store* (*fst* (*s.mem s ! j*)) (*nat-of-int c*) *off* (*bits v*) (*t-length t*))
    case *None*
    show *?thesis*
      using *reduce.intros*(*17*)[*OF - mem-some - None, of sec vs Public*] *t-def*
tp-none smem-sec types-agree-imp-types-agree-insecure store(*2*)
      unfolding *types-agree-def*
      by *fastforce*
  next
    case (*Some a*)
    show *?thesis*
      using *reduce.intros*(*16*)[*OF - mem-some - Some, of sec vs Public*] *t-def*
tp-none smem-sec types-agree-imp-types-agree-insecure store(*2*)
      unfolding *types-agree-def*
      by *fastforce*
  qed
next
  case (*Some a*)
  note *tp-some = Some*
  show *?thesis*
  proof (*cases store-packed* (*fst* (*s.mem s ! j*)) (*nat-of-int c*) *off* (*bits v*) (*tp-length*
a))
    case *None*
    show *?thesis*
      using *reduce.intros*(*19*)[*OF - mem-some - None, of t sec vs Public*] *t-def*
tp-some smem-sec types-agree-imp-types-agree-insecure store(*2*)
      unfolding *types-agree-def*
      by *fastforce*
  next
    case (*Some a*)

125

show *?thesis*
    using *reduce.intros(18)[OF - mem-some - Some, of t sec vs Public] t-def*
tp-some smem-sec types-agree-imp-types-agree-insecure store(2)
    unfolding *types-agree-def*
    by *fastforce*
  qed
 qed
 then show *?case*
  using *c-def cs-def progress-L0*
  by *fastforce*
next
 case (*current-memory* $\mathcal{C}$ *n sec*)
 obtain *j* **where** *mem-some*:*smem-ind s i = Some j*
  using *current-memory(1,9)*
  unfolding *smem-ind-def*
  by (*metis map-option-eq-Some option-projr-def*)
 thus *?case*
 proof (*cases s.mem s ! j*)
  case (*Pair a b*)
  thus *?thesis*
   using *progress-L0[OF reduce.intros(20)[OF mem-some] current-memory(5),*
of - - - vs []]
   by *fastforce*
 qed
next
 case (*grow-memory* $\mathcal{C}$ *n sec*)
 obtain *c* **where** *c-def*:*cs = [$C ConstInt32 Public c]*
  using *const-of-i32 grow-memory(2,5)*
  by *fastforce*
 obtain *j* **where** *mem-some*:*smem-ind s i = Some j*
  using *grow-memory(1,9)*
  unfolding *smem-ind-def*
  by (*metis map-option-eq-Some option-projr-def*)
 hence *smem-sec*:*snd (s.mem s ! j) = sec*
  using *grow-memory(1,9)*
  unfolding *option-projr-def*
  by *simp*
 show *?case*
  using *reduce.intros(22)[OF mem-some, of - sec] c-def smem-sec*
  by (*cases s.mem s ! j*) *fastforce*
next
 case (*empty* $\mathcal{C}$)
 thus *?case*
  unfolding *const-list-def*
  by *simp*
next
 case (*composition* $\mathcal{C}$ *es t1s t2s e t3s*)
 consider (*1*) ¬ *const-list* ($* es) | (*2*) *const-list* ($* es*) ¬ *const-list* ($*[e])
  using *composition(9)*

unfolding *const-list-def*
by *fastforce*
thus *?case*
proof (*cases*)
  case *1*
  have ($\bigwedge$*lholed*. ¬ *Lfilled 0 lholed* [*$Return*] (*cs* @ (*$∗ es*)))
     ($\bigwedge$*i lholed*. ¬ *Lfilled 0 lholed* [*$Br i*] (*cs* @ (*$∗ es*)))
  proof *safe*
    fix *lholed*
    assume *Lfilled 0 lholed* [*$Return*] (*cs* @ (*$∗ es*))
    hence ∃ *lholed′*. *Lfilled 0 lholed′* [*$Return*] (*cs* @ (*$∗ es* @ [*e*]))
    proof (*cases rule*: *Lfilled.cases*)
      case (*L0 vs es′*)
      thus *?thesis*
        using *Lfilled.intros*(*1*)[*of vs - es′*@ (*$∗*[*e*]) [*$Return*]]
        by (*metis append.assoc map-append*)
    qed *simp*
    thus *False*
      using *composition*(*6*)
      by *simp*
  next
    fix *i lholed*
    assume *Lfilled 0 lholed* [*$Br i*] (*cs* @ (*$∗ es*))
    hence ∃ *lholed′*. *Lfilled 0 lholed′* [*$Br i*] (*cs* @ (*$∗ es* @ [*e*]))
    proof (*cases rule*: *Lfilled.cases*)
      case (*L0 vs es′*)
      thus *?thesis*
        using *Lfilled.intros*(*1*)[*of vs - es′*@ (*$∗*[*e*]) [*$Br i*]]
        by (*metis append.assoc map-append*)
    qed *simp*
    thus *False*
      using *composition*(*7*)
      by *simp*
  qed
  thus *?thesis*
     using *composition*(*2*)[*OF composition*(*5*) - - *composition*(*8*) *1 composition*(*10,11,12*)] *progress-L0*[*of s vs* (*cs* @ (*$∗ es*)) - *i* - - - [] *$∗*[*e*]]
    unfolding *const-list-def*
    by *fastforce*
next
  case *2*
  hence *const-list* (*cs*@(*$∗ es*))
    using *composition*(*8*)
    unfolding *const-list-def*
    by *simp*
  moreover
  have $\mathcal{S} \cdot \mathcal{C} \vdash$ (*cs*@(*$∗ es*)) : ([] -> *t2s*)
  using *composition*(*5*) *e-typing-s-typing.intros*(*1*)[*OF composition*(*1*)] *e-type-comp-conc*
    by *fastforce*

ultimately
  show *?thesis*
    using *composition(4)[of (cs@($∗ es))] 2(2) composition(6,7) composition(10−)*
  by *fastforce*
  qed
next
  case (*weakening C es t1s t2s ts*)
  obtain *cs1 cs2* **where** *cs-def:S·C ⊢ cs1 : ([] -> ts)*
                    S·C ⊢ cs2 : ([] -> t1s)
                    cs = cs1 @ cs2
                    const-list cs1
                    const-list cs2
    using *e-type-const-list-cons[OF weakening(6,3)] e-type-const-list[of - S C ts ts @ t1s]*
    by *fastforce*
  have ($\bigwedge$ *lholed. ¬ Lfilled 0 lholed [$Return] (cs2 @ ($∗ es)))*
      ($\bigwedge$ *i lholed. ¬ Lfilled 0 lholed [$Br i] (cs2 @ ($∗ es)))*
  proof *safe*
    fix *lholed*
    assume *Lfilled 0 lholed [$Return] (cs2 @ ($∗ es))*
    hence *∃ lholed'. Lfilled 0 lholed' [$Return] (cs1 @ cs2 @ ($∗ es))*
    proof (*cases rule: Lfilled.cases*)
      case (*L0 vs es'*)
      thus *?thesis*
        using *Lfilled.intros(1)[of cs1 @ vs - es' [$Return]] cs-def(4)*
        unfolding *const-list-def*
        by *fastforce*
    qed *simp*
    thus *False*
      using *weakening(4) cs-def(3)*
      by *simp*
  next
    fix *i lholed*
    assume *Lfilled 0 lholed [$Br i] (cs2 @ ($∗ es))*
    hence *∃ lholed'. Lfilled 0 lholed' [$Br i] (cs1 @ cs2 @ ($∗ es))*
    proof (*cases rule: Lfilled.cases*)
      case (*L0 vs es'*)
      thus *?thesis*
        using *Lfilled.intros(1)[of cs1 @ vs - es' [$Br i]] cs-def(4)*
        unfolding *const-list-def*
        by *fastforce*
    qed *simp*
    thus *False*
      using *weakening(5) cs-def(3)*
      by *simp*
  qed
  hence *∃ a s' vs' es'. (|s;vs;cs2@($∗es)|) a⤳-i (|s';vs';es'|)*
    using *weakening(2)[OF cs-def(2) - - cs-def(5) weakening(7)] weakening(8−)*

128

  by *fastforce*
 thus *?case*
  using *progress-L0 [OF - cs-def(4), of s vs cs2 @ ($\ast$ es) - i - - - []] cs-def(3)*
  by *fastforce*
qed

lemma *progress-e*:
 assumes $\mathcal{S} \cdot tr \cdot None \Vdash$-$i$ $vs;cs$-$es : ts'$
   $\bigwedge k\ lholed.\ \neg(Lfilled\ k\ lholed\ [\$Return]\ cs\text{-}es)$
   $\bigwedge i\ k\ lholed.\ (Lfilled\ k\ lholed\ [\$Br\ (i)]\ cs\text{-}es) \implies i < k$
   $cs\text{-}es \neq [Trap]$
   $\neg\ const\text{-}list\ (cs\text{-}es)$
   store-typing s $\mathcal{S}$
 shows $\exists\ a\ s'\ vs'\ es'.\ (\!|s;vs;cs\text{-}es|\!)\ a \rightsquigarrow$-$i\ (\!|s';vs';es'|\!)$
proof $-$
 fix $\mathcal{C}$ *cs es ts-c*
 have *prems1*:
  $\mathcal{S} \cdot \mathcal{C} \vdash es : (ts\text{-}c \rightarrow ts') \implies$
  $\mathcal{S} \cdot \mathcal{C} \vdash cs\text{-}es : ([] \rightarrow ts') \implies$
  $cs\text{-}es = cs @ es \implies$
  const-list cs $\implies$
  $\mathcal{S} \cdot \mathcal{C} \vdash cs : ([] \rightarrow ts\text{-}c) \implies$
  $(\bigwedge k\ lholed.\ \neg(Lfilled\ k\ lholed\ [\$Return]\ cs\text{-}es)) \implies$
  $(\bigwedge i\ k\ lholed.\ (Lfilled\ k\ lholed\ [\$Br\ (i)]\ cs\text{-}es) \implies i < k) \implies$
  $cs\text{-}es \neq [Trap] \implies$
  $\neg\ const\text{-}list\ (cs\text{-}es) \implies$
  store-typing s $\mathcal{S} \implies$
  $i < length\ (s\text{-}inst\ \mathcal{S}) \implies$
  $length\ (local\ \mathcal{C}) = length\ (vs) \implies$
  $option\text{-}projr\ (memory\ \mathcal{C}) = map\text{-}option\ (\lambda j.\ snd\ ((mem\ s)!j))\ (smem\text{-}ind\ s$
$i) \implies$
   $\exists\ a\ s'\ vs'\ cs\text{-}es'.\ (\!|s;vs;cs\text{-}es|\!)\ a \rightsquigarrow$-$i\ (\!|s';vs';cs\text{-}es'|\!)$
 and *prems2*:
  $\mathcal{S} \cdot tr \cdot None \Vdash$-$i\ vs;cs\text{-}es : ts' \implies$
  $(\bigwedge k\ lholed.\ \neg(Lfilled\ k\ lholed\ [\$Return]\ cs\text{-}es)) \implies$
  $(\bigwedge i\ k\ lholed.\ (Lfilled\ k\ lholed\ [\$Br\ (i)]\ cs\text{-}es) \implies i < k) \implies$
  $cs\text{-}es \neq [Trap] \implies$
  $\neg\ const\text{-}list\ (cs\text{-}es) \implies$
  store-typing s $\mathcal{S} \implies$
   $\exists\ a\ s'\ vs'\ cs\text{-}es'.\ (\!|s;vs;cs\text{-}es|\!)\ a \rightsquigarrow$-$i\ (\!|s';vs';cs\text{-}es'|\!)$
 proof (*induction arbitrary*: *vs ts-c ts' i cs-es cs rule*: *e-typing-s-typing.inducts*)
  case (*1 $\mathcal{C}$ b-es tf $\mathcal{S}$*)
  hence $\mathcal{C} \vdash b\text{-}es : (ts\text{-}c \rightarrow ts')$
   using *e-type-comp-conc1 [of $\mathcal{S}$ $\mathcal{C}$ cs ($\ast$ b-es) [] ts'] unlift-b-e*
   by (*metis e-type-const-conv-vs typing-map-typeof*)
  then show *?case*
   using *progress-b-e [OF - 1(5) - - 1(4)] 1(3,4,9) list-all-append 1*
   unfolding *const-list-def*
   by *fastforce*

next
　case (*2 S C es t1s t2s e t3s*)
　show *?case*
　proof (*cases const-list es*)
　　case *True*
　　hence *const-list* (*cs@es*)
　　　using *2(7)*
　　　unfolding *const-list-def*
　　　by *simp*
　　moreover
　　have *∃ts″. (S·C ⊢ (cs @ es) : ([] -> ts″))*
　　　using *2(5,6)*
　　　by (*metis append.assoc e-type-comp-conc1*)
　　ultimately
　　show *?thesis*
　　　using *2(4)[OF 2(5) - - - 2(9,10,11,12,13,14,15), of (cs@es)] 2(6,16)*
　　　by *fastforce*
　next
　　case *False*
　　hence *¬const-list* (*cs@es*)
　　　unfolding *const-list-def*
　　　by *simp*
　　moreover
　　have *∃ts″. (S·C ⊢ (cs @ es) : ([] -> ts″))*
　　　using *2(5,6)*
　　　by (*metis append.assoc e-type-comp-conc1*)
　　moreover
　　have ⋀*k lholed. ¬ Lfilled k lholed* [*$Return*] (*cs @ es*)
　　proof −
　　　{
　　　　assume *∃k lholed. Lfilled k lholed* [*$Return*] (*cs @ es*)
　　　　then obtain *k lholed* **where** *local-assms:Lfilled k lholed* [*$Return*] (*cs @ es*)
　　　　　by *blast*
　　　　hence *∃lholed′. Lfilled k lholed′* [*$Return*] (*cs @ es @ [e]*)
　　　　proof (*cases rule: Lfilled.cases*)
　　　　　case (*L0 vs es′*)
　　　　　obtain *lholed′* **where** *lholed′ = LBase vs (es′@[e])*
　　　　　　by *blast*
　　　　　thus *?thesis*
　　　　　　using *L0*
　　　　　　by (*metis Lfilled.intros(1) append.assoc*)
　　　　next
　　　　　case (*LN vs ts es′ l es″ k lfilledk*)
　　　　　obtain *lholed′* **where** *lholed′ = LRec vs ts es′ l (es″@[e])*
　　　　　　by *blast*
　　　　　thus *?thesis*
　　　　　　using *LN*
　　　　　　by (*metis Lfilled.intros(2) append.assoc*)

130

```
    qed
    hence False
      using 2(6,9)
      by blast
  }
  thus ⋀k lholed. ¬ Lfilled k lholed [$Return] (cs @ es)
    by blast
qed
moreover
have ⋀i k lholed.  Lfilled k lholed [$Br i] (cs @ es) ⟹ i < k
proof −
  {
    assume ∃i k lholed.  Lfilled k lholed [$Br i] (cs @ es) ∧ ¬(i < k)
     then obtain i k lholed where local-assms:Lfilled k lholed [$Br i] (cs @
es) ¬(i < k)
       by blast
    hence ∃ lholed'.  Lfilled k lholed' [$Br i] (cs @ es @ [e]) ∧ ¬(i < k)
    proof (cases rule: Lfilled.cases)
      case (L0 vs es')
      obtain lholed' where lholed' = LBase vs (es'@[e])
        by blast
      thus ?thesis
        using L0 local-assms(2)
        by (metis Lfilled.intros(1) append.assoc)
    next
      case (LN vs ts es' l es'' k lfilledk)
      obtain lholed' where lholed' = LRec vs ts es' l (es''@[e])
        by blast
      thus ?thesis
        using LN local-assms(2)
        by (metis Lfilled.intros(2) append.assoc)
    qed
    hence False
      using 2(6,10)
      by blast
  }
  thus ⋀i k lholed.  Lfilled k lholed [$Br i] (cs @ es) ⟹ i < k
    by blast
qed
moreover
note preds = calculation
show ?thesis
proof (cases cs @ es = [Trap])
  case True
  thus ?thesis
    using reduce-simple.trap[of - (LBase [] [e])]
          Lfilled.intros(1)[of [] LBase [] [e] [e] cs @ es]
          reduce.intros(1) 2(6,11)
    unfolding const-list-def
```

131

        by (*metis append.assoc append-Nil list.pred-inject(1)*)
     next
      case *False*
      thus *?thesis*
       using *2(3)[OF - - 2(7,8) - - - - 2(13,14,15)] preds 2(6,16)*
          progress-L0[of s vs (cs @ es) - - - - - [] [e]]
       unfolding *const-list-def*
       by (*metis append.assoc append-Nil list.pred-inject(1)*)
     qed
   qed
 next
  case (*3 S C es t1s t2s ts*)
  thus *?case*
   by *fastforce*
 next
  case (*4 S C tf*)
  have *cs-es-def*:*Lfilled 0 (LBase cs []) [Trap] cs-es*
   using *Lfilled.intros(1)[OF 4(3), of - [] [Trap]] 4(2)*
   by *fastforce*
  thus *?case*
   using *reduce-simple.trap[OF 4(7) cs-es-def] reduce.intros(1)*
   by *blast*
 next
 case (*5 S C ts j vls es n*)
  consider (*1*) ($\bigwedge$*k lholed.* $\neg$ *Lfilled k lholed* [$*Return*] *es*)
         ($\bigwedge$*k lholed i.* (*Lfilled k lholed* [$*Br i*] *es*) $\implies$ *i < k*)
         es $\neq$ [*Trap*]
         $\neg$ *const-list es*
      | (*2*) $\exists$*k lholed. Lfilled k lholed* [$*Return*] *es*
      | (*3*) *const-list es* $\vee$ (*es* = [*Trap*])
      | (*4*) $\exists$*k lholed i.* (*Lfilled k lholed* [$*Br i*] *es*) $\wedge$ *i* $\geq$ *k*
   using *not-le-imp-less*
   by *blast*
  thus *?case*
  proof (*cases*)
   case *1*
   obtain *s′ vs″ a a′* **where** *temp1*:(|*s;vls;es*|) *a′*$\rightsquigarrow$*- j* (|*s′;vs″;a*|)
    using *5(3)[OF 1(1) - 1(3,4) 5(12)] 1(2)*
    by *fastforce*
   show *?thesis*
    using *reduce.intros(24)[OF temp1, of vs] progress-L0*[**where** *?cs = cs*, *OF*
- 5(6)] 5(5)
    by *fastforce*
  next
   case *2*
   then obtain *k lholed* **where** *local-assms*:(*Lfilled k lholed* [$*Return*] *es*)
    by *blast*
  then obtain *lholed′ vs′ C′* **where** *lholed′-def*:(*Lfilled k lholed′* (*vs′*@[$*Return*])
es)

$$\mathcal{S} \cdot \mathcal{C}' \vdash vs' : ([] \rightarrow ts)$$
$$const\text{-}list\ vs'$$

        using *progress-LN-return*[*OF local-assms, of* $\mathcal{S}$ - *ts ts*] *s-type-unfold*[*OF*
5(1)]
        by *fastforce*
      hence *temp1*:$\exists\, a.\ (\![Local\ n\ j\ vls\ es]\!)\ a \rightsquigarrow (\!|vs'|\!)$
      using *reduce-simple.return*[*OF lholed'-def(3)*]
          e-type-const-list[*OF lholed'-def(3,2)*] *5(2)*
      by *fastforce*
      show *?thesis*
      using *temp1 progress-L0*[*OF reduce.intros(1) 5(6)*] *5(5)*
      by *fastforce*
    next
      case *3*
      then consider *(1) const-list es | (2) es = [Trap]*
      by *blast*
      hence *temp1*:$\exists\, a.\ (\!|s;vs;[Local\ n\ j\ vls\ es]|\!)\ a \rightsquigarrow\text{-}\ i\ (\!|s;vs;es|\!)$
      proof *(cases)*
        case *1*
        have *length es = length ts*
          using *s-type-unfold*[*OF 5(1)*] *e-type-const-list*[*OF 1*]
          by *fastforce*
        thus *?thesis*
          using *reduce-simple.local-const*[*OF 1*] *reduce.intros(1) 5(2)*
          by *fastforce*
        next
          case *2*
          thus *?thesis*
           using *reduce-simple.local-trap reduce.intros(1)*
           by *fastforce*
      qed
      thus *?thesis*
      using *progress-L0*[**where** *?cs = cs, OF - 5(6)*] *5(5)*
      by *fastforce*
    next
      case *4*
      then obtain *k' lholed' i'* **where** *temp1*:*Lfilled k' lholed'* [$Br (k'+i')] *es*
        using *le-Suc-ex*
        by *blast*
      obtain $\mathcal{C}'$ **where** *c-def*:$\mathcal{C}' = ((s\text{-}inst\ \mathcal{S})!j)(\!|trust\text{-}t := trust\text{-}t\ \mathcal{C}, local := (local$
$((s\text{-}inst\ \mathcal{S})!j)) @ (map\ typeof\ vls), return := Some\ ts|\!)$
        by *blast*
      hence *es-def*:$\mathcal{S} \cdot \mathcal{C}' \vdash es : ([] \rightarrow ts)\ j < length\ (s\text{-}inst\ \mathcal{S})$
        using *5(1) s-type-unfold*
        by *fastforce+*
      hence *length (label* $\mathcal{C}'$*) = 0*
        using *c-def store-local-label-empty 5(12)*
        by *fastforce*
      thus *?thesis*

```
        using progress-LN1[OF temp1 es-def(1)]
        by linarith
    qed
  next
    case (6 C tr S cl tf)
    obtain ts″ where ts″-def:S·C ⊢ cs : ([] -> ts″) S·C ⊢ [Callcl cl] : (ts″ -> ts′)
      using 6(3,4) e-type-comp-conc1
      by fastforce
    obtain ts-c tr′ t1s t2s where cl-def:(ts″ = ts-c @ t1s)
                                        (ts′ = ts-c @ t2s)
                                        cl-type cl = (tr′,(t1s -> t2s))
      using e-type-callcl[OF ts″-def(2)]
      by fastforce
    obtain vs1 vs2 where vs-def:S·C ⊢ vs1 : ([] -> ts-c)
                               S·C ⊢ vs2 : (ts-c -> ts-c @ t1s)
                               cs = vs1 @ vs2
                               const-list vs1
                               const-list vs2
      using e-type-const-list-cons[OF 6(5)] ts″-def(1) cl-def(1)
      by fastforce
    have l:(length vs2) = (length t1s)
      using e-type-const-list vs-def(2,5)
      by fastforce
    show ?case
    proof (cases cl)
      case (Func-native x11 x12 x13 x14)
      hence func-native-def:cl = Func-native x11 (tr′,(t1s -> t2s)) x13 x14
        using cl-def(3)
        unfolding cl-type-def
        by simp
      have ∃ a a′. ⦇s;vs;vs2 @ [Callcl cl]⦈ a′⤳- i ⦇s;vs;a⦈
      using reduce.intros(5)[OF func-native-def] e-type-const-conv-vs[OF vs-def(5)]
```
l
```
        unfolding n-zeros-def
        by fastforce
      thus ?thesis
        using progress-L0 vs-def(3,4) 6(4)
        by fastforce
    next
      case (Func-host x21 x22)
      hence func-host-def:cl = Func-host (tr′,(t1s -> t2s)) x22
        using cl-def(3)
        unfolding cl-type-def
        by simp
      obtain vcs where vcs-def:vs2 = $$* vcs
        using e-type-const-conv-vs[OF vs-def(5)]
        by blast
      fix hs
      have ∃ s′ a a′. ⦇s;vs;vs2 @ [Callcl cl]⦈ a′⤳- i ⦇s′;vs;a⦈
```

134

proof (*cases host-apply s* (*t1s -> t2s*) *x22 vcs hs*)
  case *None*
  thus *?thesis*
    using *reduce.intros(7)[OF func-host-def] l vcs-def*
    by *fastforce*
next
  case (*Some a*)
  then obtain *s′ vcs′* **where** *ha-def:host-apply s* (*t1s -> t2s*) *x22 vcs hs =*
Some (*s′, vcs′*)
    by (*metis surj-pair*)
  have *list-all2 types-agree t1s vcs*
    using *e-typing-imp-list-types-agree vs-def(2,4) vcs-def*
    by *simp*
  thus *?thesis*
    using *reduce.intros(6)[OF func-host-def - - - - ha-def] l vcs-def*
        host-apply-respect-type[OF - ha-def]
    by *fastforce*
qed
thus *?thesis*
  using *vs-def(3,4) 6(4) progress-L0*
  by *fastforce*
qed
next
  case (*7 S C e0s ts t2s es n*)
  consider (*1*) ($\bigwedge k$ *lholed.* $\neg$ *Lfilled k lholed* [*$Return*] *es*)
         ($\bigwedge k$ *lholed i.* (*Lfilled k lholed* [*$Br i*] *es*) $\implies i < k$)
         es $\neq$ [*Trap*]
         $\neg$ *const-list es*
     | (*2*) $\exists k$ *lholed. Lfilled k lholed* [*$Return*] *es*
     | (*3*) *const-list es* $\vee$ (*es =* [*Trap*])
     | (*4*) $\exists k$ *lholed i.* (*Lfilled k lholed* [*$Br i*] *es*) $\wedge i = k$
     | (*5*) $\exists k$ *lholed i.* (*Lfilled k lholed* [*$Br i*] *es*) $\wedge i > k$
  using *linorder-neqE-nat*
  by *blast*
thus *?case*
proof (*cases*)
  case *1*
  have *temp1:es =* [] @ *es const-list* []
    unfolding *const-list-def*
    by *auto*
  have *temp2:S·C*(|*label :=* [*ts*] @ *label C*|) $\vdash$ [] : ([] *->* [])
    using *b-e-typing.empty e-typing-s-typing.intros(1)*
    by *fastforce*
  have $\exists s′ vs′ a a′.$ (|*s;vs;es*|) $a′\rightsquigarrow$- i (|*s′;vs′;a*|)
    using *7(5)[OF 7(2), of* [] [], *OF temp1 temp2 1(1) - 1(3,4) 7(14,15)]*
        1(2) 7(16,17)
    unfolding *const-list-def*
    by *fastforce*
  then obtain *s′ vs′ a* **where** *red-def:*$\exists a′.$ (|*s;vs;es*|) $a′\rightsquigarrow$- i (|*s′;vs′;a*|)

135

by *blast*

have *temp4*:$\bigwedge$*es. Lfilled 0* (*LBase* [] []) *es es*

  using *Lfilled.intros(1)*[*of* [] (*LBase* [] []) []]

  unfolding *const-list-def*

  by *fastforce*

 hence *temp5*:*Lfilled 1* (*LRec cs n e0s* (*LBase* [] []) []) *es* (*cs*@[*Label n e0s es*])

   using *Lfilled.intros(2)*[*of cs* (*LRec cs n e0s* (*LBase* [] []) []) *n e0s* (*LBase* [] []) [] *0 es es*] *7(8)*

   unfolding *const-list-def*

   by *fastforce*

  have *temp6*:*Lfilled 1* (*LRec cs n e0s* (*LBase* [] []) []) *a* (*cs*@[*Label n e0s a*])

    using *temp4 Lfilled.intros(2)*[*of cs* (*LRec cs n e0s* (*LBase* [] []) []) *n e0s* (*LBase* [] []) [] *0 a a*] *7(8)*

   unfolding *const-list-def*

   by *fastforce*

  show *?thesis*

   using *reduce.intros(23)*[*OF - temp5 temp6*] *7(7) red-def*

   by *fastforce*

 next

  case *2*

  then obtain *k lholed* **where** (*Lfilled k lholed* [$*Return*] *es*)

   by *blast*

  hence (*Lfilled* (*k+1*) (*LRec cs n e0s lholed* []) [$*Return*] (*cs*@[*Label n e0s es*]))

   using *Lfilled.intros(2) 7(8)*

   by *fastforce*

  thus *?thesis*

   using *7(10)*[*of k+1*] *7(7)*

  by *fastforce*

 next

  case *3*

  hence *temp1*:$\exists$ *a.* (|*s;vs;*[*Label n e0s es*]|) *a$\leadsto$- i* (|*s;vs;es*|)

   using *reduce-simple.label-const reduce-simple.label-trap reduce.intros(1)*

   by *fastforce*

  show *?thesis*

   using *progress-L0*[*OF - 7(8)*] *7(7) temp1*

   by *fastforce*

 next

  case *4*

  then obtain *k lholed* **where** *lholed-def*:(*Lfilled k lholed* [$*Br* (*k+0*)] *es*)

   by *fastforce*

  then obtain *lholed′ vs′ C′* **where** *lholed′-def*:(*Lfilled k lholed′* (*vs′*@[$*Br* (*k*)]) *es*)

$$\mathcal{S} \cdot \mathcal{C}' \vdash vs' : (\text{[} \text{]} \rightarrow ts)$$
$$const\text{-}list\ vs'$$

   using *progress-LN*[*OF lholed-def 7(2), of ts*]

   by *fastforce*

  have $\exists$ *es′ a.* (|[*Label n e0s es*]|) *a$\leadsto$* (|*vs′*@*e0s*|)

136

        using *reduce-simple.br*[*OF lholed′-def*(*3*) *- lholed′-def*(*1*)] *7*(*3*)
             e-type-const-list[*OF lholed′-def*(*3,2*)]
        by *fastforce*
      hence $\exists$ *es′ a.* $(\!|s;vs;[Label\ n\ e0s\ es]|\!)\ a \rightsquigarrow$- *i* $(\!|s;vs;es′|\!)$
        using *reduce.intros*(*1*)
        by *fastforce*
      thus *?thesis*
        using *progress-L0 7*(*7,8*)
        by *fastforce*
    next
      case *5*
      then obtain *i k lholed* **where** *lholed-def*:(*Lfilled k lholed* [$\$$*Br i*] *es*) *i > k*
        using *less-imp-add-positive*
        by *blast*
      have *k1-def*:*Lfilled* (*k+1*) (*LRec cs n e0s lholed* []) [$\$$*Br i*] *cs-es*
        using *7*(*7*) *Lfilled.intros*(*2*)[*OF 7*(*8*) *- lholed-def*(*1*)*, of - n e0s* []]
        by *fastforce*
      thus *?thesis*
        using *7*(*11*)[*OF k1-def*] *lholed-def*(*2*)
        by *simp*
    qed
  next
    case (*8 i S tvs vs C rs es ts*)
    have *length* (*local C*) *= length vs*
      using *8*(*2,3*) *store-local-label-empty*[*OF 8*(*1,11*)]
      by *fastforce*
    moreover have *option-projr* (*memory C*) *= map-option* (λ*j. snd* (*s.mem s* !
j)) (*smem-ind s i*)
      using *store-typing-imp-mem-agree-inst*[*OF 8*(*11,1*)] *8*(*3*)
      by *simp*
    ultimately show *?case*
      using *8*(*6*)[*OF 8*(*4*) *- - - 8*(*7,8,9,10,11,1*)]
          e-typing-s-typing.intros(*1*)[*OF b-e-typing.empty*[*of C*]]
      unfolding *const-list-def*
      by *fastforce*
  qed
  show *?thesis*
    using *prems2*[*OF assms*]
    by *fastforce*
qed

lemma *progress-e1*:
  assumes $S \cdot tr \cdot None \Vdash$-*i vs;es* : *ts*
  shows $\neg$(*Lfilled k lholed* [$\$$*Return*] *es*)
proof −
  {
    assume $\exists$ *k lholed.* (*Lfilled k lholed* [$\$$*Return*] *es*)
    then obtain *k lholed* **where** *local-assms*:(*Lfilled k lholed* [$\$$*Return*] *es*)
      by *blast*

    obtain $\mathcal{C}$ **where** *c-def*:$i < length$ (*s-inst* $\mathcal{S}$)
                    $\mathcal{C} = ((\textit{s-inst}\ \mathcal{S})!i)(\!|\textit{trust-t} := tr,\ local := (local\ ((\textit{s-inst}\ \mathcal{S})!i))\ @$
(*map typeof vs*), *return* := *None*$|\!)$
                  ($\mathcal{S}\cdot\mathcal{C} \vdash es : (\![]\ \text{->}\ ts)$)
    using *assms s-type-unfold*
    by *fastforce*
  have $\exists\ rs.\ return\ \mathcal{C} = Some\ rs$
    using *local-assms c-def*(*3*)
  proof (*induction* [\$*Return*] *es arbitrary*: $\mathcal{C}$ *ts rule*: *Lfilled.induct*)
    case (*L0 vs lholed es$'$*)
    thus *?case*
    using *e-type-comp-conc2*[*OF L0*(*3*)] *unlift-b-e*[*of* $\mathcal{S}$ $\mathcal{C}$ [*Return*]] *b-e-type-return*
      by *fastforce*
    next
    case (*LN vs lholed tls es$'$ l es$''$ k lfilledk*)
    thus *?case*
      using *e-type-comp-conc2*[*OF LN*(*5*)] *e-type-label*[*of* $\mathcal{S}$ $\mathcal{C}$ *tls es$'$ lfilledk*]
      by *fastforce*
  qed
  hence *False*
    using *c-def*(*2*)
    by *fastforce*
 }
 thus $\bigwedge\ k\ lholed.\ \neg$(*Lfilled k lholed* [\$*Return*] *es*)
  by *blast*
qed

lemma *progress-e2*:
 assumes $\mathcal{S}\cdot tr\cdot None \Vdash\text{-}i\ vs;es : ts$
      store-typing s $\mathcal{S}$
 shows (*Lfilled k lholed* [\$*Br* (*j*)] *es*) $\Longrightarrow j < k$
proof $-$
 {
  assume ($\exists\ i\ k\ lholed.$ (*Lfilled k lholed* [\$*Br* (*i*)] *es*) $\wedge\ i \geq k$)
  then obtain *j k lholed* **where** *local-assms*:(*Lfilled k lholed* [\$*Br* (*k+j*)] *es*)
    by (*metis le-iff-add*)
  obtain $\mathcal{C}$ **where** *c-def*:$i < length$ (*s-inst* $\mathcal{S}$)
                    $\mathcal{C} = ((\textit{s-inst}\ \mathcal{S})!i)(\!|\textit{trust-t} := tr,\ local := (local\ ((\textit{s-inst}\ \mathcal{S})!i))\ @$
(*map typeof vs*), *return* := *None*$|\!)$
                  ($\mathcal{S}\cdot\mathcal{C} \vdash es : (\![]\ \text{->}\ ts)$)
    using *assms s-type-unfold*
    by *fastforce*
  have $j < length$ (*label* $\mathcal{C}$)
    using *progress-LN1*[*OF local-assms c-def*(*3*)]
    by $-$
  hence *False*
    using *store-local-label-empty*(*1*)[*OF c-def*(*1*) *assms*(*2*)] *c-def*(*2*)
    by *fastforce*
 }

thus ($\bigwedge$ *j k lholed.* (*Lfilled k lholed* [$Br (j)] *es*) $\implies$ *j < k*)
   by *fastforce*
qed

lemma *progress-e3*:
  assumes $\mathcal{S} \cdot tr \cdot None \Vdash\text{-}i$ *vs*;*cs-es* : *ts*$'$
      cs-es $\neq$ [*Trap*]
      $\neg$ *const-list* (*cs-es*)
      store-typing s $\mathcal{S}$
  shows $\exists$ *a s$'$ vs$'$ es$'$.* (|*s*;*vs*;*cs-es*|) *a*$\rightsquigarrow$-*i* (|*s$'$*;*vs$'$*;*es$'$*|)
  using *assms progress-e progress-e1 progress-e2*
  by *fastforce*

end

7   Soundness Theorems

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

theorem *preservation*:
  assumes $\vdash\text{-}i$ *s*;*vs*;*es* : (*tr*,*ts*)
      (|*s*;*vs*;*es*|) *a*$\rightsquigarrow$-*i* (|*s$'$*;*vs$'$*;*es$'$*|)
  shows $\vdash\text{-}i$ *s$'$*;*vs$'$*;*es$'$* : (*tr*,*ts*)
proof −
  obtain $\mathcal{S}$ **where** *store-typing s* $\mathcal{S}$ $\mathcal{S} \cdot tr \cdot None \Vdash\text{-}i$ *vs*;*es* : *ts*
    using *assms*(*1*) *config-typing.simps*
    by *blast*
  hence *store-typing s$'$* $\mathcal{S}$ $\mathcal{S} \cdot tr \cdot None \Vdash\text{-}i$ *vs$'$*;*es$'$* : *ts*
    using *assms*(*2*) *store-preserved types-preserved-e*
    by *simp-all*
  thus *?thesis*
    using *config-typing.intros*
    by *blast*
qed

theorem *progress*:
  assumes $\vdash\text{-}i$ *s*;*vs*;*es* : (*tr*,*ts*)
  shows *const-list es* $\vee$ *es* = [*Trap*] $\vee$ ($\exists$ *a s$'$ vs$'$ es$'$.* (|*s*;*vs*;*es*|) *a*$\rightsquigarrow$-*i* (|*s$'$*;*vs$'$*;*es$'$*|))
proof −
  obtain $\mathcal{S}$ **where** *store-typing s* $\mathcal{S}$ $\mathcal{S} \cdot tr \cdot None \Vdash\text{-}i$ *vs*;*es* : *ts*
    using *assms config-typing.simps*
    by *blast*
  thus *?thesis*
    using *progress-e3*
    by *blast*
qed

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*
  using *assms*
  by (*cases x*; *cases y*; *simp*)

lemma *ct-eq-flip*: *ct-eq$^{-1\,-1}$ = ct-eq*
  using *ct-eq-commute*

by *fastforce*

lemma *exists-secret*: $\exists\, t.\ is\text{-}secret\text{-}t\ t$
  unfolding *t-sec-def*
  by (*auto split*: *t.splits*)

lemma *ct-eq-common-tsome*: $ct\text{-}eq\ x\ y = (\exists\, t.\ ct\text{-}eq\ x\ (TSome\ t) \land ct\text{-}eq\ (TSome\ t)\ y)$
  by (*cases x*; *cases y*; (*simp add*: *exists-secret*))

lemma *ct-list-eq-commute*:
  assumes *ct-list-eq xs ys*
  shows *ct-list-eq ys xs*
  using *assms ct-eq-commute List.List.list.rel-flip ct-eq-flip*
  unfolding *ct-list-eq-def*
  by *fastforce*

lemma *ct-list-eq-refl*: *ct-list-eq xs xs*
  unfolding *ct-list-eq-def*
  by (*metis can-secret-ct.simps*(*3*) *ct.simps*(*1*) *ct-eq.elims*(*3*) *list.rel-refl*)

lemma *ct-list-eq-length*:
  assumes *ct-list-eq xs ys*
  shows *length xs = length ys*
  using *assms list-all2-lengthD*
  unfolding *ct-list-eq-def*
  by *blast*

lemma *ct-list-eq-concat*:
  assumes *ct-list-eq xs ys*
      ct-list-eq xs′ ys′
  shows *ct-list-eq* (*xs@xs′*) (*ys@ys′*)
  using *assms*
  unfolding *ct-list-eq-def*
  by (*simp add*: *list-all2-appendI*)

lemma *ct-list-eq-ts-conv-eq*:
  ct-list-eq (*to-ct-list ts*) (*to-ct-list ts′*) = (*ts = ts′*)
  unfolding *ct-list-eq-def to-ct-list-def*
      list-all2-map1 list-all2-map2
      ct-eq.simps(*1*)
  by (*simp add*: *list-all2-eq*)

lemma *ct-list-eq-exists*: $\exists\, ys.\ ct\text{-}list\text{-}eq\ xs\ (to\text{-}ct\text{-}list\ ys)$
proof (*induction xs*)
  case *Nil*
  thus *?case*
    unfolding *ct-list-eq-def to-ct-list-def*
    by (*simp*)

next
  case (*Cons a xs*)
  thus *?case*
    unfolding *ct-list-eq-def to-ct-list-def*
    apply (*cases a*)
    apply (*metis ct-eq.simps(6) list.simps(9) list-all2-Cons1*)
    apply (*metis (full-types) can-secret-ct.simps(1) ct-eq.simps(4) ct-eq-commute*
exists-secret list.rel-inject(2) list.simps(9))
   apply (*metis (full-types) ct-list-eq-def ct-list-eq-refl list.rel-inject(2) list.simps(9)*)
    done
qed

lemma *ct-list-eq-common-tsome-list*:
  ct-list-eq xs ys = (∃ zs. ct-list-eq xs (to-ct-list zs) ∧ ct-list-eq (to-ct-list zs) ys)
proof (*induction ys arbitrary: xs*)
  case *Nil*
  thus *?case*
    unfolding *ct-list-eq-def to-ct-list-def*
    by *simp*
next
  case (*Cons a ys*)
  show *?case*
  proof (*safe*)
    assume *assms*:*ct-list-eq xs (a # ys)*
    then obtain *x' xs'* **where** *xs-def*:*xs = x'#xs'*
      by (*meson ct-list-eq-def list-all2-Cons2*)
    then obtain *zs* **where** *zs-def*:*ct-eq x' a*
                          ct-list-eq xs' (to-ct-list zs) ∧ ct-list-eq (to-ct-list zs) ys
      using *Cons*[*of xs'*] *assms list-all2-Cons*
      unfolding *ct-list-eq-def*
      by *fastforce*
    obtain *z* **where** *ct-eq x' (TSome z) ct-eq (TSome z) a*
      using *ct-eq-common-tsome*[*of x' a*] *zs-def(1)*
      by *fastforce*
    hence *ct-list-eq (x'#xs') (to-ct-list (z#zs)) ∧ ct-list-eq (to-ct-list (z#zs)) (a*
 # ys)
      using *zs-def(2) list-all2-Cons*
      unfolding *ct-list-eq-def to-ct-list-def*
      by *simp*
    thus ∃ *zs. ct-list-eq xs (to-ct-list zs) ∧ ct-list-eq (to-ct-list zs) (a # ys)*
      using *xs-def*
      by *fastforce*
  next
    fix *zs*
    assume *assms*:*ct-list-eq xs (to-ct-list zs) ct-list-eq (to-ct-list zs) (a # ys)*
    then obtain *x' xs' z' zs'* **where** *xs = x'#xs'*
                      zs = z'#zs'
                      ct-list-eq xs' (to-ct-list zs')
                      ct-list-eq (to-ct-list zs') (ys)

142

using *list-all2-Cons2*
  unfolding *ct-list-eq-def to-ct-list-def list-all2-map1 list-all2-map2*
  by (*metis* (*no-types*, *lifting*))
 thus *ct-list-eq xs* (*a # ys*)
  using *assms Cons ct-list-eq-def to-ct-list-def ct-eq-common-tsome*
  by (*metis list.simps*(*9*) *list-all2-Cons*)
 qed
qed

lemma *ct-list-eq-cons-ct-list*:
 assumes *ct-list-eq* (*to-ct-list as*) (*xs @ ys*)
 shows ∃ *bs bs'. as = bs @ bs'* ∧ *ct-list-eq* (*to-ct-list bs*) *xs* ∧ *ct-list-eq* (*to-ct-list bs'*) *ys*
 using *assms*
proof (*induction xs arbitrary*: *as*)
 case *Nil*
 thus *?case*
  by (*metis append-Nil ct-list-eq-ts-conv-eq list.simps*(*8*) *to-ct-list-def*)
next
 case (*Cons a xs*)
 thus *?case*
  unfolding *ct-list-eq-def to-ct-list-def list-all2-map1*
  by (*meson list-all2-append2*)
qed

lemma *ct-list-eq-cons-ct-list1*:
 assumes *ct-list-eq* (*to-ct-list as*) (*xs @* (*to-ct-list ys*))
 shows ∃ *bs. as = bs @ ys* ∧ *ct-list-eq* (*to-ct-list bs*) *xs*
 using *ct-list-eq-cons-ct-list*[*OF assms*] *ct-list-eq-ts-conv-eq*
 by *fastforce*

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*
 using *assms ct-list-eq-def*
 by (*meson ct-list-eq-common-tsome-list ct-list-eq-commute*)

lemma *ct-list-eq-take*:
 assumes *ct-list-eq xs ys*
 shows *ct-list-eq* (*take n xs*) (*take n ys*)
 using *assms list-all2-takeI*
 unfolding *ct-list-eq-def*
 by *blast*

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

143

using *assms*
unfolding *ct-prefix-def*
by *blast*

lemma *ct-prefixE* [*elim?*]:
  assumes *ct-prefix xs ys*
  obtains *as zs* **where** *ys = as @ zs ct-list-eq as xs*
  using *assms*
  unfolding *ct-prefix-def*
  by *blast*

lemma *ct-prefix-snoc* [*simp*]: *ct-prefix xs (ys @ [y]) = (ct-list-eq xs (ys@[y]) ∨ ct-prefix xs ys)*
proof (*safe*)
  assume *ct-prefix xs (ys @ [y]) ¬ ct-prefix xs ys*
  thus *ct-list-eq xs (ys @ [y])*
    unfolding *ct-prefix-def ct-list-eq-def*
    by (*metis butlast-append butlast-snoc ct-eq-flip list.rel-flip*)
next
  assume *ct-list-eq xs (ys @ [y])*
  thus *ct-prefix xs (ys @ [y])*
    using *ct-list-eq-commute ct-prefixI*
    by *fastforce*
next
  assume *ct-prefix xs ys*
  thus *ct-prefix xs (ys @ [y])*
    using *append-assoc*
    unfolding *ct-prefix-def*
    by *blast*
qed

lemma *ct-prefix-nil*:*ct-prefix [] xs*
                ¬ct-prefix (x # xs) []
  by (*simp-all add*: *ct-prefix-def ct-list-eq-def*)

lemma *Cons-ct-prefix-Cons*[*simp*]: *ct-prefix (x # xs) (y # ys) = ((ct-eq x y) ∧ ct-prefix xs ys)*
proof (*safe*)
  assume *ct-prefix (x # xs) (y # ys)*
  thus *ct-eq x y*
    unfolding *ct-prefix-def ct-list-eq-def*
    by (*metis ct-eq-commute hd-append2 list.sel(1) list.simps(3) list-all2-Cons2*)
next
  assume *ct-prefix (x # xs) (y # ys)*
  thus *ct-prefix xs ys*
    unfolding *ct-prefix-def ct-list-eq-def*
    by (*metis list.rel-distinct(1) list.sel(3) list-all2-Cons2 tl-append2*)
next
  assume *ct-eq x y ct-prefix xs ys*

thus *ct-prefix* (*x* # *xs*) (*y* # *ys*)
   unfolding *ct-prefix-def ct-list-eq-def*
  by (*metis* (*full-types*) *append-Cons ct-list-eq-commute ct-list-eq-def list.rel-inject*(*2*))
qed

lemma *ct-prefix-code* [*code*]:
  ct-prefix [] *xs* = *True*
  ct-prefix (*x* # *xs*) [] = *False*
  ct-prefix (*x* # *xs*) (*y* # *ys*) = ((*ct-eq x y*) ∧ *ct-prefix xs ys*)
  by (*simp-all add*: *ct-prefix-nil*)

lemma *ct-suffix-to-ct-prefix* [*code*]: *ct-suffix xs ys* = *ct-prefix* (*rev xs*) (*rev ys*)
  unfolding *ct-suffix-def ct-prefix-def ct-list-eq-def*
  by (*metis list-all2-rev1 rev-append rev-rev-ident*)

lemma *inj-TSome*: *inj TSome*
  by (*meson ct.inject injI*)

lemma *to-ct-list-append*:
  assumes *to-ct-list ts* = *as*@*bs*
  shows ∃ *as'*. *to-ct-list as'* = *as*
    ∃ *bs'*. *to-ct-list bs'* = *bs*
  using *assms*
proof (*induct as arbitrary*: *ts*)
  fix *ts*
  assume *to-ct-list ts* = [] @ *bs*
  thus ∃ *as'*. *to-ct-list as'* = []
    ∃ *bs'*. *to-ct-list bs'* = *bs*
   unfolding *to-ct-list-def*
   by *auto*
next
  case (*Cons a as*)
  fix *ts*
  assume *local-assms*:*to-ct-list ts* = (*a* # *as*) @ *bs*
  then obtain *t'* *ts'* **where** *ts* = *t'*#*ts'*
   unfolding *to-ct-list-def*
   by *auto*
  thus ∃ *as'*. *to-ct-list as'* = *a* # *as*
    ∃ *as'*. *to-ct-list as'* = *bs*
   using *Cons local-assms*
   unfolding *to-ct-list-def*
   apply *simp-all*
    apply (*metis list.simps*(*9*))
   apply *blast*
   done
qed

lemma *ct-suffixI* [*intro?*]:
  assumes *ys* = *as* @ *zs*

ct-list-eq zs xs
    shows *ct-suffix xs ys*
    using *assms*
    unfolding *ct-suffix-def*
    by *blast*

lemma *ct-suffixE* [*elim?*]:
    assumes *ct-suffix xs ys*
    obtains *as zs* **where** *ys = as @ zs ct-list-eq zs xs*
    using *assms*
    unfolding *ct-suffix-def*
    by *blast*

lemma *ct-suffix-nil*: *ct-suffix* [] *ts*
    unfolding *ct-suffix-def*
    using *ct-list-eq-refl*
    by *auto*

lemma *ct-suffix-refl*: *ct-suffix ts ts*
    unfolding *ct-suffix-def*
    using *ct-list-eq-refl*
    by *auto*

lemma *ct-suffix-length*:
    assumes *ct-suffix ts ts′*
    shows *length ts ≤ length ts′*
    using *assms list-all2-lengthD*
    unfolding *ct-suffix-def ct-list-eq-def*
    by *fastforce*

lemma *ct-suffix-take*:
    assumes *ct-suffix ts ts′*
    shows *ct-suffix* ((*take* (*length ts − n*) *ts*)) ((*take* (*length ts′ − n*) *ts′*))
    using *assms ct-list-eq-take append-eq-conv-conj*
    unfolding *ct-suffix-def*
proof −
    assume ∃ *as bs. ts′ = as @ bs ∧ ct-list-eq bs ts*
    then obtain *ccs* :: *ct list* **and** *ccsa* :: *ct list* **where**
        f1: *ts′ = ccs @ ccsa ∧ ct-list-eq ccsa ts*
        by *moura*
    then have *f2*: *length ccsa = length ts*
        by (*meson ct-list-eq-length*)
    have ⋀*n. ct-list-eq* (*take n ccsa*) (*take n ts*)
        using *f1* **by** (*meson ct-list-eq-take*)
    then show ∃ *cs csa. take* (*length ts′ − n*) *ts′ = cs @ csa ∧ ct-list-eq csa* (*take*
(*length ts − n*) *ts*)
        using *f2 f1* **by** *auto*
qed

lemma *ct-suffix-ts-conv-suffix*:
  ct-suffix (*to-ct-list ts*) (*to-ct-list ts′*) = *suffix ts ts′*
proof *safe*
  assume *ct-suffix* (*to-ct-list ts*) (*to-ct-list ts′*)
  then obtain *as bs* **where** *to-ct-list ts′* = (*to-ct-list as*) @ (*to-ct-list bs*)
                      ct-list-eq (*to-ct-list bs*) (*to-ct-list ts*)
    using *to-ct-list-append*
    unfolding *ct-suffix-def*
    by *metis*
  thus *suffix ts ts′*
    using *ct-list-eq-ts-conv-eq*
    unfolding *ct-suffix-def to-ct-list-def suffix-def*
    by (*metis map-append*)
next
  assume *suffix ts ts′*
  thus *ct-suffix* (*to-ct-list ts*) (*to-ct-list ts′*)
    using *ct-list-eq-ts-conv-eq*
    unfolding *ct-suffix-def to-ct-list-def suffix-def*
    by (*metis map-append*)
qed

lemma *ct-suffix-exists*: ∃ *ts-c*. *ct-suffix x1* (*to-ct-list ts-c*)
  using *ct-list-eq-commute ct-list-eq-exists ct-suffix-def*
  by *fastforce*

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 −
  obtain *as bs* **where** *x2-def*:*x2* = *as* @ *bs ct-list-eq x1 bs*
    using *assms ct-list-eq-commute*
    unfolding *ct-suffix-def*
    by *blast*
  then obtain *ts-as ts-bs* **where** *ct-list-eq as* (*to-ct-list ts-as*)
                        ct-list-eq x1 (*to-ct-list ts-bs*)
                        ct-list-eq (*to-ct-list ts-bs*) *bs*
    using *ct-list-eq-common-tsome-list*[*of x1 bs*] *ct-list-eq-exists*
    by *fastforce*
  thus *?thesis*
    using *x2-def ct-list-eq-commute*
    unfolding *ct-suffix-def to-ct-list-def*
    by (*metis ct-list-eq-def list-all2-appendI map-append*)
qed

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 −

obtain *as bs* **where** *to-ct-list zs = (to-ct-list as) @ (to-ct-list bs)*
                    ct-list-eq (to-ct-list bs) (xs @ ys)
    using *assms to-ct-list-append[of zs]*
    unfolding *ct-suffix-def*
    by *blast*
  thus *?thesis*
    using *assms ct-list-eq-cons-ct-list[of bs xs ys]*
    unfolding *ct-suffix-def*
  by (*metis append.assoc ct-list-eq-commute ct-list-eq-ts-conv-eq map-append to-ct-list-def*)
qed

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)*
  using *assms ct-suffix-cons-ct-list[of xs [y] zs]*
  unfolding *ct-list-eq-def to-ct-list-def*
  by (*simp add: list-all2-map2*)
    (*metis (no-types, lifting) list-all2-Cons1 list-all2-Nil*)

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)*
  using *ct-suffix-cons-ct-list[OF assms] ct-list-eq-ts-conv-eq*
  by *fastforce*

lemma *ct-suffix-cons2*:
  assumes *ct-suffix (xs) (ys@zs)*
        length xs = length zs
  shows *ct-list-eq xs zs*
  using *assms*
   by (*metis append-eq-append-conv ct-list-eq-commute ct-list-eq-def ct-suffix-def
list-all2-lengthD*)

lemma *ct-suffix-imp-ct-list-eq*:
  assumes *ct-suffix xs ys*
  shows *ct-list-eq (drop (length ys − length xs) ys) xs*
  using *assms ct-list-eq-def list-all2-lengthD*
  unfolding *ct-suffix-def*
  by *fastforce*

lemma *ct-suffix-extend-ct-list-eq*:
  assumes *ct-suffix xs ys*
        ct-list-eq xs′ ys′
  shows *ct-suffix (xs@xs′) (ys@ys′)*
  using *assms*
  unfolding *ct-suffix-def ct-list-eq-def*
  by (*meson append.assoc ct-list-eq-commute ct-list-eq-def list-all2-appendI*)

lemma *ct-suffix-extend-any1*:

148

assumes *ct-suffix xs ys*
        length xs < length ys
shows *ct-suffix (TAny#xs) ys*
proof −
  obtain *as bs* **where** *ys-def:ys = as@bs*
                    ct-list-eq bs xs
    using *assms(1) ct-suffix-def*
    by *fastforce*
  hence *length as > 0*
    using *list-all2-lengthD assms(2)*
    unfolding *ct-list-eq-def*
    by *fastforce*
  then obtain *as′ a* **where** *as-def:as = as′@[a]*
    by (*metis append-butlast-last-id length-greater-0-conv*)
  hence *ct-list-eq (a#bs) (TAny#xs)*
    using *ys-def*
     by (*metis can-secret-ct.elims(3) ct.distinct(1,3) ct-eq.elims(3) ct-list-eq-def*
list.rel-inject(2))
  thus *?thesis*
    using *as-def ys-def ct-suffix-def*
    by *fastforce*
qed

lemma *ct-suffix-singleton-any*: *ct-suffix [TAny] [t]*
  using *ct-suffix-extend-ct-list-eq[of [] [] [TAny] [t]] ct-suffix-nil*
  by (*simp add: ct-suffix-extend-any1*)

lemma *ct-suffix-cons-it*: *ct-suffix xs (xs′@xs)*
  using *ct-list-eq-refl ct-suffix-def*
  by *blast*

lemma *ct-suffix-singleton*:
  assumes *length cts > 0*
  shows *ct-suffix [TAny] cts*
proof −
  have $\bigwedge$*c. ct-prefix [TAny] [c]*
    using *ct-suffix-singleton-any ct-suffix-to-ct-prefix* **by** *force*
  then show *?thesis*
   by (*metis (no-types) Suc-leI append-butlast-last-id assms butlast.simps(2) ct-list-eq-commute*
                     ct-prefix-nil(2) ct-prefix-snoc ct-suffix-def impossible-Cons
length-Cons
              list.size(3))
qed

lemma *ct-suffix-less*:
  assumes *ct-suffix (xs@xs′) ys*
  shows *ct-suffix xs′ ys*
  using *assms*
  unfolding *ct-suffix-def*

by (*metis append-eq-appendI ct-list-eq-def list-all2-append2*)

lemma *ct-suffix-unfold-one*: *ct-suffix* (*xs*@[*x*]) (*ys*@[*y*]) = ((*ct-eq x y*) ∧ *ct-suffix xs ys*)
  using *ct-prefix-code*(*3*)
  by (*simp add*: *ct-suffix-to-ct-prefix*)

lemma *ct-suffix-shared*:
  assumes *ct-suffix cts* (*to-ct-list ts*)
       ct-suffix cts′ (*to-ct-list ts*)
  shows *ct-suffix cts cts′* ∨ *ct-suffix cts′ cts*
proof (*cases length cts > length cts′*)
  case *True*
  obtain *as bs* **where** *cts-def*:*ts = as*@*bs*
                           ct-list-eq cts (*to-ct-list bs*)
    using *assms*(*1*) *ct-suffix-def to-ct-list-def*
    by (*metis append-Nil ct-suffix-cons-ct-list*)
  obtain *as′ bs′* **where** *cts′-def*:*ts = as′*@*bs′*
                           ct-list-eq cts′ (*to-ct-list bs′*)
    using *assms*(*2*) *ct-suffix-def to-ct-list-def*
    by (*metis append-Nil ct-suffix-cons-ct-list*)
  obtain *ct1s ct2s* **where** *cts = ct1s*@*ct2s*
               length ct2s = length cts′
    using *True*
    by (*metis add-diff-cancel-right′ append-take-drop-id length-drop less-imp-le-nat nat-le-iff-add*)
  show *?thesis*
  proof −
    obtain *tts* :: *t list ⇒ ct list ⇒ ct list ⇒ t list* **and** *ttsa* :: *t list ⇒ ct list ⇒ ct list ⇒ t list* **where**
      ∀ *x0 x1 x2*. (∃ *v3 v4*. *x0 = v3* @ *v4* ∧ *ct-list-eq x1* (*to-ct-list v4*) ∧ *ct-suffix x2* (*to-ct-list v3*)) = (*x0 = tts x0 x1 x2* @ *ttsa x0 x1 x2* ∧ *ct-list-eq x1* (*to-ct-list* (*ttsa x0 x1 x2*)) ∧ *ct-suffix x2* (*to-ct-list* (*tts x0 x1 x2*)))
      by *moura*
    then have *f1*: *as′* @ *bs′ = tts* (*as′* @ *bs′*) *ct2s ct1s* @ *ttsa* (*as′* @ *bs′*) *ct2s ct1s* ∧ *ct-list-eq ct2s* (*to-ct-list* (*ttsa* (*as′* @ *bs′*) *ct2s ct1s*)) ∧ *ct-suffix ct1s* (*to-ct-list* (*tts* (*as′* @ *bs′*) *ct2s ct1s*))
      using *assms*(*1*) ‹*cts = ct1s* @ *ct2s*› *cts′-def*(*1*) *ct-suffix-cons-ct-list* **by** *force*
    then have *ct-list-eq cts′* (*to-ct-list* (*ttsa* (*as′* @ *bs′*) *ct2s ct1s*))
      by (*metis* ‹*ct-suffix cts′* (*to-ct-list ts*)› ‹*length ct2s = length cts′*› *cts′-def*(*1*) *ct-list-eq-length ct-suffix-cons2 map-append to-ct-list-def*)
    then show *?thesis*
      using *f1* **by** (*metis* ‹*cts = ct1s* @ *ct2s*› *ct-list-eq-shared ct-suffix-def*)
  qed
next
  case *False*
  hence *len*:*length cts′ ≥ length cts*
    by *linarith*
  obtain *as bs* **where** *cts-def*:*ts = as*@*bs*

150

$$ct\text{-}list\text{-}eq\ cts\ (to\text{-}ct\text{-}list\ bs)$$
  using *assms(1) ct-suffix-def to-ct-list-def*
  by (*metis append-Nil ct-suffix-cons-ct-list*)
 obtain *as′ bs′* **where** *cts′-def*:*ts = as′@bs′*
$$ct\text{-}list\text{-}eq\ cts′\ (to\text{-}ct\text{-}list\ bs′)$$
  using *assms(2) ct-suffix-def to-ct-list-def*
  by (*metis append-Nil ct-suffix-cons-ct-list*)
 obtain *ct1s ct2s* **where** *cts′ = ct1s@ct2s*
$$length\ ct2s = length\ cts$$
  using *len*
  by (*metis add-diff-cancel-right′ append-take-drop-id length-drop nat-le-iff-add*)
 show *?thesis*
 proof −
  obtain *tts :: t list ⇒ ct list ⇒ ct list ⇒ t list* **and** *ttsa :: t list ⇒ ct list ⇒ ct list ⇒ t list* **where**
   ∀ *x0 x1 x2*. (∃ *v3 v4*. *x0 = v3 @ v4* ∧ *ct-list-eq x1 (to-ct-list v4)* ∧ *ct-suffix x2 (to-ct-list v3)*) = (*x0 = tts x0 x1 x2 @ ttsa x0 x1 x2* ∧ *ct-list-eq x1 (to-ct-list (ttsa x0 x1 x2))* ∧ *ct-suffix x2 (to-ct-list (tts x0 x1 x2))*)
   by *moura*
  then have *f1*: *as @ bs = tts (as @ bs) ct2s ct1s @ ttsa (as @ bs) ct2s ct1s* ∧ *ct-list-eq ct2s (to-ct-list (ttsa (as @ bs) ct2s ct1s))* ∧ *ct-suffix ct1s (to-ct-list (tts (as @ bs) ct2s ct1s))*
   using *assms(2)* ‹*cts′ = ct1s @ ct2s*› *cts-def(1) ct-suffix-cons-ct-list* **by** *force*
  then have *ct-list-eq cts (to-ct-list (ttsa (as @ bs) ct2s ct1s))*
   by (*metis* ‹*ct-suffix cts (to-ct-list ts)*› ‹*length ct2s = length cts*› *cts-def(1) ct-list-eq-length ct-suffix-cons2 map-append to-ct-list-def*)
  then show *?thesis*
   using *f1* **by** (*metis* ‹*cts′ = ct1s @ ct2s*› *ct-list-eq-shared ct-suffix-def*)
 qed
qed

fun *checker-type-suffix*::*checker-type ⇒ checker-type ⇒ bool* **where**
 checker-type-suffix (Type ts) (Type ts′) = suffix ts ts′
| *checker-type-suffix (Type 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 ⇒ ct list ⇒ 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*])
  using *assms*
  by (*cases ct1*; *cases ct2*) (*auto split*: *if-splits*)

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*])
  using *assms select-return-top-ens-sec-ct*
  by *fastforce*

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)
  using *assms*
  by (*cases ct1*; *cases ct2*) (*auto split*: *if-split-asm*)

lemma *produce-not-bot*:
  assumes *produce a b = c*
        c ≠ Bot
  shows *a ≠ Bot b ≠ Bot*
  using *assms*
  by (*auto split*: *if-split-asm*)

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 −
  {
    assume *a1*: (*if ct-suffix ts′ (map TSome ts) then Type (take (length ts − length*
ts′) ts) else Bot) = c-t
      assume *a2*: *c-t ≠ Bot*
        obtain *ccs* :: *ct list ⇒ ct list ⇒ ct list* **and** *ccsa* :: *ct list ⇒ ct list ⇒ ct list*
where

153

f3: ∀ *cs csa*. ¬ *ct-suffix cs csa* ∨ *csa* = *ccs cs csa* @ *ccsa cs csa* ∧ *ct-list-eq*
(*ccsa cs csa*) *cs*
    using *ct-suffixE* **by** *moura*
  have *f4*: *ct-suffix ts′* (*map TSome ts*)
    using *a2 a1* **by** *metis*
  then have *f5*: *ct-list-eq* (*ccsa ts′* (*map TSome ts*)) *ts′*
    using *f3* **by** *blast*
  have *f6*: *take* (*length* (*map TSome ts*) − *length* (*ccsa ts′* (*map TSome ts*)))
(*map TSome ts*) @ *ccsa ts′* (*map TSome ts*) = *map TSome ts*
    using *f4 f3* **by** (*metis* (*full-types*) *suffixI suffix-take*)
  have ⋀*cs*. *ct-list-eq* (*cs* @ *ccsa ts′* (*map TSome ts*)) (*cs* @ *ts′*)
    using *f5 ct-list-eq-concat ct-list-eq-refl* **by** *blast*
  then have ∃ *tsa*. *ct-list-eq* (*map TSome ts*) (*map TSome tsa* @ *ts′*) ∧ *c-t* =
Type tsa
    using *f6 f5 f4 a1* **by** (*metis* (*no-types*) *ct-list-eq-length length-map take-map*)
 }
 thus *?thesis*
  using *assms to-ct-list-def*
  by *simp*
qed

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 −
 consider (*1*) *ct-suffix ts′ ts*
    | (*2*) ¬*ct-suffix ts′ ts ct-suffix ts ts′*
    | (*3*) ¬*ct-suffix ts′ ts* ¬*ct-suffix ts ts′*
  by *blast*
 thus *?thesis*
 proof (*cases*)
  case *1*
  hence *TopType* (*take* (*length ts* − *length ts′*) *ts*) = *c-t*
   using *assms*
   by *simp*
  thus *?thesis*
   using *assms*(*2*) *1 ct-list-eq-def*
   unfolding *ct-suffix-def*
  by (*metis* (*no-types, lifting*) *append-eq-append-conv append-take-drop-id diff-diff-cancel*
length-drop list-all2-lengthD)
 next
  case *2*
  thus *?thesis*
   using *assms append-eq-append-conv ct-list-eq-commute*
   unfolding *ct-suffix-def*
   by (*metis append.left-neutral ct-suffix-def ct-suffix-length le-antisym*)
 next

case *3*
  thus *?thesis*
    using *assms*
    by *auto*
 qed
qed

lemma *consume-top-leq*:
  assumes *consume* (*TopType ts*) *ts′ = c-t*
        length ts ≤ length ts′
        c-t ≠ Bot
  shows *c-t = TopType* []
  using *assms append-eq-conv-conj*
  by *fastforce*

lemma *consume-type-type*:
  assumes *consume xs cons = (Type t-int)*
  shows ∃ *tn. xs = Type tn*
  using *assms*
  apply (*cases xs*)
    apply *simp-all*
  apply (*metis checker-type.distinct(1) checker-type.distinct(5)*)
  done

lemma *produce-type-type*:
  assumes *produce xs cons = (Type tm)*
  shows ∃ *tn. xs = Type tn*
  apply (*cases xs*; *cases cons*)
  using *assms*
        apply *simp-all*
  done

lemma *consume-weaken-type*:
  assumes *consume* (*Type tn*) *cons = (Type t-int)*
  shows *consume* (*Type* (*ts@tn*)) *cons = (Type* (*ts@t-int*))
proof −
  obtain *ts′* **where** *ct-list-eq* (*to-ct-list tn*) (*to-ct-list ts′ @ cons*) ∧ *Type t-int =*
Type ts′
    using *consume-type*[*OF assms*]
    by *blast*
  have *cond*:*ct-suffix cons* (*to-ct-list tn*)
    using *assms*
    by (*simp, metis checker-type.distinct(5)*)
  hence *res*:*t-int = take* (*length tn − length cons*) *tn*
    using *assms*
    by *simp*
  have *ct-suffix cons* (*to-ct-list* (*ts@tn*))
    using *cond*
    unfolding *to-ct-list-def*

    by (*metis append-assoc ct-suffix-def map-append*)
  moreover
  have *ts@t-int = take* (*length* (*ts@tn*) − *length cons*) (*ts@tn*)
    using *res take-append cond ct-suffix-length to-ct-list-def*
    by *fastforce*
  ultimately
  show *?thesis*
    by *simp*
qed

lemma *produce-weaken-type*:
  assumes *produce* (*Type tn*) *cons* = (*Type tm*)
  shows *produce* (*Type* (*ts@tn*)) *cons* = (*Type* (*ts@tm*))
  using *assms*
  by (*cases cons*, *simp-all*)

lemma *produce-nil*: *produce ts* (*Type* []) = *ts*
  using *to-ct-list-def*
  by (*cases ts*, *simp-all*)

lemma *c-types-agree-id*: *c-types-agree* (*Type ts*) *ts*
  by *simp*

lemma *c-types-agree-top1*: *c-types-agree* (*TopType* []) *ts*
  using *ct-suffix-ts-conv-suffix to-ct-list-def*
  by (*simp add*: *ct-suffix-nil*)

lemma *c-types-agree-top2*:
  assumes *ct-list-eq ts* (*to-ct-list ts″*)
  shows *c-types-agree* (*TopType ts*) (*ts′@ts″*)
  using *assms ct-list-eq-commute ct-suffix-def to-ct-list-def*
  by *auto*

lemma *c-types-agree-imp-ct-list-eq*:
  assumes *c-types-agree* (*TopType cts*) *ts*
  shows ∃ *ts′ ts″*. (*ts* = *ts′@ts″*) ∧ *ct-list-eq cts* (*to-ct-list ts″*)
  using *assms ct-suffix-def to-ct-list-def*
  by (*simp*, *metis ct-list-eq-commute ct-list-eq-ts-conv-eq ct-suffix-ts-conv-suffix suffixE*
        to-ct-list-append(*2*))

lemma *c-types-agree-not-bot-exists*:
  assumes *ts* ≠ *Bot*
  shows ∃ *ts-c*. *c-types-agree ts ts-c*
  using *assms ct-suffix-exists*
  by (*cases ts*, *simp-all*)

lemma *consume-c-types-agree*:
  assumes *consume* (*Type ts*) *cts* = (*Type ts′*)
      c-types-agree ctn ts

156

shows $\exists\, c\text{-}t'.\ consume\ ctn\ cts = c\text{-}t' \land c\text{-}types\text{-}agree\ c\text{-}t'\ ts'$
  using *assms*
proof (*cases ctn*)
  case (*TopType x1*)
  have *1*:*ct-suffix cts* (*to-ct-list ts*)
    using *assms*
    by (*simp, metis checker-type.distinct(5)*)
  hence *ct-suffix cts x1* $\lor$ *ct-suffix x1 cts*
    using *TopType 1 assms(2) ct-suffix-shared*
    by *simp*
  thus *?thesis*
  proof (*rule disjE*)
    assume *local-assms*:*ct-suffix cts x1*
    hence *2*:*consume* (*TopType x1*) *cts = TopType* (*take* (*length x1* $-$ *length cts*) *x1*)
      by *simp*
    have (*take* (*length ts* $-$ *length cts*) *ts*) = *ts'*
      using *assms 1*
      by *simp*
    hence *c-types-agree* (*TopType* (*take* (*length x1* $-$ *length cts*) *x1*)) *ts'*
      using *2 assms local-assms TopType ct-suffix-take*
      by (*simp, metis length-map take-map to-ct-list-def*)
    thus *?thesis*
      using *2 TopType*
      by *simp*
  next
    assume *local-assms*:*ct-suffix x1 cts*
    hence *3*:*consume* (*TopType x1*) *cts = TopType* []
      by (*simp add*: *ct-suffix-length*)
    thus *?thesis*
      using *TopType c-types-agree-top1*
      by *blast*
  qed
qed *simp-all*

lemma *type-update-type*:
  assumes *type-update* (*Type ts*) (*to-ct-list cons*) *prods = ts'*
        $ts' \neq Bot$
      shows $(ts' = prods \land (\exists\, ts\text{-}c.\ prods = (TopType\ ts\text{-}c)))$
              $\lor\ (\exists\, ts\text{-}a\ ts\text{-}b.\ prods = Type\ ts\text{-}a \land ts = ts\text{-}b@cons \land ts' = Type$
$(ts\text{-}b@ts\text{-}a))$
  using *assms*
  apply (*cases prods*)
    apply *simp-all*
   apply (*metis* (*full-types*) *produce.simps(3) produce.simps(7)*)
  using *ct-suffix-ts-conv-suffix suffix-take to-ct-list-def*
  apply *fastforce*
  done

lemma *type-update-empty*: *type-update ts cons* (*Type* []) = *consume ts cons*
  using *produce-nil*
  by *simp*

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′*
       ∃ *t-ag′*. *t-ag* = *t-ag′*@*prods* ∧ *c-types-agree* (*TopType ts*) (*t-ag′*@*cons*)
proof −
  consider (*1*) *ct-suffix* (*to-ct-list cons*) *ts*
       | (*2*) ¬*ct-suffix* (*to-ct-list cons*) *ts ct-suffix ts* (*to-ct-list cons*)
       | (*3*) ¬*ct-suffix* (*to-ct-list cons*) *ts* ¬*ct-suffix ts* (*to-ct-list cons*)
    by *blast*
  hence *ct-suffix* (*to-ct-list prods*) *ts′* ∧ (∃ *t-ag′*. *t-ag* = *t-ag′*@*prods* ∧ *c-types-agree*
(*TopType ts*) (*t-ag′*@*cons*))
  proof (*cases*)
    case *1*
    hence *ts′* = (*take* (*length ts* − *length cons*) *ts*) @ *to-ct-list prods*
      using *assms*(*1*) *to-ct-list-def*
      by *simp*
    moreover
    then obtain *t-ag′* **where** *t-ag* = *t-ag′* @ *prods*
                    ct-suffix (*take* (*length ts* − *length cons*) *ts*) (*to-ct-list t-ag′*)
      using *assms*(*2*) *ct-suffix-cons-ct-list1*
      unfolding *c-types-agree.simps*
      by *blast*
    moreover
    hence *ct-suffix ts* (*to-ct-list* (*t-ag′*@*cons*))
      using *1 ct-suffix-imp-ct-list-eq ct-suffix-extend-ct-list-eq to-ct-list-def*
      by *fastforce*
    ultimately
    show *?thesis*
      using *c-types-agree.simps*(*2*) *ct-list-eq-ts-conv-eq ct-suffix-def*
      by *auto*
  next
    case *2*
    thus *?thesis*
      using *assms*
        by (*metis append.assoc c-types-agree.simps*(*2*) *checker-type.inject*(*1*) *con-sume.simps*(*2*)
             ct-list-eq-ts-conv-eq ct-suffix-cons-ct-list ct-suffix-def map-append
             produce.simps(*1*) *to-ct-list-def type-update.simps*)
  next
    case *3*
    thus *?thesis*
      using *assms*
      by *simp*

qed
  thus *ct-suffix (to-ct-list prods) ts′*
      $\exists\,t\text{-}ag'.\ t\text{-}ag = t\text{-}ag'@prods \land c\text{-}types\text{-}agree\ (TopType\ ts)\ (t\text{-}ag'@cons)$
  by *simp-all*
qed

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]*
  using *assms*
  by *simp*

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 −
  have *1*:*type-update (TopType cts) [TSome (T-i32 sec)] (TopType [sec-ct sec]) =*
tm
    using *assms(1,2)*
    by *simp*
  hence *ct-suffix cts [TSome (T-i32 sec)]* $\lor$ *ct-suffix [TSome (T-i32 sec)] cts*
    using *assms(3)*
    by *(metis consume.simps(2) produce.simps(7) type-update.simps)*
  thus *ct-list-eq cts [TSome (T-i32 sec)]*
    using *assms(2,3) ct-suffix-imp-ct-list-eq*
     by *(metis One-nat-def Suc-length-conv ct-list-eq-commute diff-Suc-1 drop-0*
list.size(3))
  show *tm = TopType [sec-ct sec]*
    using *1 assms(3) consume-top-leq*
    by *(metis One-nat-def assms(2) diff-Suc-1 diff-is-0-eq length-Cons list.size(3)*
        produce.simps(4,7) type-update.simps)
qed

lemma *type-update-select-length2*:
  assumes *type-update-select sec (TopType cts) = tm*
      length cts = 2
      $tm \neq Bot$
  shows $\exists\,t1\ t2.\ cts = [t1,\ t2] \land ct\text{-}eq\ t2\ (TSome\ (T\text{-}i32\ sec)) \land ct\text{-}eq\ t1\ (sec\text{-}ct$
sec) $\land$ *tm = TopType [ens-sec-ct sec t1]*
proof −
  obtain *x y* **where** *cts-def*:*cts = [x,y]*
    using *assms(2) List.length-Suc-conv[of cts Suc 0]*
    by *(metis length-0-conv length-Suc-conv numeral-2-eq-2)*

moreover
hence *type-update* (*TopType* [*x*,*y*]) [*sec-ct sec*, *TSome* (*T-i32 sec*)] (*TopType*
[*ens-sec-ct sec x*]) = *tm*
using *assms cts-def*
by (*simp split*: *if-splits*)
moreover
hence *ct-disj-is*:*ct-suffix* [*x*,*y*] [*sec-ct sec*, *TSome* (*T-i32 sec*)] ∨ *ct-suffix* [*sec-ct
sec*, *TSome* (*T-i32 sec*)] [*x*,*y*]
using *assms*(*3*)
by (*metis consume.simps*(*2*) *produce.simps*(*7*) *type-update.simps*)
have *ct-list-eq* [*sec-ct sec*, *TSome* (*T-i32 sec*)] ([*x*,*y*])
proof (*cases ct-suffix* [*x*,*y*] [*sec-ct sec*, *TSome* (*T-i32 sec*)])
  case *True*
  thus *?thesis*
    by (*metis append-Nil ct-suffixI ct-suffix-cons2 length-Cons*)
next
  case *False*
  thus *?thesis*
    using *ct-disj-is*
    by (*metis append-Nil ct-suffix-cons2 length-Cons*)
qed
ultimately
show *?thesis*
  by (*metis assms*(*3*) *consume-top-leq ct-eq-commute ct-list-eq-def ct-suffix-length
ct-suffix-refl length-Cons list.simps*(*11*) *produce.simps*(*4*,*7*) *type-update.simps*)
qed

lemma *type-update-select-length3*:
  assumes *type-update-select sec* (*TopType cts*) = (*TopType ctm*)
        length cts ≥ *3*
  shows ∃ *cts′ ct1 ct2 ct3*. *cts* = *cts′*@[*ct1*, *ct2*, *ct3*] ∧ *ct-eq ct3* (*TSome* (*T-i32*
sec))
        ∧ *ct-eq ct1* (*sec-ct sec*) ∧ *ct-eq ct2* (*sec-ct sec*)
proof −
  obtain *cts′ cts″* **where** *cts-def*:*cts* = *cts′*@ *cts″ length cts″* = *3*
    using *assms*(*2*)
    by (*metis append-take-drop-id diff-diff-cancel length-drop*)
  then obtain *ct1 cts″2* **where** *cts″* = *ct1*#*cts″2 length cts″2* = *Suc* (*Suc 0*)
    using *List.length-Suc-conv*[*of cts′ Suc* (*Suc 0*)]
    by (*metis length-Suc-conv numeral-3-eq-3*)
  then obtain *ct2 ct3* **where** *cts″* = [*ct1*,*ct2*,*ct3*]
    using *List.length-Suc-conv*[*of cts″2 Suc 0*]
    by (*metis length-0-conv length-Suc-conv*)
  hence *cts-def2*:*cts* = *cts′*@ [*ct1*,*ct2*,*ct3*]
    using *cts-def*
    by *simp*
  obtain *nat* **where** *length cts* = *Suc* (*Suc* (*Suc nat*))
    using *assms*(*2*)
    by (*simp add*: *cts-def*)

160

hence (*type-update* (*TopType cts*) [*sec-ct sec, sec-ct sec, TSome* (*T-i32 sec*)]
            (*select-return-top sec cts* (*cts*!(*length cts−2*)) (*cts*!(*length cts−3*))))) =
TopType ctm
   using *assms*
   by *simp*
 then obtain *c-mid* **where** *consume* (*TopType cts*) [*sec-ct sec, sec-ct sec, TSome*
(*T-i32 sec*)] = *TopType c-mid*
   by (*metis consume.simps(2) produce.simps(6) type-update.simps*)
 hence *ct-suff*:*ct-suffix* [*sec-ct sec, sec-ct sec, TSome* (*T-i32 sec*)] (*cts′*@ [*ct1,ct2,ct3*])
   using *assms(2) consume-top-geq cts-def2*
  by (*metis checker-type.distinct(3) ct-suffix-def length-Cons list.size(3) numeral-3-eq-3*)
  hence *ct-eq ct3* (*TSome* (*T-i32 sec*))
     ct-eq ct2 (*sec-ct sec*)
     ct-eq ct1 (*sec-ct sec*)
   using *ct-suffix-def ct-list-eq-def*
  by (*simp, metis append-eq-append-conv length-Cons list-all2-Cons list-all2-lengthD*)+
 thus *?thesis*
   using *cts-def2*
   by *blast*
qed

lemma *type-update-select-type-length3*:
 assumes *type-update-select sec* (*Type tn*) = (*Type tm*)
 shows ∃ *t ts′. tn = ts′*@[*t, t,* (*T-i32 sec*)]
proof −
 have *tn-cond*:(*length tn ≥ 3* ∧ (*tn*!(*length tn−2*)) = (*tn*!(*length tn−3*)))
   using *assms*
   by (*simp, metis checker-type.distinct(5)*)
 hence *tm-def*:*consume* (*Type tn*) [*TAny, TSome* (*T-i32 sec*)] = *Type tm*
   using *assms*
   by (*simp split: if-split-asm*)
 obtain *tn′ tn″* **where** *tn-split*:*tn = tn′*@*tn″*
                   length tn″ = 3
   using *assms tn-cond*
   by (*metis append-take-drop-id diff-diff-cancel length-drop*)
 then obtain *t1 tn″2* **where** *tn″ = t1*#*tn″2 length tn″2 = Suc* (*Suc 0*)
   by (*metis length-Suc-conv numeral-3-eq-3*)
 then obtain *t2 t3* **where** *tn″-def*:*tn″ = [t1,t2,t3]*
   using *List.length-Suc-conv*[*of tn″2 Suc 0*]
   by (*metis length-0-conv length-Suc-conv*)
 hence *tn-def*:*tn = tn′*@ [*t1,t2,t3*]
   using *tn-split*
   by *simp*
 hence *t1-t2-eq*:*t1 = t2*
   using *tn-cond*
   by (*metis* (*no-types, lifting*) *Suc-diff-Suc Suc-eq-plus1-left Suc-lessD tn″-def*
                 add-diff-cancel-right′ diff-is-0-eq length-append neq0-conv
                 not-less-eq-eq nth-Cons-0 nth-Cons-numeral
                 nth-append numeral-2-eq-2 numeral-3-eq-3 numeral-One

tn-split(2)

$\qquad$ *zero-less-diff*)
  have *ct-suffix* [*TAny*, *TSome* (*T-i32 sec*)] (*to-ct-list* (*tn'* @ [*t1*, *t2*, *t3*]))
    using *tn-def tm-def*
    by (*simp*, *metis checker-type.distinct(5)*)
  hence *t3* = (*T-i32 sec*)
    using *ct-suffix-unfold-one*[*of* [*TAny*] *TSome* (*T-i32 sec*) *to-ct-list* (*tn'* @ [*t1*,
t2]) *TSome t3*]
        ct-eq.simps(1)
    unfolding *to-ct-list-def*
    by *simp*
  thus *?thesis*
    using *t1-t2-eq tn-def*
    by *simp*
qed

lemma *select-return-top-exists*:
  assumes *select-return-top sec cts c1 c2* = *ctm*
        ctm ≠ *Bot*
  shows ∃ *xs*. *ctm* = *TopType xs*
  using *assms*
  by (*cases c1*; *cases c2*) (*auto split*: *if-splits*)

lemma *type-update-select-top-exists*:
  assumes *type-update-select sec xs* = (*TopType tm*)
  shows ∃ *tn*. *xs* = *TopType tn*
  using *assms*
proof (*cases xs*)
  case (*Type x2*)
  thus *?thesis*
    using *assms*
   by (*simp*, *metis checker-type.distinct(1) checker-type.distinct(3) consume.simps(1)*)
qed *simp-all*

lemma *select-return-top-ct-eq*:
  assumes *select-return-top sec cts c1 c2* = *TopType ctm*
        length cts ≥ *3*
        c-types-agree (*TopType ctm*) *cm*
  shows ∃ *c' cm'*. *cm* = *cm'*@[*c'*]
              ∧ *ct-suffix* (*take* (*length cts* − *3*) *cts*) (*to-ct-list cm'*)
              ∧ *ct-eq c1* (*TSome c'*)
              ∧ *ct-eq c2* (*TSome c'*)
  using *assms ct-suffix-cons-ct-list-single*[*of take* (*length cts* − *3*) *cts*]
  by (*cases c1*; *cases c2*) (*fastforce split*: *if-splits*)+

lemma *ens-sec-ct-imp-ct-eq*:
  assumes *ct-eq* (*ens-sec-ct sec ct*) *ct'*
  shows *ct-eq ct ct'*

using *assms*
apply (*cases sec*; *cases ct*)
apply *simp-all*
using *can-secret-ct.simps(2) ct-eq.elims(3)*
by *blast*

lemma *ens-sec-ct-imp-ct-eq-sec*:
  assumes *ct-eq ct ct′*
          sec = Secret ⟶ can-secret-ct ct′
  shows *ct-eq (ens-sec-ct sec ct) ct′*
  using *assms*
  apply (*cases sec*; *cases ct*)
  by *simp-all*

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′′*
  using *assms*
  by (*cases ct1*) *auto*

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)*
  using *assms*
  apply (*cases ct*)
  apply *simp-all*
  using *can-secret-ct.simps(2) ct-eq.elims(3)* **apply** *blast*
  apply (*metis (full-types) can-secret-ct.simps(3) checker-type.simps(6) sec-ct.elims
select-return-top.simps(2) select-return-top-sec*)
  by (*metis can-secret-ct.simps(1) ct-eq.simps(2) ct-eq.simps(7) ct-eq-commute
sec-ct.elims*)

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)*
  using *assms*
  apply (*cases ct1*; *cases ct2*)
  using *ct-suffix-cons-ct-list-single* **apply** *force*
  using *ct-suffix-cons-ct-list-single* **apply** *fastforce*
  apply (*metis c-types-agree.simps(2) ct-eq.simps(4) ct-eq-TSecret-imp-is-secret-t
ct-suffix-cons-ct-list-single last-snoc select-return-top.simps(3)*)
  using *ct-suffix-cons-ct-list-single* **apply** *force*
  using *ct-suffix-cons-ct-list-single* **apply** *fastforce*
  apply (*metis (no-types, lifting) c-types-agree.elims(2) can-secret-ct.simps(1)*

163

checker-type.distinct(*1*) *checker-type.inject*(*1*) *ct-eq.simps*(*1*) *ct-suffix-cons-ct-list-single*
select-return-top.simps(*6*) *snoc-eq-iff-butlast*)
  apply (*metis c-types-agree.simps*(*2*) *ct-eq.simps*(*4*) *ct-eq-TSecret-imp-is-secret-t*
ct-suffix-cons-ct-list-single last-snoc select-return-top.simps(*1*))
  apply (*metis c-types-agree.simps*(*2*) *ct-eq.simps*(*4*) *ct-eq-TSecret-imp-is-secret-t*
ct-suffix-cons-ct-list-single last-snoc select-return-top.simps(*5*))
  by (*metis c-types-agree.simps*(*2*) *can-secret-ct.simps*(*1*) *ct-eq.simps*(*4*) *ct-eq-TSecret-imp-is-secret-t*
ct-suffix-cons-ct-list-single ens-sec-ct.simps(*4*) *last-snoc select-return-top.simps*(*7*))

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*)

$$\vee \ (\textit{is-int-t t1} \wedge \textit{is-int-t t2} \wedge (\textit{t-length}$$

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*
  using *assms*
proof(*induction ils*)
  case (*Cons a ils*)
  thus *?case*
    apply (*simp,safe*)
      apply (*metis not-less option.distinct*(*1*))+
    done
qed *simp*

lemma *same-lab-conv-list-all*:
  assumes *same-lab ils ls* = *Some ts*

shows *list-all* ($\lambda i.\ i <$ *length ls* $\land$ *ls!i = ts) ils*
using *assms*
proof (*induction rule*: *same-lab.induct*)
case (*2 i is lab-c*)
  thus *?case*
    using *same-lab-h-conv-list-all*
  by (*metis* (*mono-tags*, *lifting*) *list-all-simps(1) not-less option.distinct(1) same-lab.simps(2)*)
qed *simp*

lemma *list-all-conv-same-lab-h*:
  assumes *list-all* ($\lambda i.\ i <$ *length ls* $\land$ *ls!i = ts) ils*
  shows *same-lab-h ils ls ts = Some ts*
  using *assms*
  by (*induction ils*, *simp-all*)

lemma *list-all-conv-same-lab*:
  assumes *list-all* ($\lambda i.\ i <$ *length ls* $\land$*ls!i = ts*) (*is@[i]*)
  shows *same-lab* (*is@[i]*) *ls = Some ts*
  using *assms*
proof (*induction* (*is@[i]*))
  case (*Cons a x*)
  thus *?case*
    using *list-all-conv-same-lab-h[OF Cons(3)]*
    by (*metis option.distinct(1) same-lab.simps(2) same-lab-h.simps(2)*)
qed *auto*

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* $\land$ (*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*

165

$$\text{then type-update ts } [\textit{TSome t, TSome t}] \ (\textit{Type } [t])$$
$$\text{else Bot})$$

| *check-single* $\mathcal{C}$ (*Testop t -*) *ts* = (*if is-int-t t*
$$\text{then type-update ts } [\textit{TSome t}] \ (\textit{Type } [\textit{T-i32} \ (\textit{t-sec } t)])$$
$$\text{else Bot})$$

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

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

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

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

| *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*)
$$\text{then type-update ts } [\textit{TSome t2}] \ (\textit{Type } [t1])$$
$$\text{else Bot})$$

| *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*)
$$\text{then type-update ts } [\textit{TSome t2}] \ (\textit{Type } [t1])$$
$$\text{else Bot})$$

| *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* ($\mathcal{C}\langle\!|$*label* := ([*tm*]
@ (*label* $\mathcal{C}$))$|\!\rangle$) *es* (*tn -> tm*))
$$\text{then type-update ts } (\textit{to-ct-list tn}) \ (\textit{Type tm})$$
$$\text{else Bot})$$

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

166

| *check-single C* (*If* (*tn -> tm*) *es1 es2*) *ts* = (*if* (*b-e-type-checker* (*C*⦇*label* := ([*tm*] @ (*label C*))⦈)) *es1* (*tn -> tm*)

∧ *b-e-type-checker* (*C*⦇*label* := ([*tm*] @ (*label C*))⦈)) *es2* (*tn -> tm*))

then *type-update ts* (*to-ct-list* (*tn*@[*T-i32 Public*])) (*Type tm*)

else *Bot*)

| *check-single C* (*Br i*) *ts* = (*if i < length* (*label C*)

then *type-update ts* (*to-ct-list* ((*label C*)!*i*)) (*TopType* []))
else *Bot*)

| *check-single C* (*Br-if i*) *ts* = (*if i < length* (*label C*)

then *type-update ts* (*to-ct-list* ((*label C*)!*i* @ [*T-i32 Public*])) (*Type* ((*label C*)!*i*))

else *Bot*)

| *check-single C* (*Br-table is i*) *ts* = (*case* (*same-lab* (*is*@[*i*]) (*label C*)) *of*

None ⇒ *Bot*

| *Some tls* ⇒ *type-update ts* (*to-ct-list* (*tls* @ [*T-i32 Public*])) (*TopType* []))

| *check-single C* (*Return*) *ts* = (*case* (*return C*) *of*

None ⇒ *Bot*

| *Some tls* ⇒ *type-update ts* (*to-ct-list tls*) (*TopType* []))

| *check-single C* (*Call i*) *ts* = (*if i < length* (*func-t C*)

then (*case* ((*func-t C*)!*i*) *of*

(*tr*,(*tn -> tm*)) ⇒ *if* (*trust-compat* (*trust-t C*) *tr*)

then *type-update ts* (*to-ct-list tn*) (*Type tm*)

else *Bot*)

else *Bot*)

| *check-single C* (*Call-indirect i*) *ts* = (*if* (*table C*) ≠ *None* ∧ *i < length* (*types-t C*)

then (*case* ((*types-t C*)!*i*) *of*

(*tr*,(*tn -> tm*)) ⇒ *if* (*trust-compat* (*trust-t C*) *tr*)

then *type-update ts* (*to-ct-list* (*tn*@[*T-i32 Public*])) (*Type tm*)

else *Bot*)

else *Bot*)

| *check-single C* (*Get-local i*) *ts* = (*if i < length* (*local C*)

then *type-update ts* [] (*Type* [(*local C*)!*i*])
else *Bot*)

| *check-single C* (*Set-local i*) *ts* = (*if i < length* (*local C*)

$then\ type\text{-}update\ ts\ [TSome\ ((local\ \mathcal{C})!i)]\ (Type\ [])$
$else\ Bot)$

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

$else\ Bot)$

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

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

$else\ Bot)$

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

$|\ check\text{-}single\ \mathcal{C}\ (Store\ t\ tp\ a\ off)\ ts\ =$
$(case\ (memory\ \mathcal{C})\ of$
$Some\ (m,\ sec)\ \Rightarrow$
$if\ t\text{-}sec\ t\ =\ sec\ \wedge\ load\text{-}store\text{-}t\text{-}bounds\ a\ tp\ t$
$then\ type\text{-}update\ ts\ [TSome\ (T\text{-}i32\ Public), TSome\ t]$
$(Type\ [])$
$else\ Bot$
$|\ None\ \Rightarrow\ Bot)$

$|\ check\text{-}single\ \mathcal{C}\ Current\text{-}memory\ ts\ =\ (if\ (memory\ \mathcal{C})\ \neq\ None$
$then\ type\text{-}update\ ts\ []\ (Type\ [T\text{-}i32\ Public])$
$else\ Bot)$

$|\ check\text{-}single\ \mathcal{C}\ Grow\text{-}memory\ ts\ =\ (if\ (memory\ \mathcal{C})\ \neq\ None$
$then\ type\text{-}update\ ts\ [TSome\ (T\text{-}i32\ Public)]\ (Type$
$[T\text{-}i32\ Public])$

$else\ Bot)$

end

168

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*
       $\mathcal{C} \vdash [e] : (t\text{-}in \text{ -> } t\text{-}out)$
  shows $\exists tn.$ *c-types-agree* (*Type x1*) *tn* $\wedge$ $\mathcal{C} \vdash [e] : (tn \text{ -> } tm)$
  using *assms(2) b-e-typing.weakening[OF assms(3)] type-update-type[OF assms(1)]*
  by *auto*

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* $\wedge$ $\mathcal{C} \vdash [e] : (tn \text{ -> } tm)$
proof −
  obtain *t-ag* **where** *t-ag-def*:*ct-suffix* (*to-ct-list t-out*) *x2*
                 tm = *t-ag* @ *t-out*
                 c-types-agree (*TopType x1*) (*t-ag* @ *t-in*)
    using *type-update-top-top[OF assms(1,2)]*
    by *fastforce*
  hence $\mathcal{C} \vdash [e] : (t\text{-}ag@t\text{-}in \text{ -> } t\text{-}ag@t\text{-}out)$
    using *b-e-typing.weakening[OF assms(3)]*
    by *fastforce*
  thus *?thesis*
    using *t-ag-def*
    by *fastforce*
qed

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*
proof (*cases ts*)
  case (*TopType x1*)
  then obtain *t-int* **where** *consume* (*TopType x1*) (*to-ct-list t-in*) = *t-int t-int* $\neq$ *Bot*
    using *assms(1,2,3)*
    by *fastforce*
  thus *?thesis*
    using *TopType ct-suffix-ct-list-eq-exists ct-suffix-ts-conv-suffix*
    unfolding *consume.simps*
    by (*metis append-Nil c-types-agree.simps(2) ct-suffix-def*)
next

case (*Type x2*)
then obtain *t-int* **where** *consume* (*Type x2*) (*to-ct-list t-in*) = *t-int t-int* ≠ *Bot*
  using *assms(1,2,3)*
  by *fastforce*
thus *?thesis*
  using *c-types-agree-id Type consume-type suffixI ct-suffix-ts-conv-suffix*
  by *fastforce*
next
  case *Bot*
  thus *?thesis*
    using *assms(2)*
    by *simp*
qed

lemma *b-e-check-single-type-not-bot-sound*:
  assumes *type-update ts* (*to-ct-list t-in*) (*Type t-out*) = *ts′*
        ts ≠ *Bot*
        ts′ ≠ *Bot*
        c-types-agree ts′ tm
        $\mathcal{C}$ ⊢ [*e*] : (*t-in -> t-out*)
  shows ∃ *tn*. *c-types-agree ts tn* ∧ $\mathcal{C}$ ⊢ [*e*] : (*tn -> tm*)
  using *assms b-e-check-single-type-sound*
proof (*cases ts*)
  case (*TopType x1*)
  then obtain *x1′* **where** *x-def*:*TopType x1′* = *ts′*
    using *assms*
    by (*simp, metis* (*full-types*) *produce.simps(1) produce.simps(6)*)
  thus *?thesis*
    using *assms b-e-check-single-top-sound TopType*
    by *fastforce*
next
  case (*Type x2*)
  then obtain *x2′* **where** *x-def*:*Type x2′* = *ts′*
    using *assms*
    by (*simp, metis* (*full-types*) *produce.simps(2) produce.simps(6)*)
  thus *?thesis*
    using *assms b-e-check-single-type-sound Type*
    by *fastforce*
next
  case *Bot*
  thus *?thesis*
    using *assms(2)*
    by *simp*
qed

lemma *b-e-check-single-sound-unop-testop-cvtop*:
  assumes *check-single* $\mathcal{C}$ *e tn′* = *tm′*
      ((*e* = (*Unop-i t uu*) ∨ *e* = (*Testop t uv*)) ∧ *is-int-t t*)

$\lor\ (e = (Unop\text{-}f\ t\ uw) \land is\text{-}float\text{-}t\ t)$
$\lor\ (e = (Cvtop\ t1\ Convert\ t\ sx) \land convert\text{-}cond\ t1\ t\ sx)$
$\lor\ (e = (Cvtop\ t1\ Reinterpret\ t\ sx) \land ((t1 \neq t) \land (t\text{-}sec\ t1 = t\text{-}sec\ t) \land$
$t\text{-}length\ t1 = t\text{-}length\ t \land sx = None))$
$\lor\ (e = (Cvtop\ t1\ Classify\ t\ sx) \land (is\text{-}int\text{-}t\ t \land is\text{-}public\text{-}t\ t \land classify\text{-}t\ t$
$= t1 \land sx = None))$
$\lor\ (e = (Cvtop\ t1\ Declassify\ t\ sx) \land ((trust\text{-}t\ \mathcal{C}) = Trusted \land is\text{-}int\text{-}t\ t \land$
$is\text{-}secret\text{-}t\ t \land declassify\text{-}t\ t = t1 \land sx = None))$
c-types-agree tm′ tm
$tn' \neq Bot$
$tm' \neq Bot$
shows $\exists\ tn.\ c\text{-}types\text{-}agree\ tn'\ tn \land \mathcal{C} \vdash [e] : (tn\ \text{->}\ tm)$
proof −
  have $(e = (Cvtop\ t1\ Convert\ t\ sx) \Longrightarrow convert\text{-}cond\ t1\ t\ sx)$
    using *assms(2)*
    by *simp*
  hence *temp0*:$(e = (Cvtop\ t1\ Convert\ t\ sx)) \Longrightarrow (type\text{-}update\ tn'\ [TSome\ t]\ (Type$
$[arity\text{-}1\text{-}result\ e]) = tm')$
    using *assms(1,5) arity-1-result-def*
    by (*simp del: convert-cond.simps*)
  have *temp1*:$(e = (Cvtop\ t1\ Reinterpret\ t\ sx)) \Longrightarrow (type\text{-}update\ tn'\ [TSome\ t]$
$(Type\ [arity\text{-}1\text{-}result\ e]) = tm')$
    using *assms(1,2,5) arity-1-result-def*
    by *simp*
  have *1*:$type\text{-}update\ tn'\ (to\text{-}ct\text{-}list\ [t])\ (Type\ [arity\text{-}1\text{-}result\ e]) = tm'$
    using *assms arity-1-result-def*
    unfolding *to-ct-list-def*
    apply (*simp del: convert-cond.simps*)
    apply (*metis (no-types, lifting) b-e.simps(979,980,983,986) check-single.simps(11,12)*
check-single.simps(2,3,6) cvtop.simps(15,16) temp0 temp1 type-update.simps)
    done
  have $\mathcal{C} \vdash [e] : ([t]\ \text{->}\ [arity\text{-}1\text{-}result\ e])$
    using *assms(2)*
    unfolding *arity-1-result-def*
    using *b-e-typing.intros(2,3,6,9,10,11,12)*
    by *fastforce*
  thus *?thesis*
    using *b-e-check-single-type-not-bot-sound[OF 1 assms(4,5,3)]*
    by *fastforce*
qed

lemma *b-e-check-single-sound-binop-relop*:
  assumes *check-single $\mathcal{C}$ e tn′ = tm′*
    $((e = Binop\text{-}i\ t\ iop \land is\text{-}int\text{-}t\ t \land (is\text{-}secret\text{-}t\ t \longrightarrow safe\text{-}binop\text{-}i\ iop))$
     $\lor\ (e = Binop\text{-}f\ t\ fop \land is\text{-}float\text{-}t\ t)$
     $\lor\ (e = Relop\text{-}i\ t\ irop \land is\text{-}int\text{-}t\ t)$
     $\lor\ (e = Relop\text{-}f\ t\ frop \land is\text{-}float\text{-}t\ t))$
    c-types-agree tm′ tm
    $tn' \neq Bot$

$tm' \neq Bot$
shows $\exists\, tn.$ *c-types-agree tn' tn* $\land\, \mathcal{C} \vdash [e] : (tn$ -> $tm)$
proof −
  have *type-update tn' (to-ct-list [t,t]) (Type [arity-2-result e]) = tm'*
    using *assms arity-2-result-def*
    unfolding *to-ct-list-def*
    by *auto*
  moreover
  have $\mathcal{C} \vdash [e] : ([t,t]$ -> $[arity\text{-}2\text{-}result\ e])$
    using *assms(2) b-e-typing.intros(4,5,7,8)*
    unfolding *arity-2-result-def*
    by *fastforce*
  ultimately
  show *?thesis*
    using *b-e-check-single-type-not-bot-sound*[*OF - assms(4,5,3)*]
    by *fastforce*
qed

lemma *b-e-type-checker-sound*:
  assumes *b-e-type-checker* $\mathcal{C}$ *es* $(tn$ -> $tm)$
  shows $\mathcal{C} \vdash es : (tn$ -> $tm)$
proof −
  fix $e\ tn'$
  have *b-e-type-checker* $\mathcal{C}$ *es* $(tn$ -> $tm) \Longrightarrow$
      $\mathcal{C} \vdash es : (tn$ -> $tm)$
  and $\bigwedge tm'\ tm.$
    check $\mathcal{C}$ *es* $tn' = tm' \Longrightarrow$
    c-types-agree $tm'\ tm \Longrightarrow$
    $\exists\, tn.$ *c-types-agree tn' tn* $\land\, \mathcal{C} \vdash es : (tn$ -> $tm)$
  and $\bigwedge tm'\ tm.$
    check-single $\mathcal{C}$ $e\ tn' = tm' \Longrightarrow$
    c-types-agree $tm'\ tm \Longrightarrow$
    $tn' \neq Bot \Longrightarrow$
    $tm' \neq Bot \Longrightarrow$
    $\exists\, tn.$ *c-types-agree tn' tn* $\land\, \mathcal{C} \vdash [e] : (tn$ -> $tm)$
  proof (*induction rule*: *b-e-type-checker-check-check-single.induct*)
    case (*1* $\mathcal{C}$ *es tn' tm*)
    thus *?case*
      by *simp*
  next
    case (*2* $\mathcal{C}$ *es' ts*)
    show *?case*
    proof (*cases es'*)
      case *Nil*
      thus *?thesis*
        using *2(5,6)*
        by (*simp add*: *b-e-type-empty*)
    next
      case (*Cons e es*)

thus *?thesis*
proof (*cases ts*)
case (*TopType x1*)
have *check-expand*:*check C es* (*check-single C e ts*) = *tm'*
  using *2*(*5,6*) *TopType Cons*
  by *simp*
obtain *ts'* **where** *ts'-def*:*check-single C e ts* = *ts'*
  by *blast*
obtain *t-int* **where** *t-int-def*:*C ⊢ es* : (*t-int* -> *tm*)
                            c-types-agree ts' t-int
  using *2*(*2*)[*OF Cons TopType check-expand 2*(*6*)] *ts'-def*
  by *blast*
obtain *t-int'* **where** *c-types-agree ts t-int' C ⊢* [*e*] : (*t-int'* -> *t-int*)
 using *2*(*1*)[*OF Cons - ts'-def*] *TopType c-types-agree.simps*(*3*) *t-int-def*(*2*)
  by *blast*
thus *?thesis*
  using *t-int-def*(*1*) *b-e-type-comp-conc Cons*
  by *fastforce*
next
  case (*Type x2*)
  have *check-expand*:*check C es* (*check-single C e ts*) = *tm'*
    using *2*(*5,6*) *Type Cons*
    by *simp*
  obtain *ts'* **where** *ts'-def*:*check-single C e ts* = *ts'*
    by *blast*
  obtain *t-int* **where** *t-int-def*:*C ⊢ es* : (*t-int* -> *tm*)
                              c-types-agree ts' t-int
    using *2*(*4*)[*OF Cons Type check-expand 2*(*6*)] *ts'-def*
    by *blast*
  obtain *t-int'* **where** *c-types-agree ts t-int' C ⊢* [*e*] : (*t-int'* -> *t-int*)
    using *2*(*3*)[*OF Cons - ts'-def*] *Type c-types-agree.simps*(*3*) *t-int-def*(*2*)
    by *blast*
  thus *?thesis*
    using *t-int-def*(*1*) *b-e-type-comp-conc Cons*
    by *fastforce*
next
  case *Bot*
  then show *?thesis*
    using *2*(*5,6*) *Cons*
    by *auto*
  qed
qed
next
case (*3 C v ts*)
hence *type-update ts* [] (*Type* [*typeof v*]) = *tm'*
  by *simp*
moreover
have *C ⊢* [*C v*] : ([] -> [*typeof v*])
  using *b-e-typing.intros*(*1*)

      by *blast*
  ultimately
  show *?case*
    using *b-e-check-single-type-not-bot-sound*[*OF - 3(3,4,2)*]
    by (*metis list.simps(8) to-ct-list-def*)
next
  case (*4 C t uu ts*)
  hence *is-int-t t*
    by (*simp*, *meson*)
  thus *?case*
    using *b-e-check-single-sound-unop-testop-cvtop 4*
    by *fastforce*
next
  case (*5 C t uv ts*)
  hence *is-float-t t*
    by (*simp*, *meson*)
  thus *?case*
    using *b-e-check-single-sound-unop-testop-cvtop 5*
    by *fastforce*
next
  case (*6 C t uw ts*)
  hence *is-int-t t $\wedge$ (is-secret-t t $\longrightarrow$ safe-binop-i uw)*
    by (*simp*, *meson*)
  thus *?case*
    using *b-e-check-single-sound-binop-relop 6*
    by *fastforce*
next
  case (*7 C t ux ts*)
  hence *is-float-t t*
    by (*simp*, *meson*)
  thus *?case*
    using *b-e-check-single-sound-binop-relop 7*
    by *fastforce*
next
  case (*8 C t uy ts*)
  hence *is-int-t t*
    by (*simp*, *meson*)
  thus *?case*
    using *b-e-check-single-sound-unop-testop-cvtop 8*
    by *fastforce*
next
  case (*9 C t uz ts*)
  hence *is-int-t t*
    by (*simp*, *meson*)
  thus *?case*
    using *b-e-check-single-sound-binop-relop 9*
    by *fastforce*
next
  case (*10 C t va ts*)

hence *is-float-t t*
  by (*simp*, *meson*)
thus *?case*
  using *b-e-check-single-sound-binop-relop 10*
  by *fastforce*
next
  case (*11 C t1 t2 sx ts*)
  hence *convert-cond t1 t2 sx*
    by (*simp del*: *convert-cond.simps*, *meson*)
  thus *?case*
    using *b-e-check-single-sound-unop-testop-cvtop 11*
    by *fastforce*
next
  case (*12 C t1 t2 sx ts*)
  hence *t1 ≠ t2 ∧ (t-sec t1 = t-sec t2) ∧ t-length t1 = t-length t2 ∧ sx = None*
    by (*simp*, *presburger*)
  thus *?case*
    using *b-e-check-single-sound-unop-testop-cvtop 12*
    by *fastforce*
next
  case (*13 C t1 t2 sx ts*)
  hence (*is-int-t t2 ∧ is-public-t t2 ∧ classify-t t2 = t1 ∧ sx = None*)
    by (*simp*, *meson*)
  thus *?case*
    using *b-e-check-single-sound-unop-testop-cvtop 13*
    by *fastforce*
next
  case (*14 C t1 t2 sx ts*)
  hence (*trust-t C = Trusted ∧ is-int-t t2 ∧ is-secret-t t2 ∧ declassify-t t2 = t1
∧ sx = None*)
    by (*simp*, *meson*)
  thus *?case*
    using *b-e-check-single-sound-unop-testop-cvtop 14*
    by *fastforce*
next
  case (*15 C ts*)
  thus *?case*
    using *b-e-typing.intros*(*13*) *c-types-agree-not-bot-exists*
    by *blast*
next
  case (*16 C ts*)
  thus *?case*
    using *b-e-typing.intros*(*14,37*)
    by *fastforce*
next
  case (*17 C ts*)
  thus *?case*
  proof (*cases ts*)
    case (*TopType x1*)

175

thus *?thesis*
  proof (*cases x1 rule*: *List.rev-cases*)
    case *Nil*
    have $\mathcal{C} \vdash [Drop] : (tm@[T\text{-}i32\ Public] \mathrel{-\!\!>} tm)$
      using *b-e-typing.intros(15,37)*
      by *fastforce*
    thus *?thesis*
      using *c-types-agree-top1 Nil TopType*
      by *fastforce*
  next
    case (*snoc ys y*)
      hence *temp1*:$(consume\ (TopType\ (ys@[y]))\ [TAny]) = tm'$
        using *17 TopType type-update-empty*
        by (*metis check-single.simps(15)*)
      hence *temp2*:*c-types-agree* (*TopType ys*) *tm*
        using *consume-top-geq[OF temp1] 17(2,3,4)*
            by (*metis Suc-leI add-diff-cancel-right′ append-eq-conv-conj con-*
sume.simps(2)
                ct-suffix-def length-Cons length-append list.size(3) trans-le-add2
                zero-less-Suc)
      obtain *t* **where** *ct-list-eq* [*y*] (*to-ct-list* [*t*])
        using *ct-list-eq-exists*
        unfolding *ct-list-eq-def to-ct-list-def list-all2-map2*
        by (*metis list-all2-Cons1 list-all2-Nil*)
      hence *c-types-agree ts* (*tm@[t]*)
        using *temp2 ct-suffix-extend-ct-list-eq snoc TopType*
        by (*simp add*: *to-ct-list-def*)
      thus *?thesis*
        using *b-e-typing.intros(15,37)*
        by *fastforce*
  qed
next
  case (*Type x2*)
  thus *?thesis*
  proof (*cases x2 rule*: *List.rev-cases*)
    case *Nil*
    hence (*consume* (*Type* []) [*TAny*]) = *tm′*
      using *17 Type type-update-empty*
      by *fastforce*
    thus *?thesis*
      using *17(4) ct-list-eq-def ct-suffix-def to-ct-list-def*
      by *simp*
  next
    case (*snoc ys y*)
        hence *temp1*:$(consume\ (Type\ (ys@[y]))\ [TAny]) = tm'$
          using *17 Type type-update-empty*
          by (*metis check-single.simps(15)*)
        hence *temp2*:*c-types-agree* (*Type ys*) *tm*
          using *17(2,3,4) ct-suffix-def*

176

by (*simp, metis One-nat-def butlast-conv-take butlast-snoc c-types-agree.simps(1)*
length-Cons list.size(3))

    obtain t **where** *ct-list-eq* [*TSome y*] (*to-ct-list* [t])

      using *ct-list-eq-exists*

      unfolding *ct-list-eq-def to-ct-list-def list-all2-map2*

      by (*metis list-all2-Cons1 list-all2-Nil*)

    hence *c-types-agree ts* (*tm*@[t])

      using *temp2 ct-suffix-extend-ct-list-eq snoc Type*

      by (*simp add*: *ct-list-eq-def to-ct-list-def*)

    thus *?thesis*

      using *b-e-typing.intros(15,37)*

      by *fastforce*

  qed

qed *simp*

next

  case (*18 C sec ts*)

  thus *?case*

  proof (*cases ts*)

    case (*TopType x1*)

    consider

      (*1*) *length x1 = 0*

     | (*2*) *length x1 = 1*

     | (*3*) *length x1 = 2*

     | (*4*) *length x1 $\geq$ 3*

     by *linarith*

    thus *?thesis*

    proof (*cases*)

      case *1*

      hence $tm' = TopType$ [*sec-ct sec*]

        using *TopType 18*

        by *simp*

      then obtain t'' tm'' **where** *tm-def*:$tm = tm''$@[t'']

                          ct-list-eq [*sec-ct sec*] [*TSome t''*]

        using *18*(*2*) *c-types-agree-imp-ct-list-eq*[*of* [*sec-ct sec*] *tm*]

        by (*simp add*: *ct-list-eq-def*)

          (*metis append-Nil ct-suffix-cons-ct-list-single*)

      have $\mathcal{C} \vdash$ [*Select sec*] : ([t'',t'',*T-i32 sec*] -> [t''])

        using *b-e-typing.intros(16) tm-def*(*2*)

        by (*cases sec*) (*simp-all add*: *ct-list-eq-def*)

      thus *?thesis*

        using *TopType 18 1 tm-def b-e-typing.weakening c-types-agree.simps(2)*

c-types-agree-top1

        by *fastforce*

    next

      case *2*

      have *type-update-select sec* (*TopType x1*) = tm'

        using *18 TopType*

        unfolding *check-single.simps*

        by *simp*

hence *x1-def*:*ct-list-eq x1* [*TSome* (*T-i32 sec*)] *tm′* = *TopType* [*sec-ct sec*]
  using *type-update-select-length1*[*OF - 2 18(4)*]
  by *simp-all*
 then obtain *t″ tm″* **where** *tm-def*:*tm* = *tm″*@[*t″*] *ct-list-eq* [*sec-ct sec*]
[*TSome t″*]
    using *18(2) c-types-agree-imp-ct-list-eq*[*of* [*sec-ct sec*] *tm*]
    by (*simp add*: *ct-list-eq-def*)
     (*metis append-Nil ct-suffix-cons-ct-list-single*)
  have *temp*:*c-types-agree* (*TopType x1*) ((*tm″*@[*t″*,*t′*])@[*T-i32 sec*])
   using *x1-def(1)*
   by (*metis c-types-agree-top2 list.simps(8,9) to-ct-list-def*)
  have $\mathcal{C}$ ⊢ [*Select sec*] : ([*t″*,*t″*,*T-i32 sec*] -> [*t′*])
   using *b-e-typing.intros(16) tm-def(2)*
   by (*cases sec*) (*simp-all add*: *ct-list-eq-def*)
  thus *?thesis*
  using *temp TopType 18 2 tm-def b-e-typing.weakening c-types-agree.simps(2)*
c-types-agree-top1
    by *fastforce*
  next
   case *3*
   have *type-update-select sec* (*TopType x1*) = *tm′*
    using *18 TopType*
    unfolding *check-single.simps*
    by *simp*
   then obtain *ct1 ct2* **where** *x1-def*:*x1* = [*ct1, ct2*]
                       ct-eq ct2 (*TSome* (*T-i32 sec*))
                       ct-eq ct1 (*sec-ct sec*)
                       tm′ = *TopType* [*ens-sec-ct sec ct1*]
    using *type-update-select-length2*[*OF - 3 18(4)*]
    by *blast*
   then obtain *t″ tm″* **where** *tm-def*:*tm* = *tm″*@[*t″*]
                      ct-list-eq [*ens-sec-ct sec ct1*] [(*TSome t″*)]
    using *18(2)*
    by (*simp add*: *ct-list-eq-def*)
     (*metis append-Nil ct-suffix-cons-ct-list-single*)
   hence *ct-list-eq x1* (*to-ct-list* [*t″, T-i32 sec*])
    using *x1-def(1,2,4) ens-sec-ct-imp-ct-eq*
    unfolding *ct-list-eq-def to-ct-list-def*
    by *fastforce*
   hence *c-types-agree* (*TopType x1*) ((*tm″*@[*t″*])@[*t″,T-i32 sec*])
    using *c-types-agree-top2*
    by *blast*
   have $\mathcal{C}$ ⊢ [*Select sec*] : ([*t″*,*t″*,*T-i32 sec*] -> [*t′*])
    using *b-e-typing.intros(16) tm-def(2) x1-def(2,3,4)*
    apply (*cases sec*)
    apply (*simp-all add*: *ct-eq-TSecret-imp-is-secret-t ct-list-eq-def*)
    done
   thus *?thesis*
    using *TopType b-e-typing.intros(16,37) tm-def x1-def(4)*

178

using ‹*c-types-agree* (*TopType x1*) ((*tm''* @ [*t''*]) @ [*t''*, *T-i32 sec*])› **by**
auto
    next
      case *4*
      then obtain *nat* **where** *nat-def*:*length x1 = Suc (Suc (Suc nat))*
        by (*metis add-eq-if diff-Suc-1 le-Suc-ex numeral-3-eq-3 nat.distinct(2)*)
      hence *tm'-def*:*type-update-select sec (TopType x1) = tm'*
        using *18 TopType*
        by *simp*
      then obtain *tm-int* **where** (*select-return-top sec x1*
                       (*x1 ! (length x1 − 2*))
                       (*x1 ! (length x1 − 3*))) = *tm-int*
                  tm-int ≠ *Bot*
        using *nat-def 18(4)*
        unfolding *type-update-select.simps*
        by *fastforce*
      then obtain *x2* **where** *x2-def*:(*select-return-top sec x1*
                        (*x1 ! (length x1 − 2*))
                        (*x1 ! (length x1 − 3*))) = *TopType x2*
        using *select-return-top-exists*
        by *fastforce*
     have *ct-suffix x1* [*sec-ct sec, sec-ct sec, TSome (T-i32 sec)*] ∨ *ct-suffix* [*sec-ct
sec, sec-ct sec, TSome (T-i32 sec)*] *x1*
        using *tm'-def nat-def 18(4)*
        by (*simp, metis (full-types) produce.simps(6)*)
      hence *tm'-eq*:*tm' = TopType x2*
        using *tm'-def nat-def 18(4) x2-def*
        by *force*
      then obtain *cts' ct1 ct2 ct3* **where** *cts'-def*:*x1 = cts'*@[*ct1, ct2, ct3*]
                               ct-eq ct3 (TSome (T-i32 sec))
                               ct-eq ct1 (sec-ct sec)
                               ct-eq ct2 (sec-ct sec)
        using *type-update-select-length3 tm'-def 4*
        by *blast*
      then obtain *c' cm'* **where** *tm-def*:*tm = cm'*@[*c'*]
                         ct-suffix cts' (to-ct-list cm')
                         ct-eq (x1 ! (length x1 − 2)) (*TSome c'*)
                         ct-eq (x1 ! (length x1 − 3)) (*TSome c'*)
        using *select-return-top-ct-eq*[*OF x2-def 4*] *tm'-eq 4 18(2)*
        by *fastforce*
      then obtain *as bs* **where** *cm'-def*:*cm' = as*@*bs*
                         ct-list-eq (to-ct-list bs) cts'
        using *ct-list-eq-cons-ct-list1 ct-list-eq-ts-conv-eq*
        by (*metis ct-suffix-def to-ct-list-append(2)*)
      hence *ct-eq ct1 (TSome c'*)
          ct-eq ct2 (TSome c')
        using *cts'-def tm-def*
        apply *simp-all*
      apply (*metis append.assoc append-Cons append-Nil length-append-singleton*

nth-append-length)
   done
  hence *c-types-agree ts (cm'@[c',c',(T-i32 sec)])*
   using *c-types-agree-top2[of - - as] cm'-def(1) TopType*
    ct-list-eq-concat[OF ct-list-eq-commute[OF cm'-def(2)]] cts'-def
   unfolding *to-ct-list-def ct-list-eq-def*
   by *fastforce*
  moreover
  have $sec = Secret \longrightarrow is\text{-}secret\text{-}t\ c'$
   using *select-return-top-secret 18(2) tm'-eq tm-def(1) x2-def*
   by *force*
  ultimately
  show *?thesis*
   using *b-e-typing.intros(16,37) tm-def*
   by *auto*
 qed
next

 case (*Type x2*)
 hence *x2-cond*:$(length\ x2 \geq 3 \land (x2!(length\ x2-2)) = (x2!(length\ x2-3)))$
  using *18*
  by (*simp, meson*)
 hence *tm'-def*:*consume* (*Type x2*) [*TAny, TSome (T-i32 sec)*] = *tm'*
    $(length\ x2 \geq 3 \land (x2!(length\ x2-2)) = (x2!(length\ x2-3)) \land (sec$
$= Secret \longrightarrow is\text{-}secret\text{-}t\ (x2!(length\ x2-2))))$
  using *18 Type*
  by (*simp-all split: if-splits*)
 obtain *ts' ts''* **where** *cts-def*:$x2 = ts'@\ ts''\ length\ ts'' = 3$
  using *x2-cond*
  by (*metis append-take-drop-id diff-diff-cancel length-drop*)
 then obtain *t1 ts''2* **where** $ts'' = t1\#ts''2\ length\ ts''2 = Suc\ (Suc\ 0)$
  using *List.length-Suc-conv[of ts' Suc (Suc 0)]*
  by (*metis length-Suc-conv numeral-3-eq-3*)
 then obtain *t2 t3* **where** $ts'' = [t1,t2,t3]$
  using *List.length-Suc-conv[of ts''2 Suc 0]*
  by (*metis length-0-conv length-Suc-conv*)
 hence *cts-def2*:$x2 = ts'@\ [t1,t2,t3]$
  using *cts-def*
  by *simp*
 have *ts'-suffix*:*ct-suffix* [*TAny, TSome (T-i32 sec)*] (*to-ct-list (ts' @ [t1, t2, t3])*)
  using *tm'-def 18(4)*
  by (*simp, metis cts-def2*)
 hence *tm'-def2*:$tm' = Type\ (ts'@[t1])$
  using *tm'-def 18(4) cts-def2*
  by *simp*
 obtain *as bs* **where** (*to-ct-list (ts' @ [t1])*) @ (*to-ct-list ([t2, t3])*) = *as@bs*
    ct-list-eq bs [*TAny, TSome (T-i32 sec)*]
  using *ts'-suffix*

180

    unfolding *ct-suffix-def to-ct-list-def*
    by *fastforce*
  hence *t3 = (T-i32 sec)*
    unfolding *to-ct-list-def ct-list-eq-def*
  by (*metis* (*no-types*, *lifting*) *Nil-is-map-conv append-eq-append-conv ct-eq.simps*(*1*)
       length-Cons list.sel(*1*,*3*) *list.simps*(*9*) *list-all2-Cons2 list-all2-lengthD*)
  moreover
  have *t1 = t2*
    using *x2-cond cts-def2*
  by (*simp*, *metis append.left-neutral append-Cons append-assoc length-append-singleton*
             nth-append-length)
  ultimately
  have *c-types-agree* (*Type x2*) ((*ts′*@[*t1*,*t1*])@[(*T-i32 sec*)])
    using *cts-def2*
    by *simp*
  thus *?thesis*
    using *b-e-typing.intros*(*16*,*37*) *Type tm′-def tm′-def2 18*(*2*)
    by *fastforce*
 qed *simp*
next
 case (*19 C tn″ tm″ es ts*)
 hence *type-update ts* (*to-ct-list tn″*) (*Type tm″*) = *tm′*
  by *auto*
 moreover
 have (*b-e-type-checker* (*C*⦇*label* := ([*tm″*] @ (*label C*))⦈) *es* (*tn″ -> tm″*))
  using *19*
  by (*simp*, *meson*)
 hence *C ⊢ [Block (tn″ -> tm″) es]* : (*tn″ -> tm″*)
  using *b-e-typing.intros*(*17*)[*OF - 19*(*1*)]
  by *blast*
 ultimately
 show *?case*
  using *b-e-check-single-type-not-bot-sound*[*OF - 19*(*4*,*5*,*3*)]
  by *blast*
next
 case (*20 C tn″ tm″ es ts*)
 hence *type-update ts* (*to-ct-list tn″*) (*Type tm″*) = *tm′*
  by *auto*
 moreover
 have (*b-e-type-checker* (*C*⦇*label* := ([*tn″*] @ (*label C*))⦈) *es* (*tn″ -> tm″*))
  using *20*
  by (*simp*, *meson*)
 hence *C ⊢ [Loop (tn″ -> tm″) es]* : (*tn″ -> tm″*)
  using *b-e-typing.intros*(*18*)[*OF - 20*(*1*)]
  by *blast*
 ultimately
 show *?case*
  using *b-e-check-single-type-not-bot-sound*[*OF - 20*(*4*,*5*,*3*)]
  by *blast*

next
  case (*21 C tn″ tm″ es1 es2 ts*)
  hence *type-update ts (to-ct-list (tn″@[(T-i32 Public)])) (Type tm″) = tm′*
    by *auto*
  moreover
  have (*b-e-type-checker (C⦇label := ([tm″] @ (label C))⦈) es1 (tn″ -> tm″)*)
    (*b-e-type-checker (C⦇label := ([tm″] @ (label C))⦈) es2 (tn″ -> tm″)*)
    using *21*
    by (*simp, meson*)+
  hence $C \vdash$ [*If (tn″ -> tm″) es1 es2*] : (*tn″@[(T-i32 Public)] -> tm″*)
    using *b-e-typing.intros(19)[OF - 21(1,2)]*
    by *blast*
  ultimately
  show *?case*
    using *b-e-check-single-type-not-bot-sound[OF - 21(5,6,4)]*
    by *blast*
next
  case (*22 C i ts*)
  hence *type-update ts (to-ct-list ((label C)!i)) (TopType []) = tm′*
    by *auto*
  moreover
  have *i < length (label C)*
    using *22*
    by (*simp, meson*)
  ultimately
  show *?case*
    using *b-e-check-single-top-not-bot-sound[OF - 22(3,4)]*
        b-e-typing.intros(20)
        b-e-typing.weakening
    by (*metis suffix-def*)
next
  case (*23 C i ts*)
   hence *type-update ts (to-ct-list ((label C)!i @ [T-i32 Public])) (Type ((label C)!i)) = tm′*
    by *auto*
  moreover
  have *i < length (label C)*
    using *23*
    by (*simp, meson*)
  hence $C \vdash$ [*Br-if i*] : ((*label C*)!i @ [*T-i32 Public*] -> (*label C*)!i)
    using *b-e-typing.intros(21)*
    by *fastforce*
  ultimately
  show *?case*
    using *b-e-check-single-type-not-bot-sound[OF - 23(3,4,2)]*
    by *fastforce*
next
  case (*24 C is i ts*)
  then obtain *tls* **where** *tls-def*:(*same-lab (is@[i]) (label C)*) = *Some tls*

by *fastforce*

hence *type-update ts (to-ct-list (tls @ [T-i32 Public])) (TopType []) = tm'*

  using *24*

  by *simp*

thus *?case*

  using *b-e-check-single-top-not-bot-sound[OF - 24(3,4)]*

      b-e-typing.intros(22)[OF same-lab-conv-list-all[OF tls-def]]

      b-e-typing.weakening

  by *(metis suffix-def)*

next

  case *(25 C ts)*

  then obtain *ts-r* **where** *(return C) = Some ts-r*

    by *fastforce*

  moreover

  hence *type-update ts (to-ct-list ts-r) (TopType []) = tm'*

    using *25*

    by *simp*

  ultimately

  show *?case*

    using *b-e-check-single-top-not-bot-sound[OF - 25(3,4)]*

      b-e-typing.intros(23)

    by *(metis suffix-def)*

next

  case *(26 C i ts)*

  obtain *tr'' tn'' tm''* **where** *func-def:(func-t C)!i = (tr'',(tn'' -> tm''))*

    by *(metis prod.exhaust tf.exhaust)*

  hence *type-update ts (to-ct-list tn'') (Type tm'') = tm'*

      i < length (func-t C)

      trust-compat (trust-t C) tr''

    using *26*

    by *(auto split: if-splits)*

  moreover

  hence $\mathcal{C} \vdash [Call\ i] : (tn'' \rightarrow tm'')$

    using *b-e-typing.intros(24) func-def*

    by *fastforce*

  ultimately

  show *?case*

    using *b-e-check-single-type-not-bot-sound[OF - 26(3,4,2)]*

    by *fastforce*

next

  case *(27 C i ts)*

  obtain *tr'' tn'' tm''* **where** *func-def:(types-t C)!i = (tr'',(tn'' -> tm''))*

    by *(metis prod.exhaust tf.exhaust)*

  hence *type-update ts (to-ct-list (tn''@[T-i32 Public])) (Type tm'') = tm'*

      (table C) ≠ None ∧ i < length (types-t C)

      trust-compat (trust-t C) tr''

    using *27*

    by *(auto split: if-splits)*

  moreover

hence $\mathcal{C} \vdash [\textit{Call-indirect i}] : (\textit{tn}''@[\textit{T-i32 Public}] \rightarrow \textit{tm}'')$
  using *b-e-typing.intros(25) func-def*
  by *fastforce*
ultimately
show *?case*
  using *b-e-check-single-type-not-bot-sound*[*OF - 27(3,4,2)*]
  by *fastforce*
next
  case (*28 $\mathcal{C}$ i ts*)
  hence *type-update ts* [] (*Type* [(*local $\mathcal{C}$*)!*i*]) = *tm'*
    by *auto*
  moreover
  have $i < length$ (*local $\mathcal{C}$*)
    using *28*
    by (*simp*, *meson*)
  hence $\mathcal{C} \vdash [\textit{Get-local i}] : ([] \rightarrow [(\textit{local } \mathcal{C})!i])$
    using *b-e-typing.intros(26)*
    by *fastforce*
  ultimately
  show *?case*
    using *b-e-check-single-type-not-bot-sound*[*OF - 28(3,4,2)*]
    unfolding *to-ct-list-def*
    by (*metis list.map-disc-iff*)
next
  case (*29 $\mathcal{C}$ i ts*)
  hence *type-update ts* (*to-ct-list* [(*local $\mathcal{C}$*)!*i*]) (*Type* []) = *tm'*
    unfolding *to-ct-list-def*
    by *auto*
  moreover
  have $i < length$ (*local $\mathcal{C}$*)
    using *29*
    by (*simp*, *meson*)
  hence $\mathcal{C} \vdash [\textit{Set-local i}] : ([(\textit{local } \mathcal{C})!i] \rightarrow [])$
    using *b-e-typing.intros(27)*
    by *fastforce*
  ultimately
  show *?case*
    using *b-e-check-single-type-not-bot-sound*[*OF - 29(3,4,2)*]
    by *fastforce*
next
  case (*30 $\mathcal{C}$ i ts*)
  hence *type-update ts* (*to-ct-list* [(*local $\mathcal{C}$*)!*i*]) (*Type* [(*local $\mathcal{C}$*)!*i*]) = *tm'*
    unfolding *to-ct-list-def*
    by *auto*
  moreover
  have $i < length$ (*local $\mathcal{C}$*)
    using *30*
    by (*simp*, *meson*)
  hence $\mathcal{C} \vdash [\textit{Tee-local i}] : ([(\textit{local } \mathcal{C})!i] \rightarrow [(\textit{local } \mathcal{C})!i])$

using *b-e-typing.intros*(*28*)
  by *fastforce*
ultimately
show *?case*
  using *b-e-check-single-type-not-bot-sound*[*OF - 30*(*3*,*4*,*2*)]
  by *fastforce*
next
  case (*31 C i ts*)
  hence *type-update ts* [] (*Type* [*tg-t* ((*global C*)!*i*)]) = *tm*′
    by *auto*
  moreover
  have *i* < *length* (*global C*)
    using *31*
    by (*simp*, *meson*)
  hence *C* ⊢ [*Get-global i*] : ([] -> [*tg-t* ((*global C*)!*i*)])
    using *b-e-typing.intros*(*29*)
    by *fastforce*
  ultimately
  show *?case*
    using *b-e-check-single-type-not-bot-sound*[*OF - 31*(*3*,*4*,*2*)]
    unfolding *to-ct-list-def*
    by (*metis list.map-disc-iff*)
next
  case (*32 C i ts*)
  hence *type-update ts* (*to-ct-list* [*tg-t* ((*global C*)!*i*)]) (*Type* []) = *tm*′
    unfolding *to-ct-list-def*
    by *auto*
  moreover
  have *i* < *length* (*global C*) ∧ *is-mut* (*global C* ! *i*)
    using *32*
    by (*simp*, *meson*)
  then obtain *t* **where** (*global C* ! *i*) = (|*tg-mut* = *T-mut*, *tg-t* = *t*|) *i* < *length*
(*global C*)
    unfolding *is-mut-def*
    by (*cases global C* ! *i*, *auto*)
  hence *C* ⊢ [*Set-global i*] : ([*tg-t* (*global C* ! *i*)] -> [])
    using *b-e-typing.intros*(*30*)[*of i C tg-t* (*global C* ! *i*)]
    unfolding *is-mut-def tg-t-def*
    by *fastforce*
  ultimately
  show *?case*
    using *b-e-check-single-type-not-bot-sound*[*OF - 32*(*3*,*4*,*2*)]
    by *fastforce*
next
  case (*33 C t tp-sx a off ts*)
  then obtain *m sec* **where** *mem-def*:(*memory C*) = *Some* (*m*,*sec*)
    by (*fastforce split*: *option.splits*)
  hence *type-update ts* (*to-ct-list* [*T-i32 Public*]) (*Type* [*t*]) = *tm*′
      t-sec t = *sec* ∧ *load-store-t-bounds a* (*option-projl tp-sx*) *t*

    using *33*
    unfolding *to-ct-list-def*
    by (*auto split*: *if-splits*)
  moreover
  hence $\mathcal{C} \vdash [Load\ t\ tp\text{-}sx\ a\ off] : ([T\text{-}i32\ Public] \mathrel{->} [t])$
    using *b-e-typing.intros(31)* *mem-def*
    by *fastforce*
  ultimately
  show *?case*
    using *b-e-check-single-type-not-bot-sound*$[OF\text{ - }33(3,4,2)]$
    by *fastforce*
next
  case (*34 $\mathcal{C}$ t tp a off ts*)
  then obtain *m sec* **where** *mem-def*:$(memory\ \mathcal{C}) = Some\ (m,sec)$
    by (*fastforce split*: *option.splits*)
  hence *type-update ts* ($to\text{-}ct\text{-}list\ [T\text{-}i32\ Public,t]$) ($Type\ []$) $= tm'$
      t-sec t = sec $\wedge$ load-store-t-bounds a tp t
    using *34*
    unfolding *to-ct-list-def*
    by (*auto split*: *if-splits*)
  moreover
  hence $\mathcal{C} \vdash [Store\ t\ tp\ a\ off] : ([T\text{-}i32\ Public,t] \mathrel{->} [])$
    using *b-e-typing.intros(32)* *mem-def*
    by *fastforce*
  ultimately
  show *?case*
    using *b-e-check-single-type-not-bot-sound*$[OF\text{ - }34(3,4,2)]$
    by *fastforce*
next
  case (*35 $\mathcal{C}$ ts*)
  hence *type-update ts* $[]$ ($Type\ [T\text{-}i32\ Public]$) $= tm'$
    by *auto*
  moreover
  have *memory* $\mathcal{C} \neq None$
    using *35*
    by (*simp*, *meson*)
  hence $\mathcal{C} \vdash [Current\text{-}memory] : ([] \mathrel{->} [T\text{-}i32\ Public])$
    using *b-e-typing.intros(33)*
    by *fastforce*
  ultimately
  show *?case*
    using *b-e-check-single-type-not-bot-sound*$[OF\text{ - }35(3,4,2)]$
    unfolding *to-ct-list-def*
    by (*metis list.map-disc-iff*)
next
  case (*36 $\mathcal{C}$ ts*)
  hence *type-update ts* ($to\text{-}ct\text{-}list\ [T\text{-}i32\ Public]$) ($Type\ [T\text{-}i32\ Public]$) $= tm'$
    unfolding *to-ct-list-def*
    by *auto*

    moreover
    have *memory $\mathcal{C} \neq$ None*
      using *36*
      by (*simp, meson*)
    hence $\mathcal{C} \vdash$ [*Grow-memory*] : ([*T-i32 Public*] -> [*T-i32 Public*])
      using *b-e-typing.intros(34)*
      by *fastforce*
    ultimately
    show *?case*
      using *b-e-check-single-type-not-bot-sound*[*OF - 36(3,4,2)*]
      by *fastforce*
  qed
  thus *?thesis*
    using *assms*
    by *simp*
qed

10.2   Completeness

lemma *check-single-imp*:
  assumes *check-single $\mathcal{C}$ e ctn = ctm*
      ctm $\neq$ Bot
  shows *check-single $\mathcal{C}$ e = id*
      $\lor$ ($\exists$ *sec. check-single $\mathcal{C}$ e = (λ ctn. type-update-select sec ctn*))
      $\lor$ ($\exists$ *cons prods. (check-single $\mathcal{C}$ e = (λ ctn. type-update ctn cons prods*)))
proof $-$
  have *True*
  and *True*
  and *check-single $\mathcal{C}$ e ctn = ctm* $\Longrightarrow$
     ctm $\neq$ Bot $\Longrightarrow$
      ?thesis
    apply (*induction rule*: *b-e-type-checker-check-check-single.induct*)
   apply (*fastforce simp add*: *assms(2) split*: *tf.splits if-splits option.splits prod.splits*)+
    done
  thus *?thesis*
    using *assms*
    by *simp*
qed

lemma *check-equiv-fold*:
  check $\mathcal{C}$ es ts = foldl (λ ts e. (case ts of Bot $\Rightarrow$ Bot $\mid$ - $\Rightarrow$ check-single $\mathcal{C}$ e ts))
ts es
proof (*induction es arbitrary*: *ts*)
  case *Nil*
  thus *?case*
    by *simp*
next
  case (*Cons e es*)
  obtain *ts'* **where** *ts'-def*:*check $\mathcal{C}$ (e # es) ts = ts'*

```
      by blast
    show ?case
    proof (cases ts = Bot)
      case True
      thus ?thesis
        using ts′-def
        by (induction es, simp-all)
    next
      case False
      thus ?thesis
        using ts′-def Cons
        by (cases ts, simp-all)
    qed
qed

lemma check-neq-bot-snoc:
  assumes check C (es@[e]) ts ≠ Bot
  shows check C es ts ≠ Bot
  using assms
proof (induction es arbitrary: ts)
  case Nil
  thus ?case
    by (cases ts, simp-all)
next
  case (Cons a es)
  thus ?case
    by (cases ts, simp-all)
qed

lemma check-unfold-snoc:
  assumes check C es ts ≠ Bot
  shows check C (es@[e]) ts = check-single C e (check C es ts)
proof −
  obtain f where f-def:f = (λ e ts. (case ts of Bot ⇒ Bot | - ⇒ check-single C e
ts))
    by blast
  have f-simp:⋀ts. ts ≠ Bot ⟹ (f e ts = check-single C e ts)
  proof −
    fix ts
    show ts ≠ Bot ⟹ (f e ts = check-single C e ts)
      using f-def
      by (cases ts, simp-all)
  qed
  have check C (es@[e]) ts = foldl (λ ts e. (case ts of Bot ⇒ Bot | - ⇒ check-single
C e ts)) ts (es@[e])
    using check-equiv-fold
    by simp
  also
  have ... = foldr (λ e ts. (case ts of Bot ⇒ Bot | - ⇒ check-single C e ts)) (rev
```

$(es@[e]))$ *ts*
   using *foldl-conv-foldr*
   by *fastforce*
 also
 have *... = f e (foldr (λ e ts. (case ts of Bot $\Rightarrow$ Bot | - $\Rightarrow$ check-single C e ts))* $(rev\ es)$ *ts)*
   using *f-def*
   by *simp*
 also
 have *... = f e (check C es ts)*
   using *foldr-conv-foldl[of - (rev es) ts] rev-rev-ident[of es] check-equiv-fold*
   by *simp*
 also
 have *... = check-single C e (check C es ts)*
   using *assms f-simp*
   by *simp*
 finally
 show *?thesis* .
qed

lemma *check-single-imp-weakening*:
 assumes *check-single C e (Type t1s) = ctm*
     ctm $\neq$ Bot
     c-types-agree ctn t1s
     c-types-agree ctm t2s
 shows $\exists$ *ctm'. check-single C e ctn = ctm'* $\wedge$ *c-types-agree ctm' t2s*
proof −
 consider (1) *check-single C e = id*
    | (2) $\exists$ *sec. check-single C e = (λctn. type-update-select sec ctn)*
    | (3) $(\exists$ *cons prods. (check-single C e = (λctn. type-update ctn cons prods)))*
   using *check-single-imp assms*
   by *blast*
 thus *?thesis*
 proof $(cases)$
  case *1*
  thus *?thesis*
   using *assms(1,3,4)*
   by *fastforce*
 next

  case *2*
  then obtain *sec* **where** *outer-2:check-single C e = type-update-select sec*
   by *blast*

  hence *t1s-cond:(length t1s $\geq$ 3 $\wedge$ (t1s!(length t1s$-$2)) = (t1s!(length t1s$-$3))*
$\wedge$ *(sec = Secret $\longrightarrow$ is-secret-t (t1s!(length t1s$-$2))))*
   using *assms(1,2)*
   by *(metis type-update-select.simps(1))*
  hence *ctm-def:ctm = consume (Type t1s) [TAny, TSome (T-i32 sec)]*

189

using *assms(1,2) outer-2*
by (*metis type-update-select.simps(1)*)
then obtain *c-t* **where** *c-t-def*:*ctm = Type c-t*
using *assms(2)*
by (*meson consume.simps(1)*)
hence *t2s-eq*:*t2s = c-t*
using *assms(4)*
by *simp*
hence *t2s-len*:*length t2s > 0*
using *t1s-cond ctm-def c-t-def assms(2)*
by (*metis Suc-leI Suc-n-not-le-n checker-type.inject(2) consume.simps(1)*
diff-is-0-eq dual-order.trans length-0-conv length-Cons length-greater-0-conv
nat.simps(3) numeral-3-eq-3 take-eq-Nil)
have *t1s-suffix-full*:*ct-suffix* [*TAny, TSome* (*T-i32 sec*)] (*to-ct-list t1s*)
using *assms(2) ctm-def ct-suffix-less*
by (*metis consume.simps(1)*)
hence *t1s-suffix*:*ct-suffix* [*TSome* (*T-i32 sec*)] (*to-ct-list t1s*)
using *assms(2) ctm-def ct-suffix-less*
by (*metis append-butlast-last-id last.simps list.distinct(1)*)
obtain *t t1s′* **where** *t1s-suffix2*:*t1s = t1s′@[t,t,*(*T-i32 sec*)]
using *type-update-select-type-length3 assms(1) c-t-def outer-2*
by *fastforce*
hence *t2s-def*:*t2s = t1s′@[t]*
(*sec = Secret $\longrightarrow$ is-secret-t t*)
using *ctm-def c-t-def t2s-eq t1s-suffix assms(2) t1s-suffix-full t1s-cond*
by *auto*
show *?thesis*
using *assms(1,3,4)*
proof (*cases ctn*)
case (*TopType x1*)
consider
(*1*) *length x1 = 0*
| (*2*) *length x1 = 1*
| (*3*) *length x1 = 2*
| (*4*) *length x1 $\geq$ 3*
by *linarith*
thus *?thesis*
proof (*cases*)
case *1*
hence *check-single C e ctn = TopType* [*sec-ct sec*]
using *outer-2 TopType*
by *simp*
thus *?thesis*
using *ct-suffix-singleton to-ct-list-def t2s-len t2s-def*
apply (*cases sec*)
apply *simp-all*
apply (*metis can-secret-ct.simps(1) ct-eq.simps(4) ct-eq-commute ct-suffix-cons-it
ct-suffix-unfold-one self-append-conv2*)
done

next
  case *2*
  hence *ct-suffix [TSome (T-i32 sec)] x1*
    using *assms(3) TopType ct-suffix-imp-ct-list-eq ct-suffix-shared t1s-suffix*
  by (*metis One-nat-def append-Nil c-types-agree.simps(2) ct-list-eq-commute ct-suffix-def*

        diff-self-eq-0 drop-0 length-Cons list.size(3))
  hence *check-single C e ctn = TopType [sec-ct sec]*
    using *outer-2 TopType 2*
    by *simp*
  thus *?thesis*
    using *ct-suffix-singleton to-ct-list-def t2s-len t2s-def*
    apply (*cases sec*)
    apply *simp-all*
  apply (*metis can-secret-ct.simps(1) ct-eq.simps(4) ct-eq-commute ct-suffix-cons-it ct-suffix-unfold-one self-append-conv2*)
  done
next
  case *3*
   hence *sel-is*:*type-update-select sec (TopType x1) = type-update (TopType x1) [sec-ct sec, TSome (T-i32 sec)] (TopType [ens-sec-ct sec (x1!(length x1−2))])*
    using *TopType*
    by *simp*
  obtain *x1b x1c* **where** *x1-is*:*x1 = [x1b, x1c]*
    using *3*
    by (*metis length-0-conv length-Suc-conv numeral-2-eq-2*)
  hence *temp1*:*ct-list-eq [x1b, x1c] (to-ct-list [t,(T-i32 sec)])*
    using *assms(3) TopType 3 t1s-suffix2*
      ct-suffix-cons2[of x1 to-ct-list (t1s′ @ [t]) to-ct-list ([t, T-i32 sec])]
    by (*simp add*: *to-ct-list-def*)
  hence *ct-eq x1b (sec-ct sec) ct-eq x1c (TSome (T-i32 sec))*
    using *ct-eq-TSome-imp-ct-eq-TSecret[OF - t2s-def(2)]*
    by (*simp-all add*: *ct-list-eq-def to-ct-list-def*)
    hence *test2*:*consume (TopType x1) [sec-ct sec, TSome (T-i32 sec)] = (TopType [])*
    using *TopType ct-list-eq-def ct-suffix-def x1-is*
    by *auto*
  have *ct-eq (ens-sec-ct sec x1b) (TSome t)*
    using *temp1 ens-sec-ct-imp-ct-eq-sec t2s-def(2)*
    apply (*cases sec*)
    apply (*simp-all add*: *ct-list-eq-def to-ct-list-def*)
    done
  hence *c-types-agree (TopType [(ens-sec-ct sec x1b)]) t2s*
    using *t2s-def(1)*
  by *simp* (*metis (no-types, lifting) ct-list-eq-commute ct-list-eq-def ct-suffix-def list.simps(11,9) map-append temp1 to-ct-list-def*)
  moreover
  have *check-single C e ctn = (TopType [(ens-sec-ct sec x1b)])*
    using *TopType sel-is outer-2 3 x1-is test2*

        by *simp*
       ultimately
       show *?thesis*
         using *outer-2 t2s-def(1) TopType x1-is temp1*
         by *simp*
     next
      case *4*
       hence *sel-is*:*type-update-select sec* (*TopType x1*) = *type-update* (*TopType x1*) [*sec-ct sec, sec-ct sec, TSome* (*T-i32 sec*)]
$$(select\text{-}return\text{-}top\; sec\; x1\; (x1!(length\; x1{-}2))\; (x1!(length\; x1{-}3)))$$
        using *TopType*
        by (*auto simp del*: *type-update.simps split*: *nat.splits*)
       obtain *nat* **where** *nat-def*:*length x1 = Suc* (*Suc* (*Suc nat*))
        by (*metis add-eq-if diff-Suc-1 le-Suc-ex numeral-3-eq-3 4 nat.distinct(2)*)
       obtain *x1′ xa xb xc* **where** *x1-split*:*x1 = x1′@[xa,xb,xc]*
       proof −
        assume *local-assms*:($\bigwedge$*x1′ x x′ x″. x1 = x1′ @ [x, x′, x″]* $\Longrightarrow$ *thesis*)
        obtain *x1′ x1″* **where** *tn-split*:*x1 = x1′@x1″*
                        length x1″ = 3
         using *4*
         by (*metis append-take-drop-id diff-diff-cancel length-drop*)
        then obtain *x x1″2* **where** *x1″ = x#x1″2 length x1″2 = Suc* (*Suc 0*)
         by (*metis length-Suc-conv numeral-3-eq-3*)
        then obtain *x′ x″* **where** *tn″-def*:*x1″= [x,x′,x″]*
         using *List.length-Suc-conv*[*of x1″2 Suc 0*]
         by (*metis length-0-conv length-Suc-conv*)
        thus *?thesis*
         using *tn-split local-assms*
         by *simp*
       qed
       hence *test4*:*ct-suffix x1′* (*to-ct-list t1s′*) ∧ *ct-list-eq* [*xa, xb, xc*] (*to-ct-list [t,t,(T-i32 sec)]*)
        using *assms*(*3*) *TopType t1s-suffix2 ct-suffix-cons-ct-list*[*of x1′ [xa, xb, xc] t1s′ @ [t, t, T-i32 sec]*]
      by *simp* (*metis append-eq-append-conv ct-list-eq-def length-Cons list.rel-map(2) list.size(3) list-all2-lengthD to-ct-list-def*)
      hence *test3*:*ct-eq xa* (*sec-ct sec*) *ct-eq xb* (*sec-ct sec*) *ct-eq xc* (*TSome* (*T-i32 sec*))
        using *ct-eq-TSome-imp-ct-eq-TSecret*[*OF - t2s-def(2)*]
        by (*simp-all add*: *ct-list-eq-def to-ct-list-def*)
       hence *test2*:*consume* (*TopType x1*) [*sec-ct sec, sec-ct sec, TSome* (*T-i32 sec*)] = (*TopType x1′*)
        using *TopType ct-list-eq-def ct-suffix-def x1-split*
        by *auto*
      have *produce* (*TopType x1′*) (*select-return-top sec* (*x1′ @ [xa, xb, xc]*) ((*x1′ @ [xa, xb, xc]*) ! *Suc* (*length x1′*)) *xa*) = *select-return-top sec* (*x1′ @ [xa, xb, xc]*) *xb xa*
       proof −

note *a1* = *x1-split*

note *a2* = *select-return-top-ens-sec-ct*

have *f3*: *x1* ! *Suc* (*length x1′*) = *xb*

using *a1* **by** (*metis* (*no-types*) *append.left-neutral append-Cons append-assoc length-append-singleton nth-append-length*)

have *select-return-top sec x1 xb xa* = *Bot* ⟶ *produce* (*TopType x1′*) (*select-return-top sec x1 xb xa*) = *select-return-top sec x1 xb xa*

using *produce.simps*(*8*) **by** *presburger*

then show *produce* (*TopType x1′*) (*select-return-top sec* (*x1′* @ [*xa*, *xb*, *xc*]) ((*x1′* @ [*xa*, *xb*, *xc*]) ! *Suc* (*length x1′*)) *xa*) = *select-return-top sec* (*x1′* @ [*xa*, *xb*, *xc*]) *xb xa*

using *f3 a2 a1* **by** *fastforce*

qed

hence *sel-is2*:*type-update-select sec* (*TopType x1*) = (*select-return-top sec x1 xb xa*)

using *sel-is x1-split test2*

by *auto*

have *top-one*:(*select-return-top sec x1 xb xa*) = *TopType* (*x1′* @ [*ens-sec-ct sec xa*]) ∨ (*select-return-top sec x1 xb xa*) = *TopType* (*x1′* @ [*ens-sec-ct sec xb*])

using *x1-split test3*

apply (*cases xa*; *cases xb*)

apply (*fastforce+*)[*5*]

apply *simp-all*

apply (*metis can-secret-ct.simps*(*1*) *ct-eq.simps*(*4*) *ct-eq-common-tsome ct-eq-commute ct-list-eq-def list.simps*(*11*) *list.simps*(*9*) *test4 to-ct-list-def*)

apply *fastforce*

apply (*metis can-secret-ct.simps*(*1*) *ct-eq.simps*(*4*) *ct-eq-common-tsome ct-eq-commute ct-list-eq-def list.simps*(*11*) *list.simps*(*9*) *test4 to-ct-list-def*)

using *ct-list-eq-def test4 to-ct-list-def*

by *auto*

have *ct-eq* (*ens-sec-ct sec xb*) (*TSome t*) *ct-eq* (*ens-sec-ct sec xa*) (*TSome t*)

using *ens-sec-ct-imp-ct-eq-sec t2s-def*(*2*) *test4*

by (*simp-all add*: *ct-list-eq-def to-ct-list-def*)

hence *c-types-agree* (*TopType* (*x1′* @ [*ens-sec-ct sec xa*])) *t2s*

    c-types-agree (*TopType* (*x1′* @ [*ens-sec-ct sec xb*])) *t2s*

using *test4 t2s-def*(*1*)

by (*simp-all add*: *ct-suffix-unfold-one to-ct-list-def*)

thus *?thesis*

using *sel-is2 outer-2 TopType x1-split top-one*

by *auto*

    qed

  qed *simp-all*

next

  case *3*

  then obtain *cons prods* **where** *c-s-def*:*check-single C e* = (*λctn. type-update ctn cons prods*)

    by *blast*

  hence *ctm-def*:*ctm* = *type-update* (*Type t1s*) *cons prods*

using *assms*(*1*)
by *fastforce*
hence *cons-suffix*:*ct-suffix cons* (*to-ct-list t1s*)
using *assms*
by (*simp*, *metis* (*full-types*) *produce.simps*(*6*))
hence *t-int-def*:*consume* (*Type t1s*) *cons* = (*Type* (*take* (*length t1s* − *length cons*) *t1s*))
using *ctm-def*
by *simp*
hence *ctm-def2*:*ctm* = *produce* (*Type* (*take* (*length t1s* − *length cons*) *t1s*)) *prods*
using *ctm-def*
by *simp*
show *?thesis*
proof (*cases ctn*)
case (*TopType x1*)
hence *ct-suffix x1* (*to-ct-list t1s*)
using *assms*(*3*)
by *simp*
thus *?thesis*
using *assms*(*2*) *ctm-def2*
proof (*cases prods*)
case (*TopType x1*)
thus *?thesis*
using *consume-c-types-agree*[*OF t-int-def assms*(*3*)] *ctm-def2 assms*(*4*) *c-s-def*
by (*metis c-types-agree.elims*(*2*) *produce.simps*(*3*,*4*) *type-update.simps*)
next
case (*Type x2*)
hence *ctm-def3*:*ctm* = *Type* ((*take* (*length t1s* − *length cons*) *t1s*)@ *x2*)
using *ctm-def2*
by *simp*
have *ct-suffix x1 cons* ∨ *ct-suffix cons x1*
using *ct-suffix-shared assms*(*3*) *TopType cons-suffix*
by *auto*
thus *?thesis*
proof (*rule disjE*)
assume *ct-suffix x1 cons*
hence *consume* (*TopType x1*) *cons* = *TopType* []
by (*simp add*: *ct-suffix-length*)
hence *check-single C e ctn* = *TopType* (*to-ct-list x2*)
using *c-s-def TopType Type*
by *simp*
thus *?thesis*
using *TopType ctm-def3 assms*(*4*) *c-types-agree-top2 ct-list-eq-refl*
by *auto*
next
assume *ct-suffix cons x1*
hence *4*:*consume* (*TopType x1*) *cons* = *TopType* (*take* (*length x1* − *length*

cons) *x1*)
   by (*simp add*: *ct-suffix-length*)
   hence *3*:*check-single* $\mathcal{C}$ *e ctn* = *TopType* ((*take* (*length x1* − *length cons*
) *x1*) @ *to-ct-list x2*)
    using *c-s-def TopType Type*
    by *simp*
   have ((*take* (*length t1s* − *length cons*) *t1s*) @ *x2*) = *t2s*
    using *assms*(*4*) *ctm-def3*
    by *simp*
   have *c-types-agree* (*TopType* (*take* (*length x1* − *length cons*) *x1*)) (*take*
(*length t1s* − *length cons*) *t1s*)
    using *consume-c-types-agree*[*OF t-int-def assms*(*3*)] *4 TopType*
    by *simp*
   hence *c-types-agree* (*TopType* (*take* (*length x1* − *length cons*) *x1* @
to-ct-list x2)) (*take* (*length t1s* − *length cons*) *t1s* @ *x2*)
    unfolding *c-types-agree.simps to-ct-list-def*
    by (*simp add*: *ct-suffix-cons2 ct-suffix-cons-it ct-suffix-extend-ct-list-eq*)
   thus *?thesis*
    using *ctm-def3 assms 3*
    by *simp*
  qed
  qed *simp*
 next
  case (*Type x2*)
  thus *?thesis*
   using *assms*
   by *simp*
 next
  case *Bot*
  thus *?thesis*
   using *assms*
   by *simp*
 qed
 qed
qed

lemma *b-e-type-checker-compose*:
 assumes *b-e-type-checker* $\mathcal{C}$ *es* (*t1s -> t2s*)
   b-e-type-checker $\mathcal{C}$ [*e*] (*t2s -> t3s*)
 shows *b-e-type-checker* $\mathcal{C}$ (*es* @ [*e*]) (*t1s -> t3s*)
proof −
 have *c-types-agree* (*check-single* $\mathcal{C}$ *e* (*Type t2s*)) *t3s*
  using *assms*(*2*)
  by *simp*
 then obtain *ctm* **where** *ctm-def*:*check-single* $\mathcal{C}$ *e* (*Type t2s*) = *ctm*
      c-types-agree ctm t3s
      ctm ≠ *Bot*
  by *fastforce*
 have *c-types-agree* (*check* $\mathcal{C}$ *es* (*Type t1s*)) *t2s*

195

    using *assms(1)*
    by *simp*
  then obtain *ctn* **where** *ctn-def*:*check C es (Type t1s) = ctn*
                        c-types-agree ctn t2s
                        ctn ≠ Bot
    by *fastforce*
  thus *?thesis*
    using *check-single-imp-weakening[OF ctm-def(1,3) ctn-def(2) ctm-def(2)]*
         check-unfold-snoc[of C es (Type t1s) e]
    by *simp*
qed

lemma *b-e-check-single-type-type*:
  assumes *check-single C e xs = (Type tm)*
  shows *∃ tn. xs = (Type tn)*
proof −
  consider *(1) check-single C e = id*
       | (2) ∃ sec. check-single C e = (λctn. type-update-select sec ctn)
       | (3) (∃ cons prods. (check-single C e = (λctn. type-update ctn cons prods)))
    using *check-single-imp assms*
    by *blast*
  thus *?thesis*
  proof *(cases)*
    case *1*
    thus *?thesis*
      using *assms*
      by *simp*
    next
      case *2*
      note *outer-2 = 2*
      thus *?thesis*
        using *assms*
      proof *(cases xs)*
        case *(TopType x1)*
        consider
           (1) length x1 = 0
          | (2) length x1 = Suc 0
          | (3) length x1 = Suc (Suc 0)
          | (4) length x1 ≥ 3
        by *linarith*
        thus *?thesis*
        proof *cases*
          case *1*
          thus *?thesis*
            using *assms 2 TopType*
            by *auto*
          next
           case *2*
           thus *?thesis*

      using *assms outer-2 TopType produce-type-type*
      by *fastforce*
    next
      case *3*
      thus *?thesis*
        using *assms 2 TopType numerals(2) type-update-select-length2*
        by *force*
    next
      case *4*
      then obtain *nat* **where** *nat-def*:*length x1 = Suc (Suc (Suc nat))*
        by (*metis add-eq-if diff-Suc-1 le-Suc-ex numeral-3-eq-3 nat.distinct(2)*)
      thus *?thesis*
        using *assms 2 TopType produce-type-type*
        by (*fastforce split*: *if-splits*)
    qed
  qed *auto*
next
  case *3*
  then obtain *cons prods* **where** *check-def*:*check-single C e = (λctn. type-update ctn cons prods)*
    by *blast*
  hence *produce (consume xs cons) prods = (Type tm)*
    using *assms(1)*
    by *simp*
  thus *?thesis*
    using *assms check-def consume-type-type produce-type-type*
    by *blast*
qed
qed

lemma *b-e-check-single-weaken-type*:
  assumes *check-single C e (Type tn) = (Type tm)*
  shows *check-single C e (Type (ts@tn)) = Type (ts@tm)*
proof −
  consider (*1*) *check-single C e = id*
      | (*2*) *∃sec. check-single C e = (λctn. type-update-select sec ctn)*
     | (*3*) (*∃cons prods. (check-single C e = (λctn. type-update ctn cons prods)*)))
    using *check-single-imp assms*
    by *blast*
  thus *?thesis*
  proof (*cases*)
    case *1*
    thus *?thesis*
      using *assms(1)*
      by *simp*
  next
    case *2*
    then obtain *sec* **where** *2*:*check-single C e = type-update-select sec*
      by *blast*

hence *cond*:(*length tn* ≥ *3* ∧ (*tn*!(*length tn*−*2*)) = (*tn*!(*length tn*−*3*)))
  using *assms*
  by (*metis checker-type.distinct*(*5*) *type-update-select.simps*(*1*))
hence *consume* (*Type tn*) [*TAny*, *TSome* (*T-i32 sec*)] = (*Type tm*)
  using *assms 2*
  by (*metis checker-type.simps*(*8*) *type-update-select.simps*(*1*))
hence *consume* (*Type* (*ts@tn*)) [*TAny*, *TSome* (*T-i32 sec*)] = (*Type* (*ts@tm*))
  using *consume-weaken-type*
  by *blast*
moreover
have (*length* (*ts@tn*) ≥ *3* ∧ ((*ts@tn*)!(*length* (*ts@tn*)−*2*)) = ((*ts@tn*)!(*length*
(*ts@tn*)−*3*)))
  using *cond*
   by (*simp, metis add.commute add-leE nth-append-length-plus numeral-Bit1
numeral-One*
            one-add-one ordered-cancel-comm-monoid-diff-class.diff-add-assoc2)
ultimately
show *?thesis*
  using *2*
  by (*simp split: if-splits*) (*metis Nat.add-diff-assoc assms checker-type.distinct*(*5*)
nth-append-length-plus type-update-select.simps(*1*))
next
  case *3*
  then obtain *cons prods* **where** *check-def*:*check-single* 𝒞 *e* = (λ*ctn. type-update
ctn cons prods*)
  by *blast*
  hence *produce* (*consume* (*Type tn*) *cons*) *prods* = (*Type tm*)
  using *assms*(*1*)
  by *simp*
  then obtain *t-int* **where** *t-int-def*:*consume* (*Type tn*) *cons* = (*Type t-int*)
  by (*metis consume.simps*(*1*) *produce.simps*(*6*))
  thus *?thesis*
  using *assms*(*1*) *check-def*
       consume-weaken-type[*OF t-int-def, of ts*]
       produce-weaken-type[*of t-int prods tm ts*]
  by *simp*
 qed
qed

lemma *b-e-check-single-weaken-top*:
 assumes *check-single* 𝒞 *e* (*Type tn*) = *TopType tm*
 shows *check-single* 𝒞 *e* (*Type* (*ts@tn*)) = *TopType tm*
proof −
 consider (*1*) *check-single* 𝒞 *e* = *id*
      | (*2*) ∃ *sec. check-single* 𝒞 *e* = (λ*ctn. type-update-select sec ctn*)
      | (*3*) (∃ *cons prods.* (*check-single* 𝒞 *e* = (λ*ctn. type-update ctn cons prods*)))
  using *check-single-imp assms*
  by *blast*
 thus *?thesis*

198

proof (*cases*)
  case *1*
   thus *?thesis*
     using *assms*
     by *simp*
  next
   case *2*
   thus *?thesis*
     using *assms*
     by (*metis checker-type.simps(4) type-update-select-top-exists*)
  next
   case *3*
   then obtain *cons prods* **where** *check-def*:*check-single C e = (λctn. type-update ctn cons prods)*
     by *blast*
   hence *produce (consume (Type tn) cons) prods = (TopType tm)*
     using *assms(1)*
     by *simp*
   moreover
   then obtain *t-int* **where** *t-int-def*:*consume (Type tn) cons = (Type t-int)*
     by (*metis checker-type.distinct(3) consume.simps(1) produce.simps(6)*)
   ultimately
   show *?thesis*
     using *check-def consume-weaken-type*
     by (*cases prods*, *auto*)
  qed
qed

lemma *b-e-check-weaken-type*:
  assumes *check C es (Type tn) = (Type tm)*
  shows *check C es (Type (ts@tn)) = (Type (ts@tm))*
  using *assms*
proof (*induction es arbitrary*: *tn tm rule*: *List.rev-induct*)
  case *Nil*
  thus *?case*
    by *simp*
next
  case (*snoc e es*)
  hence *check-single C e (check C es (Type tn)) = Type tm*
    using *check-unfold-snoc*[*OF check-neq-bot-snoc*]
    by (*metis checker-type.distinct(5)*)
  thus *?case*
    using *b-e-check-single-weaken-type b-e-check-single-type-type snoc*
    by (*metis check-unfold-snoc checker-type.distinct(5)*)
qed

lemma *check-bot*: *check C es Bot = Bot*
  by (*simp add*: *list.case-eq-if*)

lemma *b-e-check-weaken-top*:
  assumes *check C es (Type tn) = (TopType tm)*
  shows *check C es (Type (ts@tn)) = (TopType tm)*
  using *assms*
proof (*induction es arbitrary: tn tm*)
  case *Nil*
  thus *?case*
    by *simp*
next
  case (*Cons e es*)
  show *?case*
  proof (*cases (check-single C e (Type tn))*)
    case (*TopType x1*)
    hence *check-single C e (Type (ts@tn)) = TopType x1*
      using *b-e-check-single-weaken-top*
      by *blast*
    thus *?thesis*
      using *TopType Cons*
      by *simp*
  next
    case (*Type x2*)
    hence *check-single C e (Type (ts@tn)) = Type (ts@x2)*
      using *b-e-check-single-weaken-type*
      by *blast*
    thus *?thesis*
      using *Cons Type*
      by *fastforce*
  next
    case *Bot*
    thus *?thesis*
      using *check-bot Cons*
      by *simp*
  qed

qed

lemma *b-e-type-checker-weaken*:
  assumes *b-e-type-checker C es (t1s -> t2s)*
  shows *b-e-type-checker C es (ts@t1s -> ts@t2s)*
proof −
  have *c-types-agree (check C es (Type t1s)) t2s*
    using *assms(1)*
    by *simp*
  then obtain *ctn* **where** *ctn-def:check C es (Type t1s) = ctn*
                     c-types-agree ctn t2s
                     ctn ≠ Bot
    by *fastforce*
  show *?thesis*
  proof (*cases ctn*)

case (*TopType x1*)
thus *?thesis*
  using *ctn-def(1,2) b-e-check-weaken-top*[*of C es t1s x1 ts*]
    by (*metis append-assoc b-e-type-checker.simps c-types-agree-imp-ct-list-eq c-types-agree-top2*)
next
  case (*Type x2*)
  thus *?thesis*
    using *ctn-def(1,2) b-e-check-weaken-type*[*of C es t1s x2 ts*]
    by *simp*
next
  case *Bot*
  thus *?thesis*
    using *ctn-def(3)*
    by *simp*
qed
qed

lemma *b-e-type-checker-complete*:
  assumes $C \vdash es : (tn$ -> *tm*)
  shows *b-e-type-checker C es* (*tn* -> *tm*)
  using *assms*
proof (*induction es* (*tn* -> *tm*) *arbitrary*: *tn tm rule*: *b-e-typing.induct*)
  case (*select sec t C*)
  have *ct-list-eq* [*TAny, TSome* (*T-i32 sec*)] [*TSome t, TSome* (*T-i32 sec*)]
    by (*simp add*: *to-ct-list-def ct-list-eq-def*)
  thus *?case*
   using *select ct-suffix-extend-ct-list-eq*[*OF ct-suffix-nil*[*of* [*TSome t*]]] *to-ct-list-def*
    by *auto*
next
  case (*br-table C ts is i t1s t2s*)
  show *?case*
    using *list-all-conv-same-lab*[*OF br-table*]
    by (*auto simp add*: *to-ct-list-def ct-suffix-nil ct-suffix-cons-it*)
next
  case (*set-global i C t*)
  thus *?case*
    using *to-ct-list-def ct-suffix-refl is-mut-def tg-t-def*
    by *auto*
next
  case (*composition C es t1s t2s e t3s*)
  thus *?case*
    using *b-e-type-checker-compose*
    by *simp*
next
  case (*weakening C es t1s t2s ts*)
  thus *?case*
    using *b-e-type-checker-weaken*
    by *simp*

qed (*auto simp add*: *to-ct-list-def ct-suffix-refl ct-suffix-nil ct-suffix-cons-it
ct-suffix-singleton-any*)

theorem *b-e-typing-equiv-b-e-type-checker*:
  shows $(\mathcal{C} \vdash es : (tn \text{ -> } tm)) = (b\text{-}e\text{-}type\text{-}checker\ \mathcal{C}\ es\ (tn \text{ -> } tm))$
  using *b-e-type-checker-sound b-e-type-checker-complete*
  by *blast*

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′)*
  using *assms*
  unfolding *memory-public-agree-def*
  by *auto*

lemma *store-public-agree-smem-ind-eq*:
  assumes *store-public-agree s s′*
  shows *(smem-ind s i) = (smem-ind s′ i)*
  using *assms*
  unfolding *store-public-agree-def smem-ind-def*
  by *fastforce*

lemma *store-public-agree-sfunc-eq*:
  assumes *store-public-agree s s′*
  shows *(sfunc s i j) = (sfunc s′ i j)*
  using *assms*
  unfolding *store-public-agree-def sfunc-def sfunc-ind-def*
  by *fastforce*

lemma *store-public-agree-stab-eq*:
  assumes *store-public-agree s s′*
  shows *(stab s i j) = (stab s′ i j)*
  using *assms*
  unfolding *store-public-agree-def stab-def stab-s-def*
  by *presburger*

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)*
  using *assms*
  unfolding *store-public-agree-def sglob-ind-def*

by *fastforce*

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*)
  using *assms list-all2-nthD*
  unfolding *store-public-agree-def global-public-agree-def sglob-val-def sglob-def sglob-ind-def*
  by *fastforce*

lemma *store-public-agree-stypes-eq*:
  assumes *store-public-agree s s′*
  shows (*stypes s i j*) = (*stypes s′ i j*)
  using *assms*
  unfolding *store-public-agree-def stypes-def*
  by *fastforce*

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*
  using *assms store-public-agree-stab-eq store-public-agree-stypes-eq*
  by *fastforce*

lemma *public-agree-imp-typeof*:
  assumes *public-agree v v′*
  shows *typeof v* = *typeof v′*
  using *assms*
  unfolding *public-agree-def*
  by *auto*

lemma *not-typeof-imp-no-public-agree*:
  assumes *typeof v* ≠ *typeof v′*
  shows ¬*public-agree v v′*
  using *assms*
  unfolding *public-agree-def*
  by *auto*

lemma *publics-agree-imp-typeof*:
  assumes *publics-agree vs vs′*
  shows *map typeof vs* = *map typeof vs′*
  using *assms*
proof (*induction vs arbitrary*: *vs′*)
  case *Nil*
  thus *?case*
    by *blast*
next

case (*Cons a vs*)
thus *?case*
  using *public-agree-imp-typeof*
  by (*metis list.simps(9) list-all2-Cons1*)
qed

lemma *public-agree-imp-types-agree-insecure*:
  assumes *types-agree-insecure t v*
          public-agree v v′
  shows *types-agree-insecure t v′*
  using *assms public-agree-imp-typeof*
  unfolding *types-agree-insecure-def*
  by *fastforce*

lemma *public-agree-imp-types-agree*:
  assumes *types-agree t v*
          public-agree v v′
  shows *types-agree t v′*
  using *assms public-agree-imp-typeof*
  unfolding *types-agree-def*
  by *fastforce*

lemma *publics-agree-nil1*:
  assumes *publics-agree [] vs*
  shows *vs = []*
  using *assms*
  by *simp*

lemma *publics-agree-nil2*:
  assumes *publics-agree vs []*
  shows *vs = []*
  using *assms*
  by *simp*

lemma *public-agree-refl*: *public-agree v v*
  by (*simp add*: *public-agree-def*)

lemma *public-agree-public-i32*:
  assumes *public-agree* (*ConstInt32 sec c*) *v*
  shows $\exists c.\ v =$ (*ConstInt32 sec c*)
  using *assms*
  by (*cases v*; *auto simp add*: *public-agree-def typeof-def*)

lemma *public-agree-public-i64*:
  assumes *public-agree* (*ConstInt64 sec c*) *v*
  shows $\exists c.\ v =$ (*ConstInt64 sec c*)
  using *assms*
  by (*cases v*; *auto simp add*: *public-agree-def typeof-def*)

lemma *public-agree-public-f32*:
  assumes *public-agree* (*ConstFloat32 c*) *v*
  shows $\exists c.\ v = (ConstFloat32\ c)$
  using *assms*
  by (*cases v*; *auto simp add*: *public-agree-def typeof-def*)

lemma *public-agree-public-f64*:
  assumes *public-agree* (*ConstFloat64 c*) *v*
  shows $\exists c.\ v = (ConstFloat64\ c)$
  using *assms*
  by (*cases v*; *auto simp add*: *public-agree-def typeof-def*)

lemma *publics-agree-refl*: *publics-agree vs vs*
  using *public-agree-refl*
  by (*fastforce simp add*: *list-all2-refl*)

lemma *publics-agree1*:
  assumes *publics-agree* [*v*] *es′*
  shows $\exists v'.\ es' = [v']$
  using *assms*
  by (*metis list-all2-Cons1 publics-agree-nil1*)

lemma *publics-agree-secret1*:
  assumes *publics-agree* [*v*] *es′*
        t-sec (*typeof v*) = *Public*
  shows $es' = [v]$
  using *assms*
  unfolding *public-agree-def*
  by (*simp add*: *list-all2-Cons1*)

lemma *publics-agree-public1*:
  assumes *publics-agree* [*v*] *es′*
        t-sec (*typeof v*) = *Public*
  shows $\exists v'.\ es' = [v'] \land public\text{-}agree\ v\ v'$
  using *assms*
  unfolding *public-agree-def*
  by (*simp add*: *list-all2-Cons1*)

lemma *memories-public-agree-refl*: *memories-public-agree ms ms*
  unfolding *memory-public-agree-def*
  by (*simp add*: *list-all2-refl*)

lemma *globals-public-agree-refl*: *globals-public-agree gs gs*
  unfolding *global-public-agree-def public-agree-def*
  by (*simp add*: *list-all2-refl*)

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

lemma *exprs-public-agree-refl*: *exprs-public-agree es es*

by (*simp add*: *expr-public-agree-refl list-all2-refl*)

lemma *list-all2-symm*:
  assumes *list-all2 P xs ys*
        $(\bigwedge x\ y.\ P\ x\ y \implies P\ y\ x)$
  shows *list-all2 P ys xs*
  using *assms*
  by (*simp add*: *list-all2-conv-all-nth*)

lemma *public-agree-symm*:
  assumes *public-agree v v′*
  shows *public-agree v′ v*
  using *assms*
  unfolding *public-agree-def*
  by *auto*

lemma *public-agree-trans*:
  assumes *public-agree v v′*
       public-agree v′ v″
  shows *public-agree v v″*
  using *assms*
  unfolding *public-agree-def*
  by *auto*

lemma *equivp-public-agree*:*equivp public-agree*
  unfolding *equivp-def public-agree-def*
  by *metis*

lemma *publics-agree-trans*:
  assumes *publics-agree vs vs′*
       publics-agree vs′ vs″
  shows *publics-agree vs vs″*
  using *assms list.rel-transp equivp-public-agree*
  unfolding *transp-def equivp-reflp-symp-transp*
  by *blast*

lemma *memories-public-agree-symm*:
  assumes *memories-public-agree ms ms′*
  shows *memories-public-agree ms′ ms*
  using *list-all2-symm assms*
  unfolding *memory-public-agree-def*
  by *fastforce*

lemma *globals-public-agree-symm*:
  assumes *globals-public-agree gs gs′*
  shows *globals-public-agree gs′ gs*
  using *list-all2-symm assms public-agree-symm*
  unfolding *global-public-agree-def*
  by (*metis* (*mono-tags*, *lifting*))

lemma *transp-memory-public-agree*:*transp memory-public-agree*
  unfolding *transp-def memory-public-agree-def*
  by *fastforce*

lemma *memories-public-agree-trans*:
  assumes *memories-public-agree ms ms′*
          memories-public-agree ms′ ms″
  shows *memories-public-agree ms ms″*
  using *assms list.rel-transp*[*OF transp-memory-public-agree*]
  unfolding *transp-def*
  by *metis*

lemma *transp-global-public-agree*:*transp global-public-agree*
  using *equivp-public-agree*
  unfolding *transp-def global-public-agree-def equivp-reflp-symp-transp*
  by *metis*

lemma *globals-public-agree-trans*:
  assumes *globals-public-agree gs gs′*
          globals-public-agree gs′ gs″
  shows *globals-public-agree gs gs″*
  using *assms list.rel-transp*[*OF transp-global-public-agree*]
  unfolding *transp-def*
  by *metis*

lemma *equivp-memories-public-agree*: *equivp memories-public-agree*
  using *memories-public-agree-refl memories-public-agree-symm memories-public-agree-trans*
        equivpI
  unfolding *reflp-def symp-def transp-def*
  by *blast*

lemma *equivp-globals-public-agree*: *equivp globals-public-agree*
  using *globals-public-agree-refl globals-public-agree-symm globals-public-agree-trans*
        equivpI
  unfolding *reflp-def symp-def transp-def*
  by *blast*

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*
  using *assms*
  by (*simp add*: *list-all2-conv-all-nth*)

lemma *publics-agree-symm*:
  assumes *publics-agree vs vs′*
  shows *publics-agree vs′ vs*
  using *public-agree-symm list-all2-symm*[*OF assms*]
  by *fastforce*

lemma *expr-public-agree-symm*:
  assumes *expr-public-agree e e'*
  shows *expr-public-agree e' e*
  using *assms*
proof (*induction rule*: *expr-public-agree.induct*)
  case (*1 e*)
  thus *?case*
    using *expr-public-agree-refl*
    by *blast*
next
  case (*2 v v'*)
  thus *?case*
    using *public-agree-symm expr-public-agree.intros(2)*
    by *auto*
next
  case (*3 bes bes' tf*)
  show *?case*
    using *expr-public-agree.intros(3) list-all2-mono[OF list-all2-flip-args[OF 3(1)]]*
    by *fastforce*
next
  case (*4 bes bes' tf*)
  show *?case*
    using *expr-public-agree.intros(4) list-all2-mono[OF list-all2-flip-args[OF 4(1)]]*
    by *fastforce*
next
  case (*5 bes1 bes1' bes2 bes2' tf*)
  show *?case*
    using *expr-public-agree.intros(5) list-all2-mono[OF list-all2-flip-args] 5(1,2)*
    by *fastforce*
next
  case (*6 les les' es es' n*)
  show *?case*
    using *expr-public-agree.intros(6)*
        list-all2-mono[OF list-all2-flip-args[OF 6(1)]]
        list-all2-mono[OF list-all2-flip-args[OF 6(2)]]
    by *fastforce*
next
  case (*7 vs vs' es es' n iforce*)
  show *?case*
    using *expr-public-agree.intros(7) list-all2-mono[OF list-all2-flip-args[OF 7(2)]]*
        publics-agree-symm[OF 7(1)]
    by *fastforce*
qed

lemma *exprs-public-agree-symm*:
  assumes *exprs-public-agree es es'*
  shows *exprs-public-agree es' es*
  using *assms list-all2-symm expr-public-agree-symm*

by *blast*

lemma *store-public-agree-refl*: *store-public-agree s s*
  using *memories-public-agree-refl globals-public-agree-refl*
  unfolding *store-public-agree-def*
  by *simp*

lemma *store-public-agree-symm*:
  assumes *store-public-agree s s′*
  shows *store-public-agree s′ s*
  using *assms memories-public-agree-symm globals-public-agree-symm*
  unfolding *store-public-agree-def*
  by *simp*

lemma *store-public-agree-trans*:
  assumes *store-public-agree s s′*
          store-public-agree s′ s″
  shows *store-public-agree s s″*
  using *assms memories-public-agree-trans globals-public-agree-trans*
  unfolding *store-public-agree-def*
  by *metis*

lemma *expr-public-agree-imp-public-agree*:
  assumes *expr-public-agree ($C v) e*
  shows $\exists v'. e = (\$C\ v') \land public\text{-}agree\ v\ v'$
  using *assms expr-public-agree.simps public-agree-refl*
  by *auto*

lemma *expr-public-agree-block*:
  assumes *expr-public-agree ($Block tf es) les*
  shows $\exists es'.\ les = (\$Block\ tf\ es') \land exprs\text{-}public\text{-}agree\ (\$*es)\ (\$*es')$
  using *assms publics-agree-refl exprs-public-agree-refl*
  by (*fastforce simp add*: *expr-public-agree.simps*)

lemma *expr-public-agree-loop*:
  assumes *expr-public-agree ($Loop tf es) les*
  shows $\exists es'.\ les = (\$Loop\ tf\ es') \land exprs\text{-}public\text{-}agree\ (\$*es)\ (\$*es')$
  using *assms publics-agree-refl exprs-public-agree-refl*
  by (*fastforce simp add*: *expr-public-agree.simps*)

lemma *expr-public-agree-if*:
  assumes *expr-public-agree ($If tf es1 es2) les*
  shows $\exists es1'\ es2'.\ les = (\$If\ tf\ es1'\ es2') \land exprs\text{-}public\text{-}agree\ (\$*es1)\ (\$*es1')$
$\land exprs\text{-}public\text{-}agree\ (\$*es2)\ (\$*es2')$
  using *assms publics-agree-refl exprs-public-agree-refl*
  by (*fastforce simp add*: *expr-public-agree.simps*)

lemma *expr-public-agree-local*:
  assumes *expr-public-agree (Local n i vs es) les*

shows $\exists \, vs' \, es'. \; les = (Local \; n \; i \; vs' \; es') \wedge publics\text{-}agree \; vs \; vs' \wedge exprs\text{-}public\text{-}agree$
$es \; es'$
  using *assms publics-agree-refl exprs-public-agree-refl*
  by (*fastforce simp add*: *expr-public-agree.simps*)

lemma *expr-public-agree-label*:
  assumes *expr-public-agree* (*Label n les es*) *e'*
  shows $\exists \, les' \, es''. \; e' = (Label \; n \; les' \; es'') \wedge exprs\text{-}public\text{-}agree \; les \; les' \wedge exprs\text{-}public\text{-}agree$
$es \; es''$
  using *assms publics-agree-refl exprs-public-agree-refl*
  by (*fastforce simp add*: *expr-public-agree.simps*)

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 $\exists \, b\text{-}e2. \; e2 = \$b\text{-}e2$
  using *assms*
  by (*fastforce simp add*: *expr-public-agree.simps*)

lemma *exprs-public-agree-imp-expr-public-agree*:
  assumes *exprs-public-agree* [*e1*] [*e2*]
  shows *expr-public-agree e1 e2*
  using *assms*
  by *auto*

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 $\exists \, v' \, es''. \; es' = ((\$C \; v')\#es'') \wedge public\text{-}agree \; v \; v' \wedge exprs\text{-}public\text{-}agree \; es$
es''
  using *assms list-all2-Cons1*[*of expr-public-agree*] *expr-public-agree-imp-public-agree*
  by *fastforce*

lemma *exprs-public-agree-imp-publics-agree*:
  assumes *exprs-public-agree* ((*$$∗ ves*)@*es*) *es'*
  shows $\exists \, ves' \, es''. \; es' = ((\$\$∗ \; ves')@es'') \wedge publics\text{-}agree \; ves \; ves' \wedge exprs\text{-}public\text{-}agree$
$es \; es''$
  using *assms*
proof (*induction ves arbitrary*: *es'*)
  case *Nil*
  thus *?case*
    using *publics-agree-refl*
    by *auto*
next
  case (*Cons a ves*)
  obtain *b es''* **where** *es''-def*:*es'* = (*$C b*)#*es'' public-agree a b exprs-public-agree*

210

```
((($$* ves) @ es) es''
    using Cons(2) exprs-public-agree-imp-publics-agree-cons
    by fastforce
 moreover
 obtain ves' es''' where es'' = ($$* ves') @ es''' publics-agree ves ves' exprs-public-agree
es es'''
    using Cons(1)[OF es''-def(3)]
    by blast
 ultimately
 have es' = ($$* b#ves') @ es''' ∧ publics-agree (a#ves) (b#ves') ∧ exprs-public-agree
es es'''
    by (simp)
 thus ?case
    by blast
qed

lemma exprs-public-agree-imp-publics-agree1 :
 assumes exprs-public-agree (($$* ves)@[e]) es'
 shows ∃ ves' e'. es' = (($$* ves')@[e']) ∧ publics-agree ves ves' ∧ expr-public-agree
e e'
    using exprs-public-agree-imp-publics-agree[OF assms]
    by (metis list-all2-Cons1 list-all2-Nil)

lemma exprs-public-agree-imp-publics-agree1-const0 :
 assumes exprs-public-agree [e] es'
 shows ∃ e'. es' = [e'] ∧ expr-public-agree e e'
    using exprs-public-agree-imp-publics-agree1[of [] e es'] assms
    by fastforce

lemma b-e-exprs-public-agree-imp-publics-agree1-const0 :
 assumes exprs-public-agree ($*[b-e]) es'
 shows ∃ b-e'. es' = [$b-e'] ∧ expr-public-agree ($b-e) ($b-e')
proof −
 have exprs-public-agree [$b-e] es'
    using assms
    by simp
 then obtain e' where es' = [e'] expr-public-agree ($b-e) e'
    using exprs-public-agree-imp-publics-agree1-const0
    by blast
 thus ?thesis
    by (cases e') (fastforce simp add: expr-public-agree.simps)+
qed

lemma exprs-public-agree-trap-imp-is-trap :
 assumes exprs-public-agree [Trap] es
 shows es = [Trap]
    using exprs-public-agree-imp-publics-agree1-const0[OF assms]
    by (fastforce simp add: expr-public-agree.simps)
```

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′*
  using *exprs-public-agree-imp-publics-agree1*$[of\ [v]\ e\ es']$ *assms publics-agree1*
exprs-public-agree-imp-publics-agree-cons
 by *fastforce*

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′
 using *exprs-public-agree-imp-publics-agree-cons exprs-public-agree-imp-publics-agree1-const1*
assms
 by *blast*

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′
 using *exprs-public-agree-imp-publics-agree-cons exprs-public-agree-imp-publics-agree1-const2*
assms
 by *blast*

lemma *publics-agree-imp-exprs-public-agree-cons*:
 assumes *public-agree v v′*
       exprs-public-agree es es′
 shows *exprs-public-agree* $((\$C\ v)\#es)\ ((\$C\ v')\#es')$
 using *assms(2) list.rel-intros(2) public-agree-imp-expr-public-agree*$[OF\ assms(1)]$
 by *fastforce*

lemma *publics-agree-imp-exprs-public-agree*:
 assumes *publics-agree ves ves′*
       exprs-public-agree es es′
 shows *exprs-public-agree* $((\$\$*\ ves)@es)\ ((\$\$*\ ves')@es')$
 using *assms*
proof (*induction ves arbitrary*: *ves′*)
 case *Nil*
 thus *?case*
   using *publics-agree-nil1*
   by *fastforce*
next
 case (*Cons a ves*)
 obtain *b ves″* **where** *ves″-def*:*ves′* = *b#ves″ public-agree a b publics-agree ves*
ves″

```
      using Cons(2) list-all2-Cons1 [of public-agree]
      by fastforce
    show ?case
     using Cons(1)[OF ves′′-def (3) Cons(3)] ves′′-def (1,2) publics-agree-imp-exprs-public-agree-cons
      by simp
  qed


lemma expr-public-agree-const:
  assumes expr-public-agree e e′
          is-const e
  shows is-const e′
  using expr-public-agree-imp-public-agree assms e-type-const-unwrap
  unfolding is-const-def
  by fastforce


lemma exprs-public-agree-const-list:
  assumes exprs-public-agree es es′
          const-list es
  shows const-list es′
  using assms
proof (induction es arbitrary: es′)
  case Nil
  thus ?case
  by simp
next
  case (Cons a es)
  thus ?case
    by (metis append-Nil2 exprs-public-agree-imp-publics-agree list-all2-Nil
            e-type-const-conv-vs is-const-list)
qed


lemma exprs-public-agree-basic:
  assumes exprs-public-agree ($∗ ves) es′
  shows ∃ ves′. es′ = ($∗ ves′)
  using assms
proof (induction ves arbitrary: es′)
  case Nil
  thus ?case
    by simp
next
  case (Cons a ves)
  thus ?case
   using list-all2-Cons1 [of expr-public-agree $a $∗ves es′] inj-basic expr-public-agree-basic
    by (metis list.map(2))
qed


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 ∧
```

213

$$\textit{exprs-public-agree vs vs-a} \;\wedge$$
$$\textit{exprs-public-agree es es-a} \;\wedge$$
$$\textit{exprs-public-agree es}' \textit{ es}'\textit{-a}$$

using *list-all2-append1* [*of expr-public-agree*] *assms*
by *fastforce*

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 $-$
  obtain $\mathcal{C}s$ *tfs ns ms tgs* **where** $\mathcal{S}$-*def*:$\mathcal{S}$ = (|*s-inst* = $\mathcal{C}s$, *s-funcs* = *tfs*, *s-tab* = *ns*, *s-mem* = *ms*, *s-globs* = *tgs*|)
    using *s-context.cases*
    by *blast*
  obtain *insts fs tclss bss gs* **where** *s-def*:*s* = (|*s.inst* = *insts*, *s.funcs* = *fs*, *s.tab* = *tclss*, *s.mem* = *bss*, *s.globs* = *gs*|)
    using *s.cases*
    by *blast*
  obtain *insts$'$ fs$'$ tclss$'$ bss$'$ gs$'$* **where** *s$'$-def*:*s$'$* = (|*s.inst* = *insts$'$*, *s.funcs* = *fs$'$*, *s.tab* = *tclss$'$*, *s.mem* = *bss$'$*, *s.globs* = *gs$'$*|)
    using *s.cases*
    by *blast*
  have *list-all2* (*inst-typing $\mathcal{S}$*) *insts$'$ $\mathcal{C}s$*
      list-all2 (*cl-typing $\mathcal{S}$*) *fs$'$ tfs*
      list-all (*tab-agree $\mathcal{S}$*) (*concat tclss$'$*)
      list-all2 (λ *tcls n. n $\leq$ length tcls*) *tclss$'$ ns*
    using *assms $\mathcal{S}$-def s-def s$'$-def*
    unfolding *store-typing.simps store-public-agree-def*
    by *auto*
  moreover
  have *list-all2 mem-agree bss$'$ ms*
  proof $-$
    have *list-all2* (λ (*bs,sec*) (*m,sec$'$*). *m $\leq$ mem-size bs $\wedge$ sec = sec$'$*) *bss ms*
      using *assms*(*1*) *$\mathcal{S}$-def s-def*
      unfolding *store-typing.simps mem-agree-def*
      by *blast*
    moreover
    have *list-all2* (λ (*bs,sec*) (*bs$'$,sec$'$*). *mem-size bs = mem-size bs$'$ $\wedge$ sec = sec$'$*) *bss bss$'$*
      using *s-def s$'$-def assms*(*2*) *list-all2-mono*
      unfolding *store-public-agree-def memory-public-agree-def*
      by *fastforce*
    ultimately
    show *?thesis*
      by (*auto simp add*: *case-prod-beta$'$ list-all2-conv-all-nth mem-agree-def*)
  qed
  moreover
  have *list-all2 glob-agree gs$'$ tgs*

proof −
  have *list-all2 glob-agree gs tgs*
    using *assms(1) S-def s-def*
    unfolding *store-typing.simps*
    by *blast*
  moreover
  have *list-all2 (λx y. g-mut x = g-mut y $\land$ public-agree (g-val x) (g-val y)) gs gs$'$*
    using *s-def s$'$-def assms(2)*
    unfolding *store-public-agree-def global-public-agree-def*
    by *fastforce*
  ultimately
  show *?thesis*
    using *public-agree-imp-typeof*
    unfolding *glob-agree-def*
    by (*auto simp add*: *list-all2-conv-all-nth*)
  qed
  ultimately
  show *?thesis*
    using *S-def s$'$-def store-typing.intros*
    by *blast*
qed

lemma *exprs-public-agree-imp-lholed-public-agree*:
  assumes *Lfilled k lholed es les*
       exprs-public-agree les les$'$
  shows $\exists$ *lholed$'$ es$'$. lholed-public-agree lholed lholed$'$* $\land$
           exprs-public-agree es es$'$ $\land$
           Lfilled k lholed$'$ es$'$ les$'$
  using *assms*
proof (*induction arbitrary*: *les$'$ rule*: *Lfilled.induct*)
  case (*L0 vs lholed es$'$ es*)
  obtain *vs-a es-a es$'$-a* **where** *les$'$-def*:*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
    using *exprs-public-agree-app3[OF L0(3)]*
    by *fastforce*
  have *lholed-public-agree lholed (LBase vs-a es$'$-a)*
    using *L0(2) les$'$-def(2,4) lholed-public-agree.intros(1)*
    by *blast*
  thus *?case*
    using *les$'$-def(1,2,3) Lfilled.intros(1) L0(1) exprs-public-agree-const-list[OF les$'$-def(2)]*
    by *fastforce*
next
  case (*LN vs lholed n es$'$ l es$''$ k es lfilledk*)
  obtain *vs-a les-a es$''$-a* **where** *les$'$-def*:*les$'$ = vs-a @ les-a @ es$''$-a*
                       exprs-public-agree vs vs-a

215

$$exprs\text{-}public\text{-}agree\ [Label\ n\ es'\ lfilledk]\ les\text{-}a$$
$$exprs\text{-}public\text{-}agree\ es''\ es''\text{-}a$$

   using *exprs-public-agree-app3*[*OF LN(5)*]
   by *fastforce*
  obtain *es'-a lfilledk-a* **where** *les-a-def*:*les-a* = [*Label n es'-a lfilledk-a*]
$$exprs\text{-}public\text{-}agree\ es'\ es'\text{-}a$$
$$exprs\text{-}public\text{-}agree\ lfilledk\ lfilledk\text{-}a$$
  using *expr-public-agree-label les'-def(3) exprs-public-agree-imp-publics-agree1-const0*
   by *blast*
  then obtain *l' es-a* **where** *l'-def*:*lholed-public-agree l l'*
$$exprs\text{-}public\text{-}agree\ es\ es\text{-}a$$
$$Lfilled\ k\ l'\ es\text{-}a\ lfilledk\text{-}a$$
   using *LN(4)*
   by *blast*
  hence *lholed-public-agree lholed (LRec vs-a n es'-a l' es''-a)*
   using *LN(2) les'-def(2,4) les-a-def(2) lholed-public-agree.intros(2)*
   by *blast*
  thus *?case*
   using *les-a-def(1) les'-def(1) l'-def(2,3) Lfilled.intros(2)*
     exprs-public-agree-const-list[*OF les'-def(2) LN(1)*]
   by *blast*
qed

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 $\exists\,les'.\ Lfilled\ k\ lholed'\ es'\ les'\ \wedge\ exprs\text{-}public\text{-}agree\ les\ les'$
  using *assms(2,1,3)*
proof (*induction arbitrary*: *es' lholed' rule*: *Lfilled.induct*)
  case (*L0 vs lholed bes es*)
  obtain *vs-a bes-a* **where** *lholed'-def*:*lholed'* = *LBase vs-a bes-a*
$$exprs\text{-}public\text{-}agree\ vs\ vs\text{-}a$$
$$exprs\text{-}public\text{-}agree\ bes\ bes\text{-}a$$
$$const\text{-}list\ vs\text{-}a$$
   using *L0(1,2,3) lholed-public-agree.simps*[*of lholed lholed'*]
     exprs-public-agree-const-list[*of vs*]
   by *fastforce*
  show *?case*
   using *Lfilled.intros(1)*[*OF lholed'-def(4,1), of es'*] *L0(4) lholed'-def(2,3)*
     list-all2-appendI[*of expr-public-agree*]
   by *auto*
next
  case (*LN vs lholed n lres l es'' k es lfilledk*)
  obtain *vs-a lres-a l-a es''-a* **where** *lholed'-def*:*lholed'* = *LRec vs-a n lres-a l-a es''-a*
$$exprs\text{-}public\text{-}agree\ vs\ vs\text{-}a$$
$$exprs\text{-}public\text{-}agree\ lres\ lres\text{-}a$$
$$exprs\text{-}public\text{-}agree\ es''\ es''\text{-}a$$

$$lholed\text{-}public\text{-}agree\ l\ l\text{-}a$$
$$const\text{-}list\ vs\text{-}a$$
    using $LN(1,2,5)$ *lholed-public-agree.simps*[*of lholed lholed'*]
        exprs-public-agree-const-list[*of vs*]
    by *fastforce*
  obtain *lfilledk'* **where** *lfilledk*:*Lfilled k l-a es' lfilledk' exprs-public-agree lfilledk*
lfilledk'
    using $LN(4,6)$ *lholed'-def*(5)
    by *blast*
  have *exprs-public-agree* [*Label n lres lfilledk*] [*Label n lres-a lfilledk'*]
    using *lholed'-def*(3) *lfilledk*(2) *expr-public-agree.intros*(6)
    by *blast*
  thus *?case*
    using *Lfilled.intros*(2)[*OF lholed'-def*(6,1) *lfilledk*(1)] *lholed'-def*(2,4)
        list-all2-appendI[*of expr-public-agree*]
    by *auto*
qed

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

lemma *exprs-public-agree-imp-b-e-typing*:
  assumes $\mathcal{C} \vdash bes : tf$
      exprs-public-agree ($*bes$) ($*bes'$)
  shows $\mathcal{C} \vdash bes' : tf$
  using *assms*(1,2,1)
proof (*induction arbitrary*: *bes' rule*: *b-e-typing.induct*)
  case (*const C v*)
  obtain v' **where** *bes'* = [*C v'*]
            public-agree v v'
   using *exprs-public-agree-imp-publics-agree1-const0*[*of $C v*] *expr-public-agree-imp-public-agree*
       const(1)
    by *fastforce*
  thus *?case*
    using *public-agree-imp-typeof b-e-typing.intros*(1)
    by *fastforce*
next
  case (*block tf tn tm C es*)
  obtain e' **where** *e'-def*:($*bes'$) = [*e'*]
                  expr-public-agree ($Block tf es$) *e'*
     using *block*(4) *exprs-public-agree-imp-publics-agree1-const0*[*of $Block tf es*
$*bes'$]
    by *fastforce*
  obtain es' **where** *e'* = ($Block tf es'$)
          exprs-public-agree ($*es$) ($*es'$)

217

    using *e'-def*(*2*) *exprs-public-agree-refl*[*of* ($∗es*)]
    by (*fastforce simp add*: *expr-public-agree.simps*)
  thus *?case*
    using *block*(*3*)[*OF - block*(*2*)] *e'-def*(*1*) *b-e-typing.block block*(*1*)
    by *fastforce*
next
  case (*loop tf tn tm C es*)
  obtain *e'* **where** *e'-def*:($∗bes'*) = [*e'*]
               expr-public-agree ($Loop tf es*) *e'*
   using *loop*(*4*) *exprs-public-agree-imp-publics-agree1-const0*[*of* $*Loop tf es* $∗bes'*]
   by *fastforce*
  obtain *es'* **where** *e'* = ($Loop tf es'*)
            exprs-public-agree ($∗es*) ($∗es'*)
   using *e'-def*(*2*) *exprs-public-agree-refl*[*of* ($∗es*)]
   by (*fastforce simp add*: *expr-public-agree.simps*)
  thus *?case*
    using *loop*(*3*)[*OF - loop*(*2*)] *e'-def*(*1*) *b-e-typing.loop loop*(*1*)
    by *fastforce*
next
  case (*if-wasm tf tn tm C es1 es2*)
  obtain *e'* **where** *e'-def*:($∗bes'*) = [*e'*]
               expr-public-agree ($If tf es1 es2*) *e'*
   using *if-wasm*(*6*) *exprs-public-agree-imp-publics-agree1-const0*[*of* $*If tf es1 es2*
$∗bes'*]
   by *fastforce*
  obtain *es1'* *es2'* **where** *p*:*e'* = ($If tf es1' es2'*)
              exprs-public-agree ($∗es1*) ($∗es1'*)
              exprs-public-agree ($∗es2*) ($∗es2'*)
    using *e'-def*(*2*) *exprs-public-agree-refl*[*of* ($∗es1*)] *exprs-public-agree-refl*[*of*
($∗es2*)]
   by (*fastforce simp add*: *expr-public-agree.simps*)
  thus *?case*
   using *e'-def*(*1*) *b-e-typing.if-wasm*[*OF if-wasm*(*1*) *if-wasm*(*4*)[*OF - if-wasm*(*2*)]
if-wasm(*5*)[*OF - if-wasm*(*3*)]]
    by *fastforce*
next
  case (*empty C*)
  thus *?case*
    by *simp*
next
  case (*composition C es t1s t2s e t3s*)
  obtain *us vs* **where**
      bes'-def:($∗ bes'*) = (*us* @ *vs*)
             exprs-public-agree ($∗ es*) *us*
             exprs-public-agree ($∗ [*e*]) *vs*
   using *list-all2-append1*[*of expr-public-agree* $∗ *es* $∗[*e*] ($∗ bes'*)] *composition*(*5*)
   by *fastforce*
  have ∃ *b-us*. *us* = $∗*b-us*
    using *bes'-def*(*1*)

218

    apply (*induction us arbitrary*: *bes′*)
     apply *simp*
    apply (*metis* (*no-types, hide-lams*) *Cons-eq-map-D append-Cons list.simps*(*9*))
    done
  then obtain *b-us b-v* **where** *1*:($* *bes′*) = (($*b-us*) @ ($*[b-v*]))
                  exprs-public-agree ($* *es*) ($*b-us*)
                  exprs-public-agree ($* [*e*]) ($*[b-v*])
   using *b-e-exprs-public-agree-imp-publics-agree1-const0*[*OF bes′-def*(*3*)] *bes′-def*
   by (*metis to-e-list-1*)
  thus *?case*
   using *b-e-typing.composition*[*OF composition*(*3*)[*OF 1*(*2*) *composition*(*1*)] *composition*(*4*)[*OF 1*(*3*) *composition*(*2*)]]
       map-injective[*OF - inj-basic, of bes′ b-us* @ [*b-v*]]
   by *fastforce*
next
  case (*weakening* $\mathcal{C}$ *es t1s t2s ts*)
  show *?case*
   using *weakening*(*2*)[*OF weakening*(*3,1*)] *b-e-typing.weakening*
   by *fastforce*
qed *solve-exprs-public-agree-imp-b-e-typing-trivial+*

lemma *exprs-public-agree-imp-e-typing-s-typing*:
  $\mathcal{S}\cdot\mathcal{C} \vdash es : (ts$ -> $ts′) \Longrightarrow$ *exprs-public-agree es es′* $\Longrightarrow \mathcal{S}\cdot\mathcal{C} \vdash es′ : (ts$ -> $ts′)$
  $\mathcal{S}\cdot tr\cdot rs \Vdash$-*i vs;es* : *ts′* $\Longrightarrow$ *publics-agree vs vs′* $\Longrightarrow$ *exprs-public-agree es es′* $\Longrightarrow$
 $\mathcal{S}\cdot tr\cdot rs \Vdash$-*i vs′;es′* : *ts′*
proof (*induction es* (*ts* -> *ts′*) **and** *es ts′ arbitrary*: *ts ts′ es′* **and** *vs′ es′ rule*:
e-typing-s-typing.inducts)
  case (*1* $\mathcal{C}$ *b-es tf* $\mathcal{S}$)
  show *?case*
   using *1*(*2*) *exprs-public-agree-basic*[*OF 1*(*2*)]
      e-typing-s-typing.intros(*1*)[*OF exprs-public-agree-imp-b-e-typing*[*OF 1*(*1*)]]
   by *blast*
next
  case (*2* $\mathcal{S}$ $\mathcal{C}$ *es t2s e*)
  thus *?case*
   using *e-typing-s-typing.intros*(*2*)
   by (*metis* (*full-types*) *e-type-comp-conc list-all2-append1*)
next
  case (*3* $\mathcal{S}$ $\mathcal{C}$ *es t1s t2s ts*)
  thus *?case*
   using *e-typing-s-typing.intros*(*3*)
   by *blast*
next
  case (*4* $\mathcal{S}$ $\mathcal{C}$ *tf*)
  thus *?case*
   using *e-typing-s-typing.intros*(*4*) *exprs-public-agree-trap-imp-is-trap*
   by *blast*
next
  case (*5* $\mathcal{S}$ $\mathcal{C}$ *ts i vs es n*)

      obtain *vs′ es′′* **where** *es′ = [Local n i vs′ es′′]*
                            publics-agree vs vs′
                            exprs-public-agree es es′′
      using *5(4) exprs-public-agree-imp-publics-agree1-const0 expr-public-agree-local*
      by *blast*
    thus *?case*
      using *e-typing-s-typing.intros(5)[OF 5(2)] 5(3)*
      by *blast*
next
  case *(6 C tr S cl)*
  have *es′ = [Callcl cl]*
    using *6(3) exprs-public-agree-imp-publics-agree1-const0*
    by *(fastforce simp add: expr-public-agree.simps)*
  thus *?case*
    using *e-typing-s-typing.intros(6) 6(1,2)*
    by *blast*
next
  case *(7 S C e0s ts t2s es n)*
  obtain *les′ es′′* **where** *es′ = [Label n les′ es′′]*
                           exprs-public-agree e0s les′
                           exprs-public-agree es es′′
    using *7(6) exprs-public-agree-imp-publics-agree1-const0 expr-public-agree-label*
    by *blast*
  thus *?case*
    using *e-typing-s-typing.intros(7) 7(2,4,5)*
    by *blast*
next
  case *(8 i S tvs vs C rs es ts)*
  have *tvs = map typeof vs′*
    using *8(2,7) publics-agree-imp-typeof*
    by *blast*
  thus *?case*
    using *e-typing-s-typing.intros(8)[OF 8(1) - 8(3) 8(5)[OF 8(8)] 8(6)]*
    by *blast*
qed

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*
 using *assms(1) store-public-agree-imp-store-typing[OF - assms(2)] publics-agree-imp-typeof[OF assms(3)]*
       exprs-public-agree-imp-e-typing-s-typing(2)[OF - assms(3,4)]
  unfolding *config-typing.simps*
  by *fastforce*

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*
  using *exprs-public-agree-imp-config-typing*[*OF assms*(*1*)] *assms*(*2*)
  by *simp*

lemma *expr-public-agree-trans*:
  assumes *expr-public-agree a b*
        expr-public-agree b c
  shows *expr-public-agree a c*
  using *assms*
proof −
note *hyp-trans* =
  list-all2-trans[*of* (λ*x1 x2*. *expr-public-agree x1 x2* ∧ (∀ *x*. *expr-public-agree x2 x*
⟶ *expr-public-agree x1 x*)) *expr-public-agree expr-public-agree*]
show *?thesis*
  using *assms*
proof (*induction arbitrary*: *c rule*: *expr-public-agree.induct*)
  case (*1 e*)
  thus *?case*
    by *simp*
next
  case (*2 v v'*)
  {
    fix *e v v'*
    assume *local-assms*:*public-agree v v' expr-public-agree* ($*C v*) *e*
    then obtain *v''* **where** *e* = $*C v''*
                    public-agree v v''
      using *expr-public-agree-imp-public-agree*
      by *blast*
    hence *expr-public-agree* ($*C v'*) *e*
      using *equivp-public-agree expr-public-agree-imp-public-agree*
      by (*metis equivp-def local-assms*(*1*) *public-agree-imp-expr-public-agree*)
  }
  thus *?case*
    using *2 public-agree-symm*
    by *blast*
next
  case (*3 bes bes' tf*)
  obtain *bes''* **where** *c* = ($*Block tf bes''*)
                exprs-public-agree ($*bes'*) ($*bes''*)
    using *3*(*2*) *expr-public-agree-block*
    by *blast*
  thus *?case*

using *3(1) hyp-trans expr-public-agree.intros(3)*
  by *simp*
next
  case (*4 bes bes' tf*)
  obtain *bes''* **where** *c* = (*$Loop tf bes''*)
          exprs-public-agree (*$∗bes'*) (*$∗bes''*)
    using *4(2) expr-public-agree-loop*
    by *blast*
  thus *?case*
    using *expr-public-agree.intros(4) 4(1) hyp-trans*
    by *simp*
next
  case (*5 bes1 bes1' bes2 bes2' tf*)
  obtain *bes1'' bes2''* **where** *c* = (*$If tf bes1'' bes2''*)
            exprs-public-agree (*$∗bes1'*) (*$∗bes1''*)
            exprs-public-agree (*$∗bes2'*) (*$∗bes2''*)
    using *5(3) expr-public-agree-if*
    by *blast*
  thus *?case*
    using *expr-public-agree.intros(5) 5(1,2) hyp-trans*
    by *simp*
next
  case (*6 les les' es es' n*)
  obtain *les'' es''* **where** *c* = (*Label n les'' es''*)
         exprs-public-agree les' les''
         exprs-public-agree es' es''
    using *6(3) expr-public-agree-label*
    by *blast*
  thus *?case*
    using *expr-public-agree.intros(6) 6(1,2) hyp-trans*
    by *simp*
next
  case (*7 vs vs' es es' n i*)
  obtain *vs'' es''* **where** *c-def*:*c* = (*Local n i vs'' es''*)
          publics-agree vs' vs''
          exprs-public-agree es' es''
    using *7(3) expr-public-agree-local*
    by *blast*
  moreover
  hence *publics-agree vs vs''*
     using *7(1) equiv-transp*[*OF equivp-public-agree*] *list-all2-trans*[*OF - 7(1)*
c-def(2), of public-agree]
    by *fastforce*
  thus *?case*
    using *expr-public-agree.intros(7) 7(1,2) hyp-trans c-def*
    unfolding *transp-def*
    by *fastforce*
qed
qed

lemma *equivp-expr-public-agree*:*equivp expr-public-agree*
  using *expr-public-agree-trans expr-public-agree-refl expr-public-agree-symm equivpI*
  unfolding *reflp-def symp-def transp-def*
  by *blast*

lemma *equivp-exprs-public-agree*:*equivp exprs-public-agree*
  using *equivp-expr-public-agree list.rel-reflp list.rel-symp list.rel-transp*
  unfolding *equivp-reflp-symp-transp*
  by *blast*

lemma *exprs-public-agree-trans*:
  assumes *exprs-public-agree es es′*
          exprs-public-agree es′ es″
  shows *exprs-public-agree es es″*
  using *assms equivp-exprs-public-agree*
  unfolding *equivp-reflp-symp-transp transp-def*
  by *blast*

lemma *equivp-store-public-agree*:*equivp store-public-agree*
  using *store-public-agree-trans store-public-agree-refl store-public-agree-symm equivpI*
  unfolding *reflp-def symp-def transp-def*
  by *blast*

lemma *config-indistinguishable-refl*:*config-indistinguishable c c*
  using *store-public-agree-refl publics-agree-refl exprs-public-agree-refl*
  by (*cases c*) *simp*

lemma *config-indistinguishable-symm*:
  assumes *c ∼-c c′*
  shows *c′ ∼-c c*
  using *assms store-public-agree-symm publics-agree-symm exprs-public-agree-symm*
  by (*cases c*) (*cases c′; simp*)

lemma *config-indistinguishable-trans*:
  assumes *c ∼-c c′*
          c′ ∼-c c″
  shows *c ∼-c c″*
  using *assms store-public-agree-trans publics-agree-trans exprs-public-agree-trans*
  apply (*cases c*)
  apply (*cases c′*)
  apply (*cases c″*)
  apply *simp*
  apply *metis*
  done

lemma *equivp-config-indistinguishable*:*equivp config-indistinguishable*
  using *config-indistinguishable-trans config-indistinguishable-refl config-indistinguishable-symm equivpI*

unfolding *reflp-def symp-def transp-def*
by *blast*

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 $\equiv$
    $(\lambda((s,vs,es),i)\ ((s',vs',es'),i').\ ((s,vs,es) \sim$-$c\ (s',vs',es')) \wedge$
                        $(\exists\ ts.\ \vdash$-$i\ s;vs;es : (Untrusted,ts)) \wedge$
                        $i = i')$

lemma *ex-config-untrusted-equiv-refl*: $\exists\ s\ vs\ es\ i.\ (((s,vs,es),i) \sim$-$cp\ ((s,vs,es),i))$
proof $-$
  obtain $\mathcal{S}$ $\mathcal{C}s$ *e-s e-inst* **where** $\mathcal{S}$-*def*:
    $\mathcal{S} = (\!|s$-*inst* $= \mathcal{C}s,\ s$-*funcs* $= [],\ s$-*tab* $= [],\ s$-*mem* $= [],\ s$-*globs* $= []|\!)$
    $\mathcal{C}s = [(\!|trust$-$t = Untrusted,\ types$-$t = [],\ func$-$t = [],\ global = [],\ table = None,$
memory = None, local = [], label = [], return = None$|\!)]$
    e-s $= (\!|inst = e$-*inst*, *funcs* $= [],\ tab = [],\ mem = [],\ globs = []|\!)$
    e-inst $= [(\!|types = [],\ funcs = [],\ tab = None,\ mem = None,\ globs = []|\!)]$
    by *blast*
  hence *list-all2* (*inst-typing* $\mathcal{S}$) $[(\!|types = [],\ funcs = [],\ tab = None,\ mem =$
None, globs = []$|\!)]$ $\mathcal{C}s$
    unfolding *inst-typing.simps memi-agree-def*
    by *auto*
  hence $\vdash$-0 *e-s*;[];[] : (*Untrusted*,[])
    using *e-typing-s-typing.intros(1)*[*OF b-e-typing.intros(35)*] $\mathcal{S}$-*def*
    unfolding *config-typing.simps store-typing.simps s-typing.simps*
    by *auto*
  moreover
  have (*e-s*,[],[]) $\sim$-c (*e-s*,[],[])
    using $\mathcal{S}$-*def(3,4)* *store-public-agree-def*
    by *auto*
  ultimately
  show *?thesis*
    unfolding *config-untrusted-equiv-def*
    by *simp blast*
qed

lemma *config-untrusted-equiv-symm*:
  assumes $((s,vs,es),i) \sim$-$cp\ ((s',vs',es'),i')$
  shows $((s',vs',es'),i') \sim$-$cp\ ((s,vs,es),i)$
proof $-$
  have $i' = i$
    using *assms*
    unfolding *config-untrusted-equiv-def*
    by *auto*
  moreover
  have $(s',vs',es') \sim$-$c\ (s,vs,es)$
    using *assms config-indistinguishable-symm*
    unfolding *config-untrusted-equiv-def*

    by (*simp del*: *config-indistinguishable.simps*)
  moreover
  have ($\exists$ *ts*. $\vdash$-*i s';vs';es'* : (*Untrusted,ts*))
    using *assms config-indistinguishable-imp-config-typing*
    unfolding *config-untrusted-equiv-def*
    by *fastforce*
  ultimately
  show *?thesis*
    unfolding *config-untrusted-equiv-def*
    by *fastforce*
qed

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 −
  have $i = i'$
    using *assms*
    unfolding *config-untrusted-equiv-def*
    by *auto*
  moreover
  have $(s,vs,es) \sim\text{-}c (s',vs',es')$
    using *assms config-indistinguishable-trans*
    unfolding *config-untrusted-equiv-def*
    by (*simp del*: *config-indistinguishable.simps*) *blast*
  ultimately
  show *?thesis*
    using *assms*(*1*)
    unfolding *config-untrusted-equiv-def*
    by *fastforce*
qed

lemma *part-equivp-config-untrusted-equiv*:*part-equivp config-untrusted-equiv*
  using *part-equivpI*[*of config-untrusted-equiv*] *ex-config-untrusted-equiv-refl*
      config-untrusted-equiv-symm config-untrusted-equiv-trans
  unfolding *symp-def transp-def*
  by *fast*

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

quotient-type *config-untrusted-quot* = (($s \times v\ list \times e\ list$) $\times$ *nat*) / *partial*:*config-untrusted-equiv*
  by (*rule part-equivp-config-untrusted-equiv*)

lift-definition *config-untrusted-quot-inst-length* :: *config-untrusted-quot* $\Rightarrow$ *nat* **is**
($\lambda(c,i)$. *length* (*inst* (*fst c*)))
proof −
  fix *prod1*::(($s \times v\ list \times e\ list$) $\times$ *nat*) **and** *prod2*::(($s \times v\ list \times e\ list$) $\times$ *nat*)

```
      assume assms:config-untrusted-equiv prod1 prod2
      show (case prod1 of
            (c, i) ⇒ length (inst (fst c))) =
            (case prod2 of
            (c, i) ⇒ length (inst (fst c)))
      proof (cases prod1; cases prod2)
        fix a1 b1 a2 b2
        assume local-assms:prod1 = (a1,b1) prod2 = (a2,b2)
        thus ?thesis
          using assms
          unfolding config-untrusted-equiv-def
          apply (cases a1; cases a2)
          apply simp
          apply (metis config-typing.simps store-public-agree-imp-store-typing store-typing-imp-inst-length-eq)
          done
      qed
    qed
```

lift-definition *config-untrusted-quot-store-typing* :: *config-untrusted-quot* ⇒ *s-context*
⇒ *bool* **is** (λ(c,i) 𝒮. *store-typing* (*fst c*) 𝒮)
proof −
  fix *prod1*::((*s* × *v list* × *e list*) × *nat*) **and** *prod2*::((*s* × *v list* × *e list*) × *nat*)
  assume *assms*:*config-untrusted-equiv prod1 prod2*
  show (*case prod1 of*
      (*c*, *i*) ⇒ *store-typing* (*fst c*)) =
      (*case prod2 of*
      (*c*, *i*) ⇒ *store-typing* (*fst c*))
  proof (*cases prod1*; *cases prod2*)
    fix *a1 b1 a2 b2*
    assume *local-assms*:*prod1* = (*a1*,*b1*) *prod2* = (*a2*,*b2*)
    thus *?thesis*
      using *fun-eq-iff*[*symmetric, of store-typing* (*fst a1*) *store-typing* (*fst a2*)]
         assms
      unfolding *config-untrusted-equiv-def*
      apply (*cases a1*; *cases a2*)
      apply *simp*
      apply (*metis store-public-agree-imp-store-typing store-public-agree-symm*)
      done
  qed
qed

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 −
  fix 𝒮 𝒞 **and** *prod1*::((*s* × *v list* × *e list*) × *nat*) **and** *prod2*::((*s* × *v list* × *e list*)
× *nat*)
  assume *assms*:*config-untrusted-equiv prod1 prod2*
  show (*case prod1 of*
      (*c*, *i*) ⇒

```
          e-typing 𝒮
          𝒞 (snd (snd c))) =
      (case prod2 of
      (c, i) ⇒
        e-typing 𝒮
        𝒞 (snd (snd c)))
  proof (cases prod1; cases prod2)
    fix a1 b1 a2 b2
    assume local-assms:prod1 = (a1,b1) prod2 = (a2,b2)
    thus ?thesis
      using assms
          fun-eq-iff[symmetric, of e-typing 𝒮 𝒞 (snd (snd a1)) e-typing 𝒮 𝒞 (snd
(snd a2))]
      unfolding config-untrusted-equiv-def
      apply (cases a1; cases a2)
      apply simp
      apply (metis exprs-public-agree-imp-e-typing-s-typing(1) exprs-public-agree-symm
tf.exhaust)
      done
  qed
qed
```

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 −
  fix *𝒮 tr rs prod1 prod2*
  assume *assms:config-untrusted-equiv prod1 prod2*
  show *(case prod1 of*
```
      (c, i) ⇒
        s-typing 𝒮 tr
        rs i (fst (snd c))
        (snd (snd c))) =
      (case prod2 of
      (c, i) ⇒
        s-typing 𝒮 tr
        rs i (fst (snd c))
        (snd (snd c)))
```
  proof *(cases prod1; cases prod2)*
    fix *a1 b1 a2 b2*
    assume *local-assms:prod1 = (a1,b1) prod2 = (a2,b2)*
    thus *?thesis*
      using *assms*
          fun-eq-iff[symmetric, of s-typing 𝒮 tr rs b1 (fst (snd a1)) (snd (snd a1))
                                  s-typing 𝒮 tr rs b2 (fst (snd a2)) (snd (snd a2))]
      unfolding *config-untrusted-equiv-def*
      apply *(cases a1; cases a2)*
      apply *simp*
      apply *(meson exprs-public-agree-imp-e-typing-s-typing(2) exprs-public-agree-symm*

publics-agree-symm)
    done
  qed
qed

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 −
  fix *prod1 prod2*
  assume *assms*:*config-untrusted-equiv prod1 prod2*
  show (*case prod1 of*
      (*x, xa*) ⇒
       (*case x of*
       (*s, vs, es*) ⇒
        λ*i. config-typing i s vs es*)
       xa) =
     (*case prod2 of*
      (*x, xa*) ⇒
       (*case x of*
       (*s, vs, es*) ⇒
        λ*i. config-typing i s vs es*)
       xa)
  proof (*cases prod1*; *cases prod2*)
    fix *a1 b1 a2 b2*
    assume *local-assms*:*prod1* = (*a1,b1*) *prod2* = (*a2,b2*)
    thus *?thesis*
    using *assms config-indistinguishable-symm*
      fun-eq-iff [*symmetric, of* (*case a1 of* (*s, xa, xb*) ⇒ *config-typing b1 s xa xb*)
                           (*case a2 of* (*s, xa, xb*) ⇒ *config-typing b2 s xa xb*)]
    unfolding *config-untrusted-equiv-def*
    apply (*cases a1*; *cases a2*)
    apply *simp*
   apply (*meson config-indistinguishable.simps config-indistinguishable-imp-config-typing*)
    done
qed
qed

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*

228

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-Some-action t1 t2 v*) ∼-a (*Convert-Some-action t1 t2 v′*)
| *convert-None*:⟦*is-int-t t1*; *is-int-t t2*; *types-agree t1 v*; *public-agree v v′*⟧ ⟹ (*Convert-None-action t1 t2 v*) ∼-a (*Convert-None-action t1 t2 v′*)

lemma *action-indistinguishable-symm*:
　assumes *a* ∼-a *b*
　shows *b* ∼-a *a*
　using *assms*
proof (*induction rule*: *action-indistinguishable.induct*)
　case (*refl a*)
　thus *?case*
　　using *action-indistinguishable.refl*
　　by −
next
　case (*binop-32 iop c1 c2 c1′ c2′*)
　thus *?case*
　　using *action-indistinguishable.binop-32*
　　by *blast*
next
　case (*binop-64 iop c1 c2 c1′ c2′*)
　thus *?case*
　　using *action-indistinguishable.binop-64*
　　by *blast*
next
　case (*select c1 c2*)
　thus *?case*
　　using *action-indistinguishable.select*
　　by *blast*
next
　case (*host-Some s s′ vcs vcs′ tf f*)
　thus *?case*
　　using *action-indistinguishable.host-Some store-public-agree-symm publics-agree-symm*
　　by *metis*
next
　case (*host-None s s′ vcs vcs′ tf f*)
　thus *?case*
　　using *action-indistinguishable.host-None store-public-agree-symm publics-agree-symm*
　　by *metis*
next
　case (*convert-Some t1 t2 v v′*)
　thus *?case*

using *action-indistinguishable.convert-Some public-agree-imp-types-agree public-agree-symm*
  by *metis*
next
  case (*convert-None t1 t2 v v'*)
  thus *?case*
  using *action-indistinguishable.convert-None public-agree-imp-types-agree public-agree-symm*
    by *metis*
qed

lemma *action-indistinguishable-trans*:
  assumes $a \sim\text{-}a\ b$
        $b \sim\text{-}a\ c$
  shows $a \sim\text{-}a\ c$
  using *assms*
proof (*induction a b rule*: *action-indistinguishable.induct*)
  case (*refl a*)
  thus *?case*
    by −
next
  case (*binop-32 iop c1 c2 c1' c2'*)
  thus *?case*
    by (*fastforce simp add*: *action-indistinguishable.simps*)
next
  case (*binop-64 iop c1 c2 c1' c2'*)
  thus *?case*
    by (*fastforce simp add*: *action-indistinguishable.simps*)
next
  case (*select sec c*)
  thus *?case*
    by (*fastforce simp add*: *action-indistinguishable.simps*)
next
  case (*host-Some s s' vcs vcs' s-o s-o' vcs-o vcs-o' tf f hs hs'*)
  thus *?case*
  using *store-public-agree-trans*[*OF host-Some(1)*] *publics-agree-trans*[*OF host-Some(2)*]
    store-public-agree-trans[*OF host-Some(3)*] *publics-agree-trans*[*OF host-Some(4)*]
    by (*fastforce simp add*: *action-indistinguishable.simps*)
next
  case (*host-None s s' vcs vcs' tf f*)
  thus *?case*
  using *store-public-agree-trans*[*OF host-None(1)*] *publics-agree-trans*[*OF host-None(2)*]
    by (*fastforce simp add*: *action-indistinguishable.simps*)
next
  case (*convert-Some t1 t2 v v'*)
  thus *?case*
    using *public-agree-trans*[*OF convert-Some(4)*]
    by (*fastforce simp add*: *action-indistinguishable.simps*)
next
  case (*convert-None t1 t2 v v'*)
  thus *?case*

    using *public-agree-trans*[*OF convert-None(4)*]
    by (*fastforce simp add*: *action-indistinguishable.simps*)
qed

lemma *equivp-action-indistinguishable*: *equivp action-indistinguishable*
  using *action-indistinguishable.refl action-indistinguishable-symm action-indistinguishable-trans*
  unfolding *equivp-reflp-symp-transp reflp-def symp-def transp-def*
  by *blast*

lemma *equivp-obs*: *equivp* (*list-all2 action-indistinguishable*)
  using *equivp-action-indistinguishable list.rel-reflp list.rel-symp list.rel-transp*
  unfolding *equivp-reflp-symp-transp*
  by *blast*

quotient-type (**overloaded**) *observation = action list / list-all2 action-indistinguishable*
  using *equivp-obs*
  by *blast*

abbreviation *abs-obs* :: *action list $\Rightarrow$ observation* ($\mathcal{A}$ - *60*) **where**
  abs-obs a $\equiv$ abs-observation a

inductive *reduction-actions* :: [*s, v list, e list, nat, action list*] $\Rightarrow$ *bool* (*r′-actions* ⦇*-;-;-*⦈ - - *60*) **where**
  ⟦*const-list es $\vee$ es = [Trap]*⟧ $\Longrightarrow$ *r-actions* ⦇*s;vs;es*⦈ *i* []
| ⟦⦇*s;vs;es*⦈ *a⤳-i* ⦇*s′;vs′;es′*⦈; *r-actions* ⦇*s′;vs′;es′*⦈ *i as*⟧ $\Longrightarrow$ *r-actions* ⦇*s;vs;es*⦈ *i* (*a#as*)

inductive *reduce-weight* :: [*s, v list, e list, nat, nat, s, v list, e list*] $\Rightarrow$ *bool* ((⦇*-;-;-*⦈) |-|⤳′- - ⦇*-;-;-*⦈) *60*) **where**
  ⦇*s;vs;es*⦈ *a⤳-i* ⦇*s′;vs′;es′*⦈ $\Longrightarrow$ ⦇*s;vs;es*⦈ |(*weight a*)|⤳-i ⦇*s′;vs′;es′*⦈

inductive *reduction-weight* :: [*s, v list, e list, nat, nat*] $\Rightarrow$ *bool* (*r′-weight* ⦇*-;-;-*⦈ - - *60*) **where**
  ⟦*const-list es $\vee$ es = [Trap]*⟧ $\Longrightarrow$ *r-weight* ⦇*s;vs;es*⦈ *i 0*
| ⟦⦇*s;vs;es*⦈ |*w*|⤳-i ⦇*s′;vs′;es′*⦈; *r-weight* ⦇*s′;vs′;es′*⦈ *i w′*⟧ $\Longrightarrow$ *r-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*))
  using *assms*
proof (*induction rule*: *reduction-actions.induct*)
  case (*1 es s vs i*)
  thus *?case*
    using *reduction-weight.intros(1)*
    by *fastforce*
next
  case (*2 s vs es a i s′ vs′ es′ as*)

show *?case*
    using *reduce-weight.intros(1)[OF 2(1)] reduction-weight.intros(2)[OF - 2(3)]*
    by *fastforce*
qed

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 −
  show *smem-ind s′ i = Some j*
    using *store-public-agree-smem-ind-eq assms(1,2)*
    by *fastforce*
  moreover
  show *j < length (s.mem s′)*
    using *store-typing-imp-inst-length-eq[OF assms(3)] assms(1,4)*
       store-public-agree-imp-store-typing[OF assms(3,2)]
       store-typing-imp-mem-agree-Some(1)[OF assms(3)]
       store-typing-imp-mem-length-eq
    by *fastforce*
  thus *mem-agree:memories-public-agree (s.mem s) (s.mem s′) memory-public-agree*
((s.mem s)!j) ((s.mem s′)!j)
    using *assms(2)*
    by *(metis store-public-agree-def, metis list-all2-nthD2 store-public-agree-def)*
  thus *∃m′. s.mem s′ ! j = (m′, sec)*
    using *assms(5)*
    unfolding *memory-public-agree-def*
    by *(metis eq-snd-iff)*
qed

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−
  have *option-projr (memory (s-inst $\mathcal{S}$! i)) = Some sec*
    using *store-typing-imp-mem-agree-inst[OF assms(3)] assms(1,2,4)*

       store-typing-imp-inst-length-eq[*OF assms(3)*]
   by *fastforce*
  thus *t-sec-is*:*t-sec t = sec*
   using *b-e-type-load(1)*[*OF unlift-b-e*[*of S C* [*Load t tp a off*]]]
       e-type-comp-conc1[*of S C* [$C *ConstInt32 sec′ k*] [$*Load t tp a off*]]
       assms(5,6)
   unfolding *option-projr-def*
   by *fastforce*
qed

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 S
       i < length (inst s)
       $\mathcal{C} = (s\text{-}inst\ \mathcal{S}\ !\ i)$⦇*trust-t := tr, local := local (s-inst S ! i) @ tvs, label :=*
arb-labs, return := arb-return⦈
       $\mathcal{S} \cdot \mathcal{C} \vdash$ [$C *ConstInt32 sec′ k*, $C *v*, $*Store t tp a off*] : *(ts -> ts′)*
  shows *t-sec t = sec*
      sec′ = Public
      types-agree t v
      $\exists v′\ v′′.\ es′=$ [$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)
      $\exists m′.\ s.mem\ s′\ !\ j = (m′, sec)$
proof −
  have *option-projr (memory (s-inst S ! i)) = Some sec*
   using *store-typing-imp-mem-agree-inst*[*OF assms(5)*] *assms(1,2,6)*
      store-typing-imp-inst-length-eq[*OF assms(5)*]
   by *fastforce*
  thus *t-sec-is*:*t-sec t = sec*
   using *b-e-type-store(2)*[*OF unlift-b-e*[*of S C* [*Store t tp a off*]]]
      e-type-comp-conc2[*of S C* [$C *ConstInt32 sec′ k*] [$C *v*] [$*Store t tp a off*]]
      assms(7,8)
   unfolding *option-projr-def*
   by *fastforce*
  show *sec-def*:*sec′ = Public*
          types-agree t v
   using *types-preserved-store(2,3) assms*
   by *auto*
  thus *es′-def*:$\exists v′\ v′′.\ es′=$ [$C *v′*, $C *v′′*, $*Store t tp a off*] ∧
                 v′ = (ConstInt32 sec′ k) ∧
                 public-agree v v′′

using *exprs-public-agree-imp-publics-agree1-const2* [*OF assms(3)*]
  by (*fastforce simp add: expr-public-agree.simps public-agree-def typeof-def t-sec-def*)
  show *mem-agree: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)
           $\exists m'. s.mem \; s' \; ! \; j = (m', sec)$
    using *memories-public-agree-helper* [*OF assms(1,4,5,6,2)*]
    by *auto*
qed

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 $\exists bs'. \; load \; m' \; n \; off \; l = Some \; bs'$
  using *assms load-size*
  unfolding *memory-public-agree-def*
  by (*metis fst-conv option.exhaust*)

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 $\exists bs'. \; load\text{-}packed \; sx \; m' \; n \; off \; lp \; l = Some \; bs'$
  using *assms load-packed-size*
  unfolding *memory-public-agree-def*
  by (*metis fst-conv option.exhaust*)

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 $\exists mem''. \; store \; m' \; (nat\text{-}of\text{-}int \; k) \; off \; (bits \; v'') \; (t\text{-}length \; t) = Some \; mem''$
$\wedge$
             memory-public-agree (mem',sec) (mem'',sec)
proof $-$
  obtain *mem'' ***where*** store-def:store m' (nat-of-int k) off (bits v'') (t-length t)*
= Some mem''
    using *store-size1 assms(4,5,6,7)*
    unfolding *memory-public-agree-def*
    by (*metis option.exhaust prod.sel(1)*)
  moreover

have *mem-eq*:*mem-size mem′ = mem-size mem″*
  using *assms(4,5,6,7) store-size store-def*
  unfolding *memory-public-agree-def*
  by *simp metis*
have *memory-public-agree (mem′,sec) (mem″,sec)*
proof (*cases sec*)
  case *Secret*
  thus *?thesis*
    using *Secret mem-eq*
    unfolding *memory-public-agree-def*
    by *auto*
next
  case *Public*
  hence *v = v″*
    using *assms(1,2,3)*
    unfolding *types-agree-def public-agree-def*
    by *fastforce*
  thus *?thesis*
    using *assms(4,5,6,7) store-def mem-eq*
    unfolding *memory-public-agree-def*
    by *auto*
qed
ultimately
show *?thesis*
  by *blast*
qed

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 ∃ *mem″. store-packed m′ (nat-of-int k) off (bits v″) (tp-length tp) =*
Some mem″ ∧
            memory-public-agree (mem′,sec) (mem″,sec)
proof −
 obtain *mem″* **where** *store-def*:*store-packed m′ (nat-of-int k) off (bits v″) (tp-length*
tp) = Some mem″
    using *store-packed-size1 assms(4,5,6,7)*
    unfolding *memory-public-agree-def*
    by (*metis option.exhaust prod.sel(1)*)
  moreover
  have *mem-eq*:*mem-size mem′ = mem-size mem″*
    using *assms(4,5,6,7) store-packed-size store-def*
    unfolding *memory-public-agree-def*
    by *simp metis*

have *memory-public-agree* (*mem′,sec*) (*mem″,sec*)
proof (*cases sec*)
  case *Secret*
  thus *?thesis*
    using *Secret mem-eq*
    unfolding *memory-public-agree-def*
    by *auto*
next
  case *Public*
  hence $v = v''$
    using *assms(1,2,3)*
    unfolding *types-agree-def public-agree-def*
    by *fastforce*
  thus *?thesis*
    using *assms(4,5,6,7) store-def mem-eq*
    unfolding *memory-public-agree-def*
    by *auto*
qed
ultimately
show *?thesis*
  by *blast*
qed

lemma *binop-i-secret-imp-binop-i-some*:
  assumes *safe-binop-i iop*
  shows $\exists c.\ app\text{-}binop\text{-}i\ iop\ c1\ c2 = Some\ c$
  using *assms*
proof (*cases iop*)
  case (*Shr sx*)
  thus *?thesis*
  by (*cases sx*) (*auto simp add*: *app-binop-i-def*)
qed (*auto simp add*: *safe-binop-i-def app-binop-i-def*)

lemma *cvtop-secret-imp-cvt-some*:
  assumes $\mathcal{S}\cdot\mathcal{C} \vdash [\$C\ v,\ \$Cvtop\ t2\ Convert\ t1\ sx] : (ts\ \text{->}\ ts')$
      is-secret-t (*typeof v*)
  shows $\exists v'.\ cvt\ t2\ sx\ v = Some\ v'$
proof −
  have *sec-agree:typeof* $v = t1$
         $t2 \neq t1$
         t-sec $t2 = $ *t-sec* $t1$
         ($sx = None$) = (*is-float-t* $t2 \wedge$ *is-float-t* $t1 \vee$ *is-int-t* $t2 \wedge$ *is-int-t* $t1$
$\wedge$ *t-length* $t2 <$ *t-length* $t1$)
    using *typeof-cvtop*[*OF assms(1)*]
    unfolding *typeof-def*
    by *blast*+
  have ($t1 = (T\text{-}i32\ Secret) \wedge t2 = (T\text{-}i64\ Secret)) \vee (t1 = (T\text{-}i64\ Secret) \wedge t2$
$= (T\text{-}i32\ Secret))$
  proof −

236

have *ints*:*is-int-t t2 is-int-t t1*
    using *sec-agree(1,3) assms(2) is-secret-int-t*
    by *auto*
  show *?thesis*
    using *sec-agree(1,2,3) is-int-t-exists[OF ints(1)] is-int-t-exists[OF ints(2)] assms(2) t-sec-def*
    by *auto*
qed
then consider (*1*) *t1 = (T-i32 Secret) t2 = (T-i64 Secret) $\exists\,c.\ v = ConstInt32$
Secret c*
           | (*2*) *t1 = (T-i64 Secret) t2 = (T-i32 Secret) $\exists\,c.\ v = ConstInt64$
Secret c*
  using *typeof-i32 typeof-i64 sec-agree(1)*
  unfolding *public-agree-def*
  by *fastforce*
thus *?thesis*
proof *cases*
  case *1*
  then obtain *s* **where** *sx = Some s*
    using *sec-agree(4)*
    unfolding *is-int-t-def t-length-def is-float-t-def*
    by *auto*
  thus *?thesis*
    using *1*
    unfolding *cvt-def cvt-i64-def*
    by (*cases s*) *auto*
  next
    case *2*
    hence *sx = None*
      using *sec-agree(4)*
      unfolding *is-int-t-def t-length-def*
      by *auto*
    thus *?thesis*
      using *2*
      unfolding *cvt-def cvt-i32-def*
      by *auto*
  qed
qed

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* $\wedge\ (a \sim\text{-}a\ a′)$
  using *assms*
proof (*induction rule*: *reduce-simple.induct*)
  case (*unop-i32 sec′ c sec iop*)
  obtain *v* **where** *v-def*:*es′ =* [$\$C\ v$, $\$Unop\text{-}i\ (T\text{-}i32\ sec)\ iop$]

237

$$public\text{-}agree\ (ConstInt32\ sec'\ c)\ v$$
    using *exprs-public-agree-imp-publics-agree1-const1*[*OF unop-i32(1)*]
    by (*fastforce simp add: expr-public-agree.simps*)
  have *sec-agree:sec'* = *sec*
    using *typeof-unop-testop*[*OF unop-i32(2)*]
    unfolding *typeof-def*
    by *auto*
  show *?case*
  proof (*cases sec*)
    case *Secret*
    obtain *c'* **where** *v = ConstInt32 sec c'*
      using *sec-agree v-def(2) public-agree-public-i32*
      by *blast*
    moreover
    have *expr-public-agree* ($C ConstInt32 sec (app-unop-i iop c)) ($C ConstInt32 sec (app-unop-i iop c'))
      using *t-sec-def Secret expr-public-agree.intros(2)*[*of ConstInt32 sec (app-unop-i iop c) ConstInt32 sec (app-unop-i iop c')*]
      unfolding *public-agree-def typeof-def*
      by *simp*
    ultimately
    show *?thesis*
      using *v-def(1) reduce-simple.unop-i32 action-indistinguishable.intros(1)*
      by *fastforce*
  next
    case *Public*
    hence *v = ConstInt32 sec c*
      using *v-def(2) sec-agree*
      unfolding *public-agree-def typeof-def t-sec-def*
      by *auto*
    thus *?thesis*
        using *v-def(1) reduce-simple.unop-i32 exprs-public-agree-refl*[*of* [$C ConstInt32 sec (app-unop-i iop c)]]
           action-indistinguishable.intros(1)
      by *fastforce*
  qed
next
  case (*unop-i64 sec' c sec iop*)
  obtain *v* **where** *v-def:es'* = [$C v, $Unop-i (T-i64 sec) iop]
$$public\text{-}agree\ (ConstInt64\ sec'\ c)\ v$$
    using *exprs-public-agree-imp-publics-agree1-const1*[*OF unop-i64(1)*]
    by (*fastforce simp add: expr-public-agree.simps*)
  have *sec-agree:sec'* = *sec*
    using *typeof-unop-testop*[*OF unop-i64(2)*]
    unfolding *typeof-def*
    by *auto*
  show *?case*
  proof (*cases sec*)
    case *Secret*

238

obtain c' **where** $v = ConstInt64\ sec\ c'$
  using *sec-agree v-def(2) public-agree-public-i64*
  by *blast*
moreover
have *expr-public-agree* ($C\ ConstInt64\ sec\ (app-unop-i\ iop\ c)$) ($C\ ConstInt64$
sec ($app-unop-i\ iop\ c'$))
    using *Secret expr-public-agree.intros(2)[of ConstInt64 sec (app-unop-i iop c)*
ConstInt64 sec (app-unop-i iop c')]
  unfolding *public-agree-def typeof-def t-sec-def*
  by *simp*
ultimately
show *?thesis*
  using *v-def(1) reduce-simple.unop-i64 action-indistinguishable.intros(1)*
  by *fastforce*
next
  case *Public*
  hence $v = ConstInt64\ sec\ c$
    using *v-def(2) sec-agree*
    unfolding *public-agree-def typeof-def t-sec-def*
    by *auto*
  thus *?thesis*
      using *v-def(1) reduce-simple.unop-i64 exprs-public-agree-refl[of [$C Con-*
stInt64 sec (app-unop-i iop c)]]
        action-indistinguishable.intros(1)
    by *fastforce*
  qed
next
  case ($unop\text{-}f32\ c\ fop$)
  obtain v **where** $v\text{-}def\text{:}es' = [\$C\ v,\ \$Unop\text{-}f\ T\text{-}f32\ fop]$
                public-agree ($ConstFloat32\ c$) v
    using *exprs-public-agree-imp-publics-agree1-const1[OF unop-f32(1)]*
    by (*fastforce simp add: expr-public-agree.simps*)
  hence $v = ConstFloat32\ c$
    using *v-def(2)*
    unfolding *public-agree-def typeof-def t-sec-def*
    by *auto*
  thus *?case*
      using *v-def(1) reduce-simple.unop-f32 exprs-public-agree-refl[of [$C Const-*
Float32 (app-unop-f fop c)]]
        action-indistinguishable.intros(1)
    by *fastforce*
next
  case ($unop\text{-}f64\ c\ fop$)
  obtain v **where** $v\text{-}def\text{:}es' = [\$C\ v,\ \$Unop\text{-}f\ T\text{-}f64\ fop]$
                public-agree ($ConstFloat64\ c$) v
    using *exprs-public-agree-imp-publics-agree1-const1[OF unop-f64(1)]*
    by (*fastforce simp add: expr-public-agree.simps*)
  hence $v = ConstFloat64\ c$
    using *v-def(2)*

239

unfolding *public-agree-def typeof-def t-sec-def*
   by *auto*
  thus *?case*
    using *v-def*(1) *reduce-simple.unop-f64 exprs-public-agree-refl*[*of* [$C *Const-Float64* (*app-unop-f fop c*)]]
      action-indistinguishable.intros(1)
   by *fastforce*
next
  case (*binop-i32-Some iop c1 c2 c sec′ sec″ sec*)
  have *is-safe*:*sec′ = sec sec″ = sec is-secret-t* (*T-i32 sec*) $\Longrightarrow$ *safe-binop-i iop*
   using *typeof-binop-relop*[*OF binop-i32-Some*(3)]
   unfolding *typeof-def*
   by *fastforce+*
  then obtain *c1′ c2′* **where** *es′-def*:*es′* = [$C *ConstInt32 sec c1′*, $C *ConstInt32 sec c2′*, $*Binop-i* (*T-i32 sec*) *iop*]
                      public-agree (*ConstInt32 sec c1*) (*ConstInt32 sec c1′*)
                      public-agree (*ConstInt32 sec c2*) (*ConstInt32 sec c2′*)
   using *exprs-public-agree-imp-publics-agree1-const2*[*OF binop-i32-Some*(2)]
      expr-public-agree-refl[*of* $*Binop-i* (*T-i32 sec*) *iop*]
      public-agree-public-i32[*of sec c1*]
      public-agree-public-i32[*of sec c2*]
   by (*fastforce simp add*: *expr-public-agree.simps*)
  show *?case*
  proof (*cases sec*)
   case *Secret*
   then obtain *c′* **where** *c-def*:*app-binop-i iop c1′ c2′ = Some c′*
   using *binop-i-secret-imp-binop-i-some typeof-binop-relop*(3)[*OF binop-i32-Some*(3)]
    unfolding *typeof-def t-sec-def*
    by *fastforce*
  have *abs-agree*:(*Binop-i32-Some-action iop c1 c2*) $\sim$-a (*Binop-i32-Some-action iop c1′ c2′*)
    using *is-safe*(3) *Secret t-sec-def*
    by (*simp add*: *action-indistinguishable.intros*(2))
   thus *?thesis*
    using *reduce-simple.binop-i32-Some*[*OF c-def*] *es′-def*(1) *Secret*
      expr-public-agree.intros(2)[*of ConstInt32 sec c ConstInt32 sec c′*]
    unfolding *public-agree-def typeof-def t-sec-def*
    by *fastforce*
  next
   case *Public*
   hence *c1 = c1′ c2 = c2′*
    using *es′-def*(2,3)
    unfolding *public-agree-def typeof-def t-sec-def*
    by *auto*
   thus *?thesis*
    using *es′-def*(1) *reduce-simple.binop-i32-Some*[*OF binop-i32-Some*(1)]
      exprs-public-agree-refl[*of* [$C *ConstInt32 sec c*]] *action-indistinguishable.intros*(1)
    by *fastforce*
  qed

next

  case (*binop-i32-None iop c1 c2 sec′ sec″ sec*)

  have *sec′ = sec sec″ = sec is-secret-t* (*T-i32 sec*) $\implies$ *safe-binop-i iop*

    using *typeof-binop-relop*[*OF binop-i32-None*(*3*)]

    unfolding *typeof-def*

    by *fastforce+*

  then obtain *c1′ c2′* **where** *es′-def*:*es′* = [$*C ConstInt32 sec c1′*, $*C ConstInt32 sec c2′*, $*Binop-i* (*T-i32 sec*) *iop*]

                        public-agree (*ConstInt32 sec c1*) (*ConstInt32 sec c1′*)

                        public-agree (*ConstInt32 sec c2*) (*ConstInt32 sec c2′*)

    using *exprs-public-agree-imp-publics-agree1-const2*[*OF binop-i32-None*(*2*)]

       expr-public-agree-refl[*of* $*Binop-i* (*T-i32 sec*) *iop*]

       public-agree-public-i32[*of sec c1*]

       public-agree-public-i32[*of sec c2*]

    by (*fastforce simp add*: *expr-public-agree.simps*)

  show *?case*

  proof (*cases sec*)

    case *Secret*

    thus *?thesis*

      using *binop-i-secret-imp-binop-i-some*[*of iop c1 c2*] *typeof-binop-relop*(*3*)[*OF*

  binop-i32-None(*3*)]

          binop-i32-None(*1*)

      unfolding *typeof-def t-sec-def*

      by *fastforce*

    next

    case *Public*

    hence *c1* = *c1′ c2* = *c2′*

      using *es′-def*(*2,3*)

      unfolding *public-agree-def typeof-def t-sec-def*

      by *auto*

    thus *?thesis*

      using *es′-def*(*1*) *reduce-simple.binop-i32-None*[*OF binop-i32-None*(*1*)]

         exprs-public-agree-refl[*of* [*Trap*]] *action-indistinguishable.intros*(*1*)

      by *fastforce*

  qed

next

  case (*binop-i64-Some iop c1 c2 c sec′ sec″ sec*)

  have *is-safe*:*sec′* = *sec sec″* = *sec is-secret-t* (*T-i64 sec*) $\implies$ *safe-binop-i iop*

    using *typeof-binop-relop*[*OF binop-i64-Some*(*3*)]

    unfolding *typeof-def*

    by *fastforce+*

  then obtain *c1′ c2′* **where** *es′-def*:*es′* = [$*C ConstInt64 sec c1′*, $*C ConstInt64 sec c2′*, $*Binop-i* (*T-i64 sec*) *iop*]

                        public-agree (*ConstInt64 sec c1*) (*ConstInt64 sec c1′*)

                        public-agree (*ConstInt64 sec c2*) (*ConstInt64 sec c2′*)

    using *exprs-public-agree-imp-publics-agree1-const2*[*OF binop-i64-Some*(*2*)]

       expr-public-agree-refl[*of* $*Binop-i* (*T-i64 sec*) *iop*]

       public-agree-public-i64[*of sec c1*]

       public-agree-public-i64[*of sec c2*]

    by (*fastforce simp add*: *expr-public-agree.simps*)

  show *?case*

  proof (*cases sec*)

    case *Secret*

    then obtain c' **where** *c-def*:*app-binop-i iop c1′ c2′ = Some c′*

    using *binop-i-secret-imp-binop-i-some typeof-binop-relop(3)[OF binop-i64-Some(3)]*

      unfolding *typeof-def t-sec-def*

      by *fastforce*

    have *abs-agree*:(*Binop-i64-Some-action iop c1 c2*) $\sim$-a (*Binop-i64-Some-action iop c1′ c2′*)

      using *is-safe(3) Secret t-sec-def*

      by (*simp add*: *action-indistinguishable.intros(3)*)

    thus *?thesis*

      using *reduce-simple.binop-i64-Some[OF c-def] es′-def(1) Secret*

          expr-public-agree.intros(2)[of ConstInt64 sec c ConstInt64 sec c′]

      unfolding *public-agree-def typeof-def t-sec-def*

      by *fastforce*

  next

    case *Public*

    hence *c1 = c1′ c2 = c2′*

      using *es′-def(2,3)*

      unfolding *public-agree-def typeof-def t-sec-def*

      by *auto*

    thus *?thesis*

      using *es′-def(1) reduce-simple.binop-i64-Some[OF binop-i64-Some(1)]*

        exprs-public-agree-refl[of [$C ConstInt64 sec c]] action-indistinguishable.intros(1)

      by *fastforce*

  qed

next

  case (*binop-i64-None iop c1 c2 sec′ sec″ sec*)

  have *sec′ = sec sec″ = sec is-secret-t* (*T-i64 sec*) $\implies$ *safe-binop-i iop*

    using *typeof-binop-relop[OF binop-i64-None(3)]*

    unfolding *typeof-def*

    by *fastforce+*

  then obtain *c1′ c2′* **where** *es′-def*:*es′ = [$C ConstInt64 sec c1′, $C ConstInt64 sec c2′, $Binop-i (T-i64 sec) iop]*

                            public-agree (*ConstInt64 sec c1*) (*ConstInt64 sec c1′*)

                            public-agree (*ConstInt64 sec c2*) (*ConstInt64 sec c2′*)

    using *exprs-public-agree-imp-publics-agree1-const2[OF binop-i64-None(2)]*

        expr-public-agree-refl[of $Binop-i (T-i64 sec) iop]

        public-agree-public-i64[of sec c1]

        public-agree-public-i64[of sec c2]

    by (*fastforce simp add*: *expr-public-agree.simps*)

  show *?case*

  proof (*cases sec*)

    case *Secret*

    thus *?thesis*

      using *binop-i-secret-imp-binop-i-some[of iop c1 c2] typeof-binop-relop(3)[OF binop-i64-None(3)]*

242

       binop-i64-None(*1*)
    unfolding *typeof-def t-sec-def*
    by *fastforce*
  next
    case *Public*
    hence $c1 = c1'$ $c2 = c2'$
      using *es'-def*(*2,3*)
      unfolding *public-agree-def typeof-def t-sec-def*
      by *auto*
    thus *?thesis*
      using *es'-def*(*1*) *reduce-simple.binop-i64-None*[*OF binop-i64-None*(*1*)]
         exprs-public-agree-refl[*of* [*Trap*]] *action-indistinguishable.intros*(*1*)
      by *fastforce*
  qed
next
  case (*binop-f32-Some fop c1 c2 c*)
  obtain *c1'* *c2'* **where** *es'-def:es'* = [*\$C ConstFloat32 c1'*, *\$C ConstFloat32 c2'*,
\$Binop-f T-f32 fop]
                     public-agree (*ConstFloat32 c1*) (*ConstFloat32 c1'*)
                     public-agree (*ConstFloat32 c2*) (*ConstFloat32 c2'*)
    using *exprs-public-agree-imp-publics-agree1-const2*[*OF binop-f32-Some*(*2*)]
        expr-public-agree-refl[*of \$Binop-f T-f32 fop*]
        public-agree-public-f32[*of c1*]
        public-agree-public-f32[*of c2*]
    unfolding *typeof-def*
    by (*fastforce simp add*: *expr-public-agree.simps*)
  hence $c1 = c1'$ $c2 = c2'$
    using *es'-def*(*2,3*)
    unfolding *public-agree-def typeof-def t-sec-def*
    by *auto*
  thus *?case*
    using *es'-def*(*1*) *reduce-simple.binop-f32-Some*[*OF binop-f32-Some*(*1*)]
      exprs-public-agree-refl[*of* [*\$C ConstFloat32 c*]] *action-indistinguishable.intros*(*1*)
    by *fastforce*
next
  case (*binop-f32-None fop c1 c2*)
  obtain *c1'* *c2'* **where** *es'-def:es'* = [*\$C ConstFloat32 c1'*, *\$C ConstFloat32 c2'*,
\$Binop-f T-f32 fop]
                     public-agree (*ConstFloat32 c1*) (*ConstFloat32 c1'*)
                     public-agree (*ConstFloat32 c2*) (*ConstFloat32 c2'*)
    using *exprs-public-agree-imp-publics-agree1-const2*[*OF binop-f32-None*(*2*)]
        expr-public-agree-refl[*of \$Binop-f T-f32 fop*]
        public-agree-public-f32[*of c1*]
        public-agree-public-f32[*of c2*]
    unfolding *typeof-def*
    by (*fastforce simp add*: *expr-public-agree.simps*)
  hence $c1 = c1'$ $c2 = c2'$
    using *es'-def*(*2,3*)
    unfolding *public-agree-def typeof-def t-sec-def*

      by *auto*
    thus *?case*
      using *es'-def*(*1*) *reduce-simple.binop-f32-None*[*OF binop-f32-None*(*1*)]
          exprs-public-agree-refl[*of* [*Trap*]] *action-indistinguishable.intros*(*1*)
      by *fastforce*
next
  case (*binop-f64-Some fop c1 c2 c*)
  then obtain *c1' c2'* **where** *es'-def*:*es'* = [*\$C ConstFloat64 c1'*, *\$C ConstFloat64*
c2', *\$Binop-f T-f64 fop*]
                        public-agree (*ConstFloat64 c1*) (*ConstFloat64 c1'*)
                        public-agree (*ConstFloat64 c2*) (*ConstFloat64 c2'*)
    using *exprs-public-agree-imp-publics-agree1-const2*[*OF binop-f64-Some*(*2*)]
      expr-public-agree-refl[*of \$Binop-f T-f64 fop*]
      public-agree-public-f64[*of c1*]
      public-agree-public-f64[*of c2*]
    unfolding *typeof-def*
    by (*fastforce simp add*: *expr-public-agree.simps*)
  hence *c1* = *c1' c2* = *c2'*
    using *es'-def*(*2,3*)
    unfolding *public-agree-def typeof-def t-sec-def*
    by *auto*
  thus *?case*
    using *es'-def*(*1*) *reduce-simple.binop-f64-Some*[*OF binop-f64-Some*(*1*)]
      exprs-public-agree-refl[*of* [*\$C ConstFloat64 c*]] *action-indistinguishable.intros*(*1*)
    by *fastforce*
next
  case (*binop-f64-None fop c1 c2*)
  then obtain *c1' c2'* **where** *es'-def*:*es'* = [*\$C ConstFloat64 c1'*, *\$C ConstFloat64*
c2', *\$Binop-f T-f64 fop*]
                        public-agree (*ConstFloat64 c1*) (*ConstFloat64 c1'*)
                        public-agree (*ConstFloat64 c2*) (*ConstFloat64 c2'*)
    using *exprs-public-agree-imp-publics-agree1-const2*[*OF binop-f64-None*(*2*)]
      expr-public-agree-refl[*of \$Binop-f T-f64 fop*]
      public-agree-public-f64[*of c1*]
      public-agree-public-f64[*of c2*]
    unfolding *typeof-def*
    by (*fastforce simp add*: *expr-public-agree.simps*)
  hence *c1* = *c1' c2* = *c2'*
    using *es'-def*(*2,3*)
    unfolding *public-agree-def typeof-def t-sec-def*
    by *auto*
  thus *?case*
    using *es'-def*(*1*) *reduce-simple.binop-f64-None*[*OF binop-f64-None*(*1*)]
      exprs-public-agree-refl[*of* [*Trap*]] *action-indistinguishable.intros*(*1*)
    by *fastforce*
next
  case (*testop-i32 sec' c sec testop*)
  obtain *v* **where** *v-def*:*es'* = [*\$C v*, *\$Testop* (*T-i32 sec*) *testop*]
                public-agree (*ConstInt32 sec' c*) *v*

    using *exprs-public-agree-imp-publics-agree1-const1*[*OF testop-i32*(*1*)]
    by (*fastforce simp add*: *expr-public-agree.simps*)
  have *sec-agree*:*sec′* = *sec*
    using *typeof-unop-testop*[*OF testop-i32*(*2*)]
    unfolding *typeof-def*
    by *auto*
  show *?case*
  proof (*cases sec*)
    case *Secret*
    obtain *c′* **where** *v* = *ConstInt32 sec c′*
      using *sec-agree v-def*(*2*) *public-agree-public-i32*
      by *blast*
    moreover
    have *expr-public-agree* ($*C ConstInt32 sec* (*wasm-bool* (*app-testop-i testop c*)))
($*C ConstInt32 sec* (*wasm-bool* (*app-testop-i testop c′*)))
      using *Secret expr-public-agree.intros*(*2*)
      unfolding *public-agree-def typeof-def t-sec-def*
      by *simp*
    ultimately
    show *?thesis*
      using *v-def*(*1*) *reduce-simple.testop-i32 action-indistinguishable.intros*(*1*)
      by *fastforce*
  next
    case *Public*
    hence *v* = *ConstInt32 sec c*
      using *v-def*(*2*) *sec-agree*
      unfolding *public-agree-def typeof-def t-sec-def*
      by *auto*
    thus *?thesis*
      using *v-def*(*1*) *reduce-simple.testop-i32 action-indistinguishable.intros*(*1*)
           exprs-public-agree-refl[*of* [$*C ConstInt32 sec* (*wasm-bool* (*app-testop-i*
testop c))]]
      by *fastforce*
  qed
next
  case (*testop-i64 sec′ c sec testop*)
  obtain *v* **where** *v-def*:*es′* = [$*C v*, $*Testop* (*T-i64 sec*) *testop*]
                 public-agree (*ConstInt64 sec′ c*) *v*
    using *exprs-public-agree-imp-publics-agree1-const1*[*OF testop-i64*(*1*)]
    by (*fastforce simp add*: *expr-public-agree.simps*)
  have *sec-agree*:*sec′* = *sec*
    using *typeof-unop-testop*[*OF testop-i64*(*2*)]
    unfolding *typeof-def*
    by *auto*
  show *?case*
  proof (*cases sec*)
    case *Secret*
    obtain *c′* **where** *v* = *ConstInt64 sec c′*
      using *sec-agree v-def*(*2*) *public-agree-public-i64*

245

      by *blast*
    moreover
    have *expr-public-agree* ($C ConstInt32 sec (wasm-bool (app-testop-i testop c)))
($C ConstInt32 sec (wasm-bool (app-testop-i testop c′)))
      using *Secret expr-public-agree.intros*(*2*)
      unfolding *public-agree-def typeof-def t-sec-def*
      by *simp*
    ultimately
    show *?thesis*
      using *v-def*(*1*) *reduce-simple.testop-i64 action-indistinguishable.intros*(*1*)
      by *fastforce*
  next
    case *Public*
    hence *v = ConstInt64 sec c*
      using *v-def*(*2*) *sec-agree*
      unfolding *public-agree-def typeof-def t-sec-def*
      by *auto*
    thus *?thesis*
      using *v-def*(*1*) *reduce-simple.testop-i64 action-indistinguishable.intros*(*1*)
          exprs-public-agree-refl[*of* [$C ConstInt32 sec (wasm-bool (app-testop-i
testop c))]]
      by *fastforce*
  qed
next
  case (*relop-i32 sec′ c1 sec″ c2 sec iop*)
  have *sec′ = sec sec″ = sec*
    using *typeof-binop-relop*[*OF relop-i32*(*2*)]
    unfolding *typeof-def*
    by *fastforce+*
  then obtain *c1′ c2′* **where** *es′-def*:*es′* = [$C ConstInt32 sec c1′, $C ConstInt32
sec c2′, $Relop-i (T-i32 sec) iop]
                    public-agree (*ConstInt32 sec c1*) (*ConstInt32 sec c1′*)
                    public-agree (*ConstInt32 sec c2*) (*ConstInt32 sec c2′*)
    using *exprs-public-agree-imp-publics-agree1-const2*[*OF relop-i32*(*1*)]
      expr-public-agree-refl[*of* $Relop-i (T-i32 sec) iop]
      public-agree-public-i32[*of sec c1*]
      public-agree-public-i32[*of sec c2*]
    by (*fastforce simp add*: *expr-public-agree.simps*)
  show *?case*
  proof (*cases sec*)
    case *Secret*
    hence *public-agree* (*ConstInt32 sec (wasm-bool (app-relop-i iop c1 c2))*) (*ConstInt32
sec (wasm-bool (app-relop-i iop c1′ c2′))*)
      unfolding *public-agree-def typeof-def t-sec-def*
      by *fastforce*
    thus *?thesis*
      using *reduce-simple.relop-i32 es′-def*(*1*) *Secret expr-public-agree.intros*(*2*)
          action-indistinguishable.intros(*1*)
      by *fastforce*

246

next
  case *Public*
  hence *c1 = c1′ c2 = c2′*
    using *es′-def(2,3)*
    unfolding *public-agree-def typeof-def t-sec-def*
    by *auto*
  thus *?thesis*
    using *es′-def(1) reduce-simple.relop-i32 action-indistinguishable.intros(1)*
        exprs-public-agree-refl[of [$C ConstInt32 sec (wasm-bool (app-relop-i iop
c1 c2))]]
    by *fastforce*
  qed
next
  case (*relop-i64 sec′ c1 sec″ c2 sec iop*)
  have *sec′ = sec sec″ = sec*
    using *typeof-binop-relop[OF relop-i64(2)]*
    unfolding *typeof-def*
    by *fastforce+*
  then obtain *c1′ c2′* **where** *es′-def:es′ = [$C ConstInt64 sec c1′, $C ConstInt64*
sec c2′, $Relop-i (T-i64 sec) iop]
                        public-agree (ConstInt64 sec c1) (ConstInt64 sec c1′)
                        public-agree (ConstInt64 sec c2) (ConstInt64 sec c2′)
    using *exprs-public-agree-imp-publics-agree1-const2[OF relop-i64(1)]*
        expr-public-agree-refl[of $Relop-i (T-i64 sec) iop]
        public-agree-public-i64[of sec c1]
        public-agree-public-i64[of sec c2]
    by (*fastforce simp add: expr-public-agree.simps*)
  show *?case*
  proof (*cases sec*)
    case *Secret*
   hence *public-agree (ConstInt32 sec (wasm-bool (app-relop-i iop c1 c2))) (ConstInt32*
sec (wasm-bool (app-relop-i iop c1′ c2′)))
      unfolding *public-agree-def typeof-def t-sec-def*
      by *fastforce*
    thus *?thesis*
    using *reduce-simple.relop-i64 es′-def(1) Secret expr-public-agree.intros(2)*
        action-indistinguishable.intros(1)
    by *fastforce*
  next
    case *Public*
    hence *c1 = c1′ c2 = c2′*
      using *es′-def(2,3)*
      unfolding *public-agree-def typeof-def t-sec-def*
      by *auto*
    thus *?thesis*
    using *es′-def(1) reduce-simple.relop-i64 action-indistinguishable.intros(1)*
        exprs-public-agree-refl[of [$C ConstInt32 sec (wasm-bool (app-relop-i iop
c1 c2))]]
    by *fastforce*

qed
next
  case (*relop-f32 c1 c2 fop*)
  then obtain *c1′ c2′* **where** *es′-def*:*es′* = [$*C ConstFloat32 c1′*, $*C ConstFloat32 c2′*, $*Relop-f T-f32 fop*]
$$public\text{-}agree\ (ConstFloat32\ c1)\ (ConstFloat32\ c1′)$$
$$public\text{-}agree\ (ConstFloat32\ c2)\ (ConstFloat32\ c2′)$$
    using *exprs-public-agree-imp-publics-agree1-const2*[*OF relop-f32*(*1*)]
        expr-public-agree-refl[*of* $*Relop-f T-f32 fop*]
        public-agree-public-f32[*of c1*]
        public-agree-public-f32[*of c2*]
    by (*fastforce simp add*: *expr-public-agree.simps*)
  hence *c1 = c1′ c2 = c2′*
    using *es′-def*(*2,3*)
    unfolding *public-agree-def typeof-def t-sec-def*
    by *auto*
  thus *?case*
    using *es′-def*(*1*) *reduce-simple.relop-f32 action-indistinguishable.intros*(*1*)
        exprs-public-agree-refl[*of* [$*C ConstInt32 Public* (*wasm-bool* (*app-relop-f fop c1 c2*))]]
    by *fastforce*
next
  case (*relop-f64 c1 c2 fop*)
  then obtain *c1′ c2′* **where** *es′-def*:*es′* = [$*C ConstFloat64 c1′*, $*C ConstFloat64 c2′*, $*Relop-f T-f64 fop*]
$$public\text{-}agree\ (ConstFloat64\ c1)\ (ConstFloat64\ c1′)$$
$$public\text{-}agree\ (ConstFloat64\ c2)\ (ConstFloat64\ c2′)$$
    using *exprs-public-agree-imp-publics-agree1-const2*[*OF relop-f64*(*1*)]
        expr-public-agree-refl[*of* $*Relop-f T-f64 fop*]
        public-agree-public-f64[*of c1*]
        public-agree-public-f64[*of c2*]
    by (*fastforce simp add*: *expr-public-agree.simps*)
  hence *c1 = c1′ c2 = c2′*
    using *es′-def*(*2,3*)
    unfolding *public-agree-def typeof-def t-sec-def*
    by *auto*
  thus *?case*
    using *es′-def*(*1*) *reduce-simple.relop-f64 action-indistinguishable.intros*(*1*)
        exprs-public-agree-refl[*of* [$*C ConstInt32 Public* (*wasm-bool* (*app-relop-f fop c1 c2*))]]
    by *fastforce*
next
  case (*convert-Some t1 v t2 sx v′*)
  obtain *v″* **where** *v-def*:*es′* = [$*C v″*, $*Cvtop t2 Convert t1 sx*]
$$public\text{-}agree\ v\ v″$$
    using *exprs-public-agree-imp-publics-agree1-const1*[*OF convert-Some*(*3*)]
    by (*fastforce simp add*: *expr-public-agree.simps*)
  have *sec-agree*:*typeof v = t1*
$$t2 \neq t1$$

248

$t\text{-}sec\ t2 = t\text{-}sec\ t1$

$(sx = None) = (is\text{-}float\text{-}t\ t2 \land is\text{-}float\text{-}t\ t1 \lor is\text{-}int\text{-}t\ t2 \land is\text{-}int\text{-}t\ t1$
$\land\ t\text{-}length\ t2 < t\text{-}length\ t1)$

  using *typeof-cvtop*[*OF convert-Some*(*4*)]

  unfolding *typeof-def*

  by *blast+*

 show *?case*

 proof (*cases t-sec* (*typeof v*))

  case *Secret*

  have $\mathcal{S}\cdot\mathcal{C} \vdash [\$C\ v'',\ \$Cvtop\ t2\ Convert\ t1\ sx] : (ts \text{ -> } ts')$

   using *e-typing-s-typing.intros*(*1*)

      exprs-public-agree-imp-b-e-typing[*OF unlift-b-e*[*of* - - [*C v*, *Cvtop t2*

Convert t1 sx]]]

     v-def(*1*) *convert-Some*(*3,4*)

   by (*metis to-e-list-2*)

  hence $\exists v'''.\ cvt\ t2\ sx\ v'' = Some\ v'''$

   using *cvtop-secret-imp-cvt-some Secret public-agree-imp-typeof*[*OF v-def*(*2*)]

   by *fastforce*

  then obtain v''' **where** $v'''\text{-}def$:$cvt\ t2\ sx\ v'' = Some\ v'''$

   by *blast*

  hence *exprs-public-agree* $[(\$C\ v')]\ [(\$C\ v''')]$

   using *t-cvt convert-Some*(*2*) *sec-agree*(*1,3*) *Secret expr-public-agree.intros*(*2*)

   unfolding *public-agree-def*

   by *simp*

  thus *?thesis*

  using *reduce-simple.convert-Some*[*OF* - *v'''-def*] *v-def*(*1*) *action-indistinguishable.intros*(*7*)

    public-agree-imp-types-agree-insecure[*OF convert-Some*(*1*)]

  by (*metis Secret is-secret-int-t sec-agree*(*1,3*) *types-agree-def v-def*(*2*))

 next

  case *Public*

  hence $v'' = v$

   using *v-def*(*2*)

   unfolding *public-agree-def*

   by *fastforce*

  thus *?thesis*

  using *convert-Some*(*1,2*) *v-def*(*1*) *exprs-public-agree-refl reduce-simple.convert-Some*

    action-indistinguishable.intros(*1*)

   by *blast*

 qed

next

 case (*convert-None t1 v t2 sx*)

 obtain v'' **where** $v\text{-}def$:$es' = [\$C\ v'',\ \$Cvtop\ t2\ Convert\ t1\ sx]$

          public-agree v v''

  using *exprs-public-agree-imp-publics-agree1-const1*[*OF convert-None*(*3*)]

  by (*fastforce simp add*: *expr-public-agree.simps*)

 have *sec-agree*:*typeof v = t1*

       $t2 \neq t1$

       $t\text{-}sec\ t2 = t\text{-}sec\ t1$

       $(sx = None) = (is\text{-}float\text{-}t\ t2 \land is\text{-}float\text{-}t\ t1 \lor is\text{-}int\text{-}t\ t2 \land is\text{-}int\text{-}t\ t1$

$\land$ *t-length t2* $<$ *t-length t1*)
    using *typeof-cvtop*[*OF convert-None*(*4*)]
    unfolding *typeof-def*
    by *blast+*
  show *?case*
  proof (*cases t-sec* (*typeof v*))
    case *Secret*
    thus *?thesis*
     using *cvtop-secret-imp-cvt-some*[*OF convert-None*(*4*)] *Secret convert-None*(*2*)
     by *fastforce*
  next
    case *Public*
    hence $v'' = v$
     using *v-def*(*2*)
     unfolding *public-agree-def*
     by *fastforce*
    thus *?thesis*
     using *convert-None*(*1*,*2*) *v-def*(*1*) *exprs-public-agree-refl reduce-simple.convert-None*
       action-indistinguishable.intros(*1*)
     by *blast*
  qed
next
  case (*reinterpret t1 v t2*)
  obtain v'' **where** *v-def*:*es'* = [$\$C$ v'', $\$Cvtop$ *t2 Reinterpret t1 None*]
              public-agree v v''
    using *exprs-public-agree-imp-publics-agree1-const1*[*OF reinterpret*(*2*)]
    by (*fastforce simp add*: *expr-public-agree.simps*)
  have *typeof-v*:*typeof v* = *t1 t2* $\neq$ *t1 t-sec t2* = *t-sec t1 t-length t2* = *t-length t1*
    using *typeof-cvtop*(*1*,*3*)[*OF reinterpret*(*3*)]
    by *blast+*
  thus *?case*
  proof (*cases t-sec* (*typeof* v''))
    case *Secret*
    hence *exprs-public-agree* [$\$C$ *wasm-deserialise* (*bits v*) *t2*] [$\$C$ *wasm-deserialise*
(*bits* v'') *t2*]
      using *wasm-deserialise-type*[*of* - *t2*] *expr-public-agree.intros*(*2*) *typeof-v*(*1*,*3*)
        public-agree-imp-typeof[*OF v-def*(*2*)]
      unfolding *public-agree-def*
      by *auto*
    thus *?thesis*
     using *v-def reduce-simple.reinterpret reinterpret*(*1*) *public-agree-imp-types-agree-insecure*
       action-indistinguishable.intros(*1*)
     by *blast*
  next
    case *Public*
    hence $v = v''$
     using *v-def*(*2*)
     unfolding *public-agree-def*
     by *auto*

thus *?thesis*
  using *v-def*(*1*) *reduce-simple.reinterpret reinterpret*(*1*) *action-indistinguishable.intros*(*1*)
       exprs-public-agree-refl[*of* [$*C wasm-deserialise* (*bits v*) *t2*]]
  by *blast*
qed
next
 case (*classify t1 v t2*)
 obtain v'' **where** *v-def*:*es'* = [$*C* v'', $*Cvtop t2 Classify t1 None*]
        public-agree v v''
  using *exprs-public-agree-imp-publics-agree1-const1*[*OF classify*(*2*)]
  by (*fastforce simp add*: *expr-public-agree.simps*)
 have *typeof-v*:*typeof v* = *t1 is-int-t t1 is-public-t t1 classify-t t1* = *t2*
  using *typeof-cvtop*(*1*,*4*)[*OF classify*(*3*)]
  by *blast+*
 thus *?case*
 proof (*cases t-sec* (*typeof* v''))
  case *Secret*
  thus *?thesis*
   using *typeof-v*(*1*,*3*) *v-def*(*2*) *public-agree-imp-typeof*
   by *auto*
 next
  case *Public*
  hence $v = v''$
   using *v-def*(*2*)
   unfolding *public-agree-def*
   by *auto*
  thus *?thesis*
   using *v-def*(*1*) *reduce-simple.classify classify*(*1*) *action-indistinguishable.intros*(*1*)
       exprs-public-agree-refl[*of* [$*C classify v*]]
   by *blast*
 qed
next
 case (*declassify t1 v t2*)
 show *?case*
  using *typeof-cvtop*(*5*)[*OF declassify*(*3*)] *declassify*(*4*)
  by *fastforce*
next
 case *unreachable*
 thus *?case*
  using *reduce-simple.unreachable exprs-public-agree-imp-publics-agree1-const0*[*OF*
unreachable(*1*)]
      action-indistinguishable.intros(*1*)
  by (*fastforce simp add*: *expr-public-agree.simps*)
next
 case *nop*
 thus *?case*
  using *reduce-simple.nop exprs-public-agree-imp-publics-agree1-const0*[*OF nop*(*1*)]
      action-indistinguishable.intros(*1*)
  by (*fastforce simp add*: *expr-public-agree.simps*)

next

  case (*drop v*)

  obtain v'' **where** *v-def*:*es′* = [$C v''$, $Drop$]

                    public-agree v v''

    using *exprs-public-agree-imp-publics-agree1-const1*[*OF drop*(*1*)]

    by (*fastforce simp add*: *expr-public-agree.simps*)

  thus *?case*

    using *reduce-simple.drop action-indistinguishable.intros*(*1*)

    by *blast*

next

  case (*select-false n v1 v2 sec sec′*)

  then obtain *v1′ v2′ n′* **where** *es′-def*:*es′* = [$C v1′$, $C v2′$, $C ConstInt32 sec n′$, $Select sec$]

                          public-agree v1 v1′

                          public-agree v2 v2′

                        public-agree (*ConstInt32 sec n*) (*ConstInt32 sec n′*)

                          sec = *sec′*

                          typeof v1 = *typeof v2*

                          sec = *Secret* ⟶ *is-secret-t* (*typeof v1*)

                          sec = *Secret* ⟶ *is-secret-t* (*typeof v2*)

    using *exprs-public-agree-imp-publics-agree1-const3*[*OF select-false*(*2*)]

        typeof-select[*OF select-false*(*3*)] *public-agree-public-i32*

    unfolding *typeof-def t-sec-def*

    by (*fastforce simp add*: *expr-public-agree.simps*)

  show *?case*

  proof (*cases sec*)

    case *Secret*

    hence *v2-v1′*:*public-agree v2 v1′*

      using *es′-def*(*2,6,7,8*) *public-agree-def*

      by *auto*

    thus *?thesis*

      using *reduce-simple.select-false*

         reduce-simple.select-true

         expr-public-agree.intros(*2*)

         action-indistinguishable.intros(*1*)

         es′-def(*1,3*)

    by (*metis Secret action-indistinguishable.select es′-def*(*5*) *expr-public-agree-imp-expr-publics-agree*)

  next

    case *Public*

    hence *n* = *n′*

      using *es′-def*(*4*)

      unfolding *public-agree-def typeof-def t-sec-def*

      by *simp*

    thus *?thesis*

     using *reduce-simple.select-false*[*OF select-false*(*1*)] *expr-public-agree.intros*(*2*)

        action-indistinguishable.intros(*1*) *es′-def*(*1,3,5*)

     by *fastforce*

  qed

next

252

case (*select-true n v1 v2 sec sec′*)

then obtain *v1′ v2′ n′* **where** *es′-def*:*es′* = [$C *v1′*, $C *v2′*, $C *ConstInt32 sec n′*, $*Select sec*]

$$public\text{-}agree\ v1\ v1′$$
$$public\text{-}agree\ v2\ v2′$$
$$public\text{-}agree\ (ConstInt32\ sec\ n)\ (ConstInt32\ sec\ n′)$$
$$sec = sec′$$
$$typeof\ v1 = typeof\ v2$$
$$sec = Secret \longrightarrow is\text{-}secret\text{-}t\ (typeof\ v1)$$
$$sec = Secret \longrightarrow is\text{-}secret\text{-}t\ (typeof\ v2)$$

  using *exprs-public-agree-imp-publics-agree1-const3*[*OF select-true(2)*]
     typeof-select[*OF select-true(3)*] *public-agree-public-i32*

  unfolding *typeof-def t-sec-def*

  by (*fastforce simp add: expr-public-agree.simps*)

show *?case*

proof (*cases sec*)

  case *Secret*

  hence *v2-v1′*:*public-agree v1 v2′*

    using *es′-def(3,6,7,8) public-agree-def*

    by *auto*

  thus *?thesis*

    using *reduce-simple.select-false*

      reduce-simple.select-true

      expr-public-agree.intros(2)

      action-indistinguishable.intros(1)

      es′-def(1,2)

  by (*metis Secret action-indistinguishable.select es′-def(5) expr-public-agree-imp-expr-publics-agree*)

next

  case *Public*

  hence *n = n′*

    using *es′-def(4)*

    unfolding *public-agree-def typeof-def t-sec-def*

    by *simp*

  thus *?thesis*

    using *reduce-simple.select-true*[*OF select-true(1)*] *expr-public-agree.intros(2)*

      action-indistinguishable.intros(1) es′-def(1,2,5)

    by *fastforce*

  qed

next

  case (*block vs n t1s t2s m es*)

  obtain *vs′ e* **where** *es′-def*:*es′* = (*vs′* @ [*e*])

          exprs-public-agree vs vs′

          expr-public-agree ($*Block (t1s -> t2s) es*) *e*

          const-list vs′

  using *block(1,5) list-all2-append1*[*of expr-public-agree vs* [($*Block (t1s -> t2s) es*)]]

      exprs-public-agree-const-list

  by (*metis exprs-public-agree-imp-publics-agree1-const0*)

  then obtain *bes* **where** *e-def*:*e* = ($*Block (t1s -> t2s) bes*)

$$exprs\text{-}public\text{-}agree\ (\$*es)\ (\$*bes)$$
    using *exprs-public-agree-refl*
    unfolding *expr-public-agree.simps*[*of* ($Block (t1s -> t2s) es) e]
    by *auto*
  have *exprs-public-agree* [*Label m* [] (*vs* @ ($* es))] [*Label m* [] (*vs'* @ ($* bes))]
    using *expr-public-agree.intros*(6) *exprs-public-agree-refl*[*of* []]
       es'-def(2) *e-def*(2)
    by (*simp add*: *list-all2-appendI*)
  thus *?case*
     using *reduce-simple.block*[*OF es'-def*(4) - *block*(3,4)] *block*(2) *es'-def*(1)
list-all2-lengthD[*OF es'-def*(2)]
       e-def(1) *action-indistinguishable.intros*(1)
    by *fastforce*
next
  case (*loop vs n t1s t2s m es*)
  obtain *vs' e* **where** *es'-def*:*es'* = (*vs'* @ [*e*])
                  exprs-public-agree vs vs'
                  expr-public-agree ($Loop (t1s -> t2s) es) e
                  const-list vs'
    using *loop*(1,5) *list-all2-append1*[*of expr-public-agree vs* [($Loop (t1s -> t2s)
es)]]
       exprs-public-agree-const-list
    by (*metis exprs-public-agree-imp-publics-agree1-const0*)
  then obtain *bes* **where** *e-def*:*e* = ($Loop (t1s -> t2s) bes)
                 exprs-public-agree ($*es) ($*bes)
    using *exprs-public-agree-refl*
    unfolding *expr-public-agree.simps*[*of* ($Loop (t1s -> t2s) es) e]
    by *auto*
  have *exprs-public-agree* [*Label n* [$Loop (t1s -> t2s) es] (*vs* @ ($* es))]
                 [*Label n* [$Loop (t1s -> t2s) bes] (*vs'* @ ($* bes))]
   using *expr-public-agree.intros*(6)[*OF* - *list-all2-appendI*[*OF es'-def*(2) *e-def*(2)]]
       es'-def(3) *e-def*(1)
    by *fastforce*
  thus *?case*
  using *reduce-simple.loop*[*OF es'-def*(4) - *loop*(3,4)] *loop*(2) *es'-def*(1) *list-all2-lengthD*[*OF
es'-def*(2)]
       e-def(1) *action-indistinguishable.intros*(1)
    by *fastforce*
next
  case (*if-false n sec tf e1s e2s*)
  have *sec* = *Public*
    using *typeof-if*[*OF if-false*(3)]
    by −
  then obtain *e* **where** *es'-def*:*es'* = [$C ConstInt32 sec n, e]
                 expr-public-agree ($If tf e1s e2s) e
    using *exprs-public-agree-imp-publics-agree1-const1*[*OF if-false*(2)]
    unfolding *public-agree-def typeof-def t-sec-def*
    by *fastforce*
  moreover

then obtain *e1s′ e2s′* **where** *e-def*:*e* = ($*If tf e1s′ e2s′*)

                               exprs-public-agree ($**e1s*) ($**e1s′*)

                               exprs-public-agree ($**e2s*) ($**e2s′*)

  using *exprs-public-agree-refl*

  unfolding *expr-public-agree.simps*[*of* ($*If tf e1s e2s*)]

  by *fastforce*

moreover

have *exprs-public-agree* [$*Block tf e2s*] [$*Block tf e2s′*]

  using *expr-public-agree.intros*(*3*)[*OF e-def*(*3*)]

  by *simp*

ultimately

show *?case*

  using *reduce-simple.if-false if-false*(*1*) *action-indistinguishable.intros*(*1*)

  by *fastforce*

next

  case (*if-true n sec tf e1s e2s*)

  have *sec* = *Public*

    using *typeof-if*[*OF if-true*(*3*)]

    by −

  then obtain *e* **where** *es′-def*:*es′* = [$*C ConstInt32 sec n, e*]

                           expr-public-agree ($*If tf e1s e2s*) *e*

    using *exprs-public-agree-imp-publics-agree1-const1*[*OF if-true*(*2*)]

    unfolding *public-agree-def typeof-def t-sec-def*

    by *fastforce*

  moreover

  then obtain *e1s′ e2s′* **where** *e-def*:*e* = ($*If tf e1s′ e2s′*)

                                 exprs-public-agree ($**e1s*) ($**e1s′*)

                                 exprs-public-agree ($**e2s*) ($**e2s′*)

    using *exprs-public-agree-refl*

    unfolding *expr-public-agree.simps*[*of* ($*If tf e1s e2s*)]

    by *fastforce*

  moreover

  have *exprs-public-agree* [$*Block tf e1s*] [$*Block tf e1s′*]

    using *expr-public-agree.intros*(*3*)[*OF e-def*(*2*)]

    by *simp*

  ultimately

  show *?case*

    using *reduce-simple.if-true if-true*(*1*) *action-indistinguishable.intros*(*1*)

    by *fastforce*

next

  case (*label-const vs n les*)

  obtain *les′ vs′* **where** *es′* = [*Label n les′ vs′*]

                 exprs-public-agree les les′

                 exprs-public-agree vs vs′

    using *expr-public-agree-label exprs-public-agree-imp-publics-agree1-const0*[*OF label-const*(*2*)]

    by (*fastforce simp add*: *expr-public-agree.simps*)

  thus *?case*

    using *reduce-simple.label-const exprs-public-agree-const-list*[*OF - label-const*(*1*)]

255

    action-indistinguishable.intros(1)
  by *fastforce*
next
  case (*label-trap n les*)
  obtain *les′* **where** *es′* = [*Label n les′* [*Trap*]]
                  exprs-public-agree les les′
    using *expr-public-agree-label exprs-public-agree-imp-publics-agree1-const0*[*OF label-trap(1)*]
        exprs-public-agree-imp-publics-agree1-const0[*of Trap*]
    by (*fastforce simp add*: *expr-public-agree.simps*)
  thus *?case*
   using *reduce-simple.label-trap exprs-public-agree-refl*[*of* [*Trap*]] *action-indistinguishable.intros(1)*
    by *fastforce*
next
  case (*br vs n i lholed LI es*)
  obtain *les LI′* **where** *es′-def*:*es′* = [*Label n les LI′*]
                      exprs-public-agree es les
                      exprs-public-agree LI LI′
    using *expr-public-agree-local exprs-public-agree-imp-publics-agree1-const0*[*OF br(4)*]
    by (*fastforce simp add*: *expr-public-agree.simps*)
  obtain *lholed′ vs′* **where** *les-def*:*Lfilled i lholed′* (*vs′* @ [*$Br i*]) *LI′*
                      lholed-public-agree lholed lholed′
                      exprs-public-agree vs vs′
   using *exprs-public-agree-imp-lholed-public-agree*[*OF br(3) es′-def(3)*]
        exprs-public-agree-imp-publics-agree1-const0[*of $Br i*]
   unfolding *list-all2-append1*[*of expr-public-agree vs* [*$Br i*]]
   by (*fastforce simp add*: *expr-public-agree.simps*)
  show *?case*
   using *reduce-simple.br*[*OF - - les-def(1)*] *les-def(3) br(1,2)*
      exprs-public-agree-const-list[*OF les-def(3)*] *es′-def(1,2)*
     list-all2-lengthD[*OF les-def(3)*] *list-all2-appendI action-indistinguishable.intros(1)*
    by *fastforce*
next
  case (*br-if-false n sec i*)
  have *sec* = *Public*
    using *typeof-br-if*[*OF br-if-false(3)*]
    by −
  hence *es′-def*:*es′* = [*$C ConstInt32 sec n*, (*$Br-if i*)]
    using *exprs-public-agree-imp-publics-agree1-const1*[*OF br-if-false(2)*]
    unfolding *public-agree-def typeof-def t-sec-def*
    by (*fastforce simp add*: *expr-public-agree.simps*)
  thus *?case*
   using *reduce-simple.br-if-false*[*OF br-if-false(1)*] *action-indistinguishable.intros(1)*
    by *blast*
next
  case (*br-if-true n sec i*)
  have *sec* = *Public*
    using *typeof-br-if*[*OF br-if-true(3)*]

by −
  hence *es′-def*:*es′* = [$*C ConstInt32 sec n*, ($*Br-if i*)]
    using *exprs-public-agree-imp-publics-agree1-const1*[*OF br-if-true*(*2*)]
    unfolding *public-agree-def typeof-def t-sec-def*
    by (*fastforce simp add*: *expr-public-agree.simps*)
  thus *?case*
    using *reduce-simple.br-if-true*[*OF br-if-true*(*1*)] *expr-public-agree.intros*(*1*)
        action-indistinguishable.intros(*1*)
    by *blast*
next
  case (*br-table is c sec i′*)
  have *sec* = *Public*
    using *typeof-br-table*[*OF br-table*(*3*)]
    by −
  hence *es′-def*:*es′* = [$*C ConstInt32 sec c*, ($*Br-table is i′*)]
    using *exprs-public-agree-imp-publics-agree1-const1*[*OF br-table*(*2*)]
    unfolding *public-agree-def typeof-def t-sec-def*
    by (*fastforce simp add*: *expr-public-agree.simps*)
  thus *?case*
    using *reduce-simple.br-table*[*OF br-table*(*1*)] *expr-public-agree.intros*(*1*)
        action-indistinguishable.intros(*1*)
    by *blast*
next
  case (*br-table-lengthis c sec i′*)
  have *sec* = *Public*
    using *typeof-br-table*[*OF br-table-length*(*3*)]
    by −
  hence *es′-def*:*es′* = [$*C ConstInt32 sec c*, ($*Br-table is i′*)]
    using *exprs-public-agree-imp-publics-agree1-const1*[*OF br-table-length*(*2*)]
    unfolding *public-agree-def typeof-def t-sec-def*
    by (*fastforce simp add*: *expr-public-agree.simps*)
  thus *?case*
   using *reduce-simple.br-table-length*[*OF br-table-length*(*1*)] *expr-public-agree.intros*(*1*)
        action-indistinguishable.intros(*1*)
    by *blast*
next
  case (*local-const es n i vs*)
  obtain *vs′ ves* **where** *es′* = [*Local n i vs′ ves*]
                exprs-public-agree es ves
                publics-agree vs vs′
     using *expr-public-agree-local exprs-public-agree-imp-publics-agree1-const0*[*OF*
local-const(*3*)]
    by *fastforce*
  thus *?case*
   using *reduce-simple.local-const exprs-public-agree-const-list*[*OF - local-const*(*1*)]
        local-const(*2*) *list-all2-lengthD action-indistinguishable.intros*(*1*)
    by *fastforce*
next
  case (*local-trap n i vs*)

257

obtain vs' **where** $es' = [Local\ n\ i\ vs'\ [Trap]]$
$\qquad\qquad publics\text{-}agree\ vs\ vs'$
$\quad$ **using** $expr\text{-}public\text{-}agree\text{-}local\ exprs\text{-}public\text{-}agree\text{-}imp\text{-}publics\text{-}agree1\text{-}const0\,[OF$
$local\text{-}trap(1)]$
$\qquad\quad exprs\text{-}public\text{-}agree\text{-}imp\text{-}publics\text{-}agree1\text{-}const0\,[of\ Trap]$
$\quad$ **by** ($fastforce\ simp\ add\colon expr\text{-}public\text{-}agree.simps$)
thus *?case*
$\;$ **using** $reduce\text{-}simple.local\text{-}trap\ exprs\text{-}public\text{-}agree\text{-}refl\,[of\ [Trap]]\ action\text{-}indistinguishable.intros(1)$
$\quad$ **by** $fastforce$
next
$\quad$ **case** ($return\ vs\ n\ j\ lholed\ es\ i\ vls$)
$\quad$ **obtain** $vls'\ les$ **where** $es'\text{-}def\colon es' = [Local\ n\ i\ vls'\ les]$
$\qquad\qquad\qquad\qquad publics\text{-}agree\ vls\ vls'$
$\qquad\qquad\qquad\qquad exprs\text{-}public\text{-}agree\ es\ les$
$\qquad$ **using** $expr\text{-}public\text{-}agree\text{-}local\ exprs\text{-}public\text{-}agree\text{-}imp\text{-}publics\text{-}agree1\text{-}const0\,[OF$
$return(4)]$
$\quad$ **by** ($fastforce\ simp\ add\colon expr\text{-}public\text{-}agree.simps$)
$\quad$ **obtain** $lholed'\ vs'$ **where** $les\text{-}def\colon Lfilled\ j\ lholed'\ (vs'\ @\ [\$Return])\ les$
$\qquad\qquad\qquad\qquad lholed\text{-}public\text{-}agree\ lholed\ lholed'$
$\qquad\qquad\qquad\qquad exprs\text{-}public\text{-}agree\ vs\ vs'$
$\quad$ **using** $exprs\text{-}public\text{-}agree\text{-}imp\text{-}lholed\text{-}public\text{-}agree\,[OF\ return(3)\ es'\text{-}def(3)]$
$\qquad\quad exprs\text{-}public\text{-}agree\text{-}imp\text{-}publics\text{-}agree1\text{-}const0\,[of\ \$Return]$
$\quad$ **unfolding** $list\text{-}all2\text{-}append1\,[of\ expr\text{-}public\text{-}agree\ vs\ [\$Return]]$
$\quad$ **by** ($fastforce\ simp\ add\colon expr\text{-}public\text{-}agree.simps$)
$\quad$ **show** *?case*
$\quad$ **using** $reduce\text{-}simple.return\,[OF\ \text{-}\ \text{-}\ les\text{-}def(1)]\ les\text{-}def(3)\ return(1,2)$
$\qquad\quad exprs\text{-}public\text{-}agree\text{-}const\text{-}list\,[OF\ les\text{-}def(3)]\ es'\text{-}def(1)$
$\qquad\quad list\text{-}all2\text{-}lengthD\,[OF\ les\text{-}def(3)]\ action\text{-}indistinguishable.intros(1)$
$\quad$ **by** $fastforce$
next
$\quad$ **case** ($tee\text{-}local\ v\ i$)
$\quad$ **obtain** v'' **where** $v\text{-}def\colon es' = [v'',\ (\$Tee\text{-}local\ i)]$
$\qquad\qquad\qquad expr\text{-}public\text{-}agree\ v\ v''$
$\quad$ **using** $tee\text{-}local(2)\ list\text{-}all2\text{-}Cons1\,[of\ expr\text{-}public\text{-}agree]$
$\quad$ **by** ($fastforce\ simp\ add\colon expr\text{-}public\text{-}agree.simps\,[of\ \$Tee\text{-}local\ i]$)
$\quad$ **thus** *?case*
$\quad$ **using** $reduce\text{-}simple.tee\text{-}local\ expr\text{-}public\text{-}agree\text{-}const\,[OF\ v\text{-}def(2)\ tee\text{-}local(1)]$
$\qquad\quad expr\text{-}public\text{-}agree\text{-}refl\ action\text{-}indistinguishable.intros(1)$
$\quad$ **by** $fastforce$
next
$\quad$ **case** ($trap\ es\ lholed$)
$\quad$ **have** $es' \neq [Trap]$
$\quad$ **proof** $-$
$\qquad$ **{**
$\qquad\quad$ **assume** $es' = [Trap]$
$\qquad\quad$ **hence** *False*
$\qquad\quad$ **using** $trap(1,3)\ exprs\text{-}public\text{-}agree\text{-}imp\text{-}publics\text{-}agree1\text{-}const0\,[OF\ exprs\text{-}public\text{-}agree\text{-}symm]$
$\qquad\qquad$ **by** ($fastforce\ simp\ add\colon expr\text{-}public\text{-}agree.simps\,[of\ Trap]$)
$\qquad$ **}**

thus *?thesis*
  by *blast*
qed
moreover
thus *?case*
 using *exprs-public-agree-imp-lholed-public-agree[OF trap(2,3)] action-indistinguishable.intros(1)*
     reduce-simple.trap exprs-public-agree-trap-imp-is-trap
  by *fastforce*
qed

lemma *exprs-public-agree-imp-reduce*:
  assumes $(\!|s;vs;es|\!)\ a \leadsto\text{-}\ i\ (\!|s\text{-}a;vs\text{-}a;es\text{-}a|\!)$
       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-labs, return := arb-return$|\!)$
       $\mathcal{S}{\cdot}\mathcal{C} \vdash es : (ts \text{ -> } ts′)$
  shows $\exists\ a′\ s′\text{-}a\ vs′\text{-}a\ es′\text{-}a.\ (\!|s′;vs′;es′|\!)\ a′\leadsto\text{-}\ i\ (\!|s′\text{-}a;vs′\text{-}a;es′\text{-}a|\!) \wedge$
                     exprs-public-agree es-a es′-a $\wedge$
                     publics-agree vs-a vs′-a $\wedge$
                     store-public-agree s-a s′-a $\wedge$
                     $(a \sim\text{-}a\ a′)$
  using *assms assms(1)*
proof (*induction arbitrary: s′ vs′ es′ arb-labs arb-return ts ts′ tvs $\mathcal{C}$ rule: reduce.induct*)
  case (*basic e a e′ s vs i*)
  show *?case*
    using *publics-agree-imp-reduce-simple[OF basic(1,2,9)]*
       reduce.intros(1)[of es′ - - s′ vs′] basic(3,4,8)
    by *fastforce*
next
  case (*call s vs j i*)
  have *es′ = [\$Call j]*
    using *call(1)*
    by (*simp add: list-all2-Cons1 expr-public-agree.simps*)
  hence $(\!|s′;vs′;es′|\!)\ Call\text{-}action\leadsto\text{-}i\ (\!|s′;vs′;[Callcl\ (sfunc\ s′\ i\ j)]|\!)$
    using *reduce.intros(2)*
    by *blast*
  thus *?case*
     using *call(2,3) store-public-agree-sfunc-eq exprs-public-agree-refl[of [Callcl*
(sfunc s i j)]]
       action-indistinguishable.intros(1)
    by *fastforce*
next
  case (*call-indirect-Some s i c cl j tf vs sec*)

hence *sec = Public*
  using *types-preserved-call-indirect-None*(*2*)
  by *blast*
hence *es′ = [$C ConstInt32 Public c, $Call-indirect j]*
  using *exprs-public-agree-imp-publics-agree1-const1*[*OF call-indirect-Some*(*4*)]
  unfolding *public-agree-def typeof-def t-sec-def*
  by (*fastforce simp add*: *expr-public-agree.simps*)
hence ⦇*s′;vs′;es′*⦈ (*Call-indirect-Some-action c*)↝-*i* ⦇*s′;vs′;*[*Callcl cl*]⦈
  using *reduce.intros*(*3*)[*OF - - call-indirect-Some*(*3*)] *call-indirect-Some*(*1,2,6*)
        store-public-agree-stab-eq store-public-agree-stypes-eq
  by *fastforce*
thus *?case*
 using *call-indirect-Some*(*5,6*) *store-public-agree-sfunc-eq exprs-public-agree-refl*[*of*
[*Callcl cl*]]
        action-indistinguishable.intros(*1*)
  by *fastforce*
next
  case (*call-indirect-None s i c cl j vs sec*)
  hence *sec = Public*
    using *types-preserved-call-indirect-None*(*2*)
    by *blast*
  hence *es′ = [$C ConstInt32 Public c, $Call-indirect j]*
    using *exprs-public-agree-imp-publics-agree1-const1*[*OF call-indirect-None*(*2*)]
    unfolding *public-agree-def typeof-def t-sec-def*
    by (*fastforce simp add*: *expr-public-agree.simps*)
  hence ⦇*s′;vs′;es′*⦈ (*Call-indirect-None-action c*)↝-*i* ⦇*s′;vs′;*[*Trap*]⦈
    using *reduce.intros*(*4*) *store-agree-imp-callcl-cond*[*OF call-indirect-None*(*4,1*)]
    by *fastforce*
  thus *?case*
   using *call-indirect-None*(*3,4*) *store-public-agree-sfunc-eq exprs-public-agree-refl*[*of*
[*Trap*]]
        action-indistinguishable.intros(*1*)
    by *fastforce*
next
  case (*callcl-native cl j tr t1s t2s ts es ves vcs n k m zs s vs i*)
  obtain *vcs′* **where** *vcs′-def*:*es′ = ((*$$∗vcs′*) @ [*Callcl cl*]) publics-agree vcs vcs′*
    using *callcl-native*(*2,8*) *exprs-public-agree-imp-publics-agree1*[*of vcs Callcl cl
es′*]
    by (*fastforce simp add*: *expr-public-agree.simps*)
  hence ⦇*s′;vs′;*(*$$∗vcs′*) @ [*Callcl cl*]⦈ (*Callcl-native-action n*)↝- *i* ⦇*s′;vs′;*[*Local
m j* (*vcs′ @ zs*) [*$Block* ([] -> *t2s*) *es*]]⦈
    using *reduce.callcl-native*[*OF - - - callcl-native*(*4,5,6,7*)] *list-all2-lengthD*[*OF
vcs′-def*(*2*)]
        callcl-native(*1,3*)
    by *fastforce*
  moreover
  have *expr-public-agree*
        (*Local m j* (*vcs @ zs*) [*$Block* ([] -> *t2s*) *es*])
          (*Local m j* (*vcs′ @ zs*) [*$Block* ([] -> *t2s*) *es*])

     using *expr-public-agree.intros(7)[OF - exprs-public-agree-refl] vcs′-def*
        list-all2-appendI[OF - publics-agree-refl[of zs]]
    by *fastforce*
  ultimately
  show *?case*
    using *callcl-native(9,10) vcs′-def(1) action-indistinguishable.intros(1)*
    by *fastforce*
next
  case (*callcl-host-Some cl tr t1s t2s f ves vcs n m s hs s-a vcs-a vs i s′*)
  obtain *vcs′* **where** *es′-def*:*es′* = (($\$\$*vcs′$) @ [(*Callcl cl*)])
                 publics-agree vcs vcs′
   using *exprs-public-agree-imp-publics-agree1[of vcs (Callcl cl)] callcl-host-Some(2,7)*
   by (*fastforce simp add*: *expr-public-agree.simps*)
  have *tr-def*:*tr = Untrusted*
    using *typeof-callcl-host callcl-host-Some(1,2,13,14)*
    unfolding *trust-compat-def*
    by *fastforce*
  obtain *s′-a vcs′-a* **where** *host-def*:*host-apply s′ (t1s -> t2s) f vcs′ hs = Some (s′-a, vcs′-a)*
                     store-public-agree s-a s′-a
                     publics-agree vcs-a vcs′-a
    using *es′-def(2) callcl-host-Some(6,9) host-trust-security-Some*
    by *blast*
  have *length vcs′ = n*
    using *list-all2-lengthD callcl-host-Some(3) es′-def(2)*
    by *fastforce*
  thus *?case*
    using *reduce.callcl-host-Some[OF callcl-host-Some(1) - - callcl-host-Some(4,5) host-def(1)]*
      es′-def callcl-host-Some(8,9) host-def(2,3) action-indistinguishable.host-Some tr-def
      publics-agree-imp-exprs-public-agree[OF host-def(3) exprs-public-agree-refl[of []]]
    by (*metis append-Nil2*)
next
  case (*callcl-host-None cl tr t1s t2s f ves vcs n m s vs hs i*)
  obtain *vcs′* **where** *es′-def*:*es′* = (($\$\$*vcs′$) @ [(*Callcl cl*)])
                 publics-agree vcs vcs′
   using *exprs-public-agree-imp-publics-agree1[of vcs (Callcl cl)] callcl-host-None(2,6)*
   by (*fastforce simp add*: *expr-public-agree.simps*)
  have *tr-def*:*tr = Untrusted*
    using *typeof-callcl-host callcl-host-None(1,2,12,13)*
    unfolding *trust-compat-def*
    by *fastforce*
  have *length vcs′ = n*
    using *list-all2-lengthD callcl-host-None(3) es′-def(2)*
    by *fastforce*
  thus *?case*
    using *reduce.callcl-host-None[OF callcl-host-None(1) - - callcl-host-None(4,5)]*

   callcl-host-None(*7*,*8*) *exprs-public-agree-refl*[*of* [*Trap*]] *es′-def*
   action-indistinguishable.host-None tr-def
  by *metis*
next
 case (*get-local vi j s v vs i*)
 have *es′* = [\$*Get-local j*]
  using *get-local*(*2*) *exprs-public-agree-imp-publics-agree1*[*of* [] \$*Get-local j es′*]
  by *clarsimp* (*simp add*: *expr-public-agree.simps*)
 moreover
 obtain *vi′ v′ vs″* **where** *vs′-def*:*vs′* = *vi′* @ [*v′*] @ *vs″ publics-agree vi vi′*
publics-agree [*v*] [*v′*] *publics-agree vs vs″*
  using *get-local*(*3*) *list-all2-append1*[*of public-agree vi* ([*v*]@*vs*) *vs′*]
   list-all2-append1[*of public-agree* [*v*] *vs*]
  by (*metis publics-agree1*)
 ultimately
 have (|*s′*;*vi′*@[*v′*]@*vs″*;*es′*|)   *Get-local-action*⤳- *i* (|*s′*;*vi′*@[*v′*]@*vs″*;[\$*C v′*]|)
  using *reduce.get-local get-local*(*1*) *list-all2-lengthD*
  by *fastforce*
 thus *?case*
 using *vs′-def*(*1*,*3*) *get-local*(*3*,*4*) *public-agree-imp-expr-public-agree action-indistinguishable.intros*(*1*)
  by *fastforce*
next
 case (*set-local vi j s v vs v′ i*)
 obtain *v″* **where** *v″-def*:*es′* = [\$*C v″*, \$*Set-local j*] *public-agree v′ v″*
  using *exprs-public-agree-imp-publics-agree1-const1*[*OF set-local*(*2*)]
  by *clarsimp* (*auto simp add*: *expr-public-agree.simps*)
 moreover
 obtain *vi-a v-a vs-a* **where** *vs′-def*:*vs′* = *vi-a* @ [*v-a*] @ *vs-a publics-agree vi*
vi-a publics-agree [*v*] [*v-a*] *publics-agree vs vs-a*
  using *set-local*(*3*) *list-all2-append1*[*of public-agree vi* ([*v*]@*vs*) *vs′*]
   list-all2-append1[*of public-agree* [*v*] *vs*]
  by (*metis publics-agree1*)
 ultimately
 have (|*s′*;*vi-a*@[*v-a*]@*vs-a*;*es′*|)   *Set-local-action*⤳- *i* (|*s′*;*vi-a*@[*v″*]@*vs-a*;[]|)
  using *reduce.set-local set-local*(*1*) *list-all2-lengthD*
  by *fastforce*
 thus *?case*
  using *vs′-def v″-def*(*2*) *set-local*(*3*,*4*) *action-indistinguishable.intros*(*1*)
  by (*fastforce simp add*: *list-all2-appendI*)
next
 case (*get-global s vs j i*)
 have *es′* = [\$*Get-global j*]
  using *get-global*(*1*) *exprs-public-agree-imp-publics-agree1*[*of* [] \$*Get-global j es′*]
  by *simp* (*fastforce simp add*: *expr-public-agree.simps*)
 hence (|*s′*;*vs′*;*es′*|)   *Get-global-action*⤳- *i* (|*s′*;*vs′*;[\$*C sglob-val s′ i j*]|)
  using *reduce.get-global*
  by *fastforce*
 moreover
 have *public-agree* (*sglob-val s i j*) (*sglob-val s′ i j*)

proof −
  have *length (global C) > j*
   using *b-e-type-get-global get-global(8) unlift-b-e[of - - [Get-global j] (ts -> ts′)]*
   by *fastforce*
  hence *sglob-ind s i j < length (s.globs s)*
   using *get-global(6,7) store-typing-imp-glob-agree(1)[OF get-global(4)]*
     store-typing-imp-inst-length-eq[OF get-global(4)]
     store-typing-imp-glob-length-eq[OF get-global(4)]
   by *fastforce*
  thus *?thesis*
   using *store-public-agree-sglob-val-agree[OF get-global(3)]*
   by *fastforce*
 qed
 ultimately
 show *?case*
  using *get-global(2,3) expr-public-agree.intros(2) action-indistinguishable.intros(1)*
  by *fastforce*
next
 case (*set-global s i j v s-a vs s′*)
 obtain *v″* **where** *es′-def:es′ = [$C v″, $Set-global j]*
               public-agree v v″
  using *exprs-public-agree-imp-publics-agree1-const1[OF set-global(2)]*
  by (*fastforce simp add: expr-public-agree.simps*)
 have *length (global C) > j*
  using *b-e-type-set-global set-global(9) b-e-type-comp2-unlift*
   e-type-comp-conc1[of S C [$C v] [$Set-global j] ts ts′]
  by *blast*
 moreover
 obtain *k* **where** *k-def:sglob-ind s i j = k*
  by *blast*
 ultimately
 have *k < length (s.globs s)*
  using *set-global(7,8) store-typing-imp-glob-agree(1)[OF set-global(5)]*
   store-typing-imp-inst-length-eq[OF set-global(5)]
   store-typing-imp-glob-length-eq[OF set-global(5)]
  by *fastforce*
 moreover
 hence *k′-def:global-public-agree ((s.globs s)!k) ((s.globs s′)!k) sglob-ind s′ i j =*
k
  using *k-def store-public-agree-sglob-ind-eq[OF set-global(4)] set-global(4) list-all2-nthD*
  unfolding *store-public-agree-def*
  by *fastforce+*
 hence *global-public-agree (((s.globs s)!k)(|g-val := v|)) (((s.globs s′)!k)(|g-val :=*
v″|))
  using *es′-def(2)*
  unfolding *global-public-agree-def*
  by *fastforce*
 ultimately
 have *globals-public-agree ((globs s)[k:=((globs s)!k)(|g-val := v|)])*

263

$$((\textit{globs } s')[k:=((\textit{globs } s')!k)(\!|\textit{g-val} := v''|\!)])$$
      using *store-public-agree-sglob-ind-eq*[*OF set-global*(*4*)] *list-all2-update-cong*[*of*
global-public-agree]
          set-global(*4*)
    unfolding *store-public-agree-def*
    by *fastforce*
  hence *store-public-agree* (*supdate-glob s i j v*) (*supdate-glob s' i j v''*)
    using *set-global*(*4*) *k-def k'-def*(*2*)
    unfolding *store-public-agree-def supdate-glob-def supdate-glob-s-def*
    by *simp*
  thus *?case*
    using *es'-def reduce.set-global set-global*(*1,3*) *exprs-public-agree-refl*[*of* []]
        action-indistinguishable.intros(*1*)
    by *fastforce*
next
  case (*load-Some s i j m sec k off t bs vs sec' a*)
  have *t-sec-is*:*t-sec t = sec*
    using *load-helper*[*OF load-Some*(*1,2,7,9,10,11*)]
    by −
  obtain *m' bs'* **where** *m'-def*:*smem-ind s' i = Some j*
                      s.mem s' ! j = (m', sec)
                      memory-public-agree (*s.mem s ! j*) (*s.mem s' ! j*)
                      load m' (nat-of-int k) off (t-length t) = Some bs'
    using *memories-public-agree-helper*[*OF load-Some*(*1,6,7,9,2*)]
        load-m-imp-load-m'[*OF - load-Some*(*2*) *- load-Some*(*3*)]
    by *fastforce*
  have *sec-def*:*sec' = Public*
    using *types-preserved-load*(*2*)[*OF load-Some*(*11*)] *exists-v-typeof*
    by *fastforce*
  hence *es' =* [$C *ConstInt32 sec' k*, $*Load t None a off*]
    using *exprs-public-agree-imp-publics-agree1-const1*[*OF load-Some*(*4*)]
    by (*fastforce simp add*: *expr-public-agree.simps public-agree-def typeof-def t-sec-def*)
    hence $(\!|s';vs';es'|\!)$ (*Load-Some-action t* (*nat-of-int k*) *a off*)$\leadsto$- *i* $(\!|s';vs';$[$C
wasm-deserialise bs' t]$|\!)$
    using *reduce.load-Some*[*OF m'-def*(*1,2,4*)]
    by *fastforce*
  moreover
  have *public-agree* (*wasm-deserialise bs t*) (*wasm-deserialise bs' t*)
  proof (*cases sec*)
    case *Secret*
    thus *?thesis*
      using *wasm-deserialise-type t-sec-is*
      unfolding *public-agree-def*
      by *auto*
  next
    case *Public*
    thus *?thesis*
      using *load-Some*(*2,3*) *m'-def*(*2,3,4*)
      unfolding *public-agree-def memory-public-agree-def*

      by *auto*
    qed
    ultimately
    show *?case*
     using *load-Some(5,6) expr-public-agree.intros(2) action-indistinguishable.intros(1)*
      by *fastforce*
next
  case (*load-None s i j m sec k off t vs sec′ a*)
  have *sec-def:sec′ = Public*
   using *types-preserved-load(2)[OF load-None(11)] exists-v-typeof*
   by *fastforce*
  hence *es′ = [$C ConstInt32 sec′ k, $Load t None a off]*
   using *exprs-public-agree-imp-publics-agree1-const1[OF load-None(4)]*
  by (*fastforce simp add: expr-public-agree.simps public-agree-def typeof-def t-sec-def*)
  hence $(\!| s′;vs′;es′ |\!)$ (*Load-None-action t (nat-of-int k) a off*)⤳- *i* $(\!| s′;vs′;[Trap] |\!)$
   using *reduce.load-None memories-public-agree-helper[OF load-None(1,6,7,9,2)]*
     load-None(2,3) load-size
   by (*metis prod.sel(1) memory-public-agree-def*)
  thus *?case*
   using *load-None(5,6) exprs-public-agree-refl[of [Trap]] action-indistinguishable.intros(1)*
   by *fastforce*
next
  case (*load-packed-Some s i j m sec sx k off tp t bs vs sec′ a*)
  have *t-sec-is:t-sec t = sec*
   using *load-helper[OF load-packed-Some(1,2,7,9,10,11)]*
   by −
  obtain *m′ bs′* **where** *m′-def:smem-ind s′ i = Some j*
                     s.mem s′ ! j = (m′, sec)
                     memory-public-agree (s.mem s ! j) (s.mem s′ ! j)
                     load-packed sx m′ (nat-of-int k) off (tp-length tp) (t-length
t) = Some bs′
   using *memories-public-agree-helper[OF load-packed-Some(1,6,7,9,2)]*
    load-packed-m-imp-load-packed-m′[OF - load-packed-Some(2) - load-packed-Some(3)]
   by *fastforce*
  have *sec-def:sec′ = Public*
   using *types-preserved-load(2)[OF load-packed-Some(11)] exists-v-typeof*
   by *fastforce*
  hence *es′ = [$C ConstInt32 sec′ k, $Load t (Some (tp, sx)) a off]*
   using *exprs-public-agree-imp-publics-agree1-const1[OF load-packed-Some(4)]*
   by (*fastforce simp add: expr-public-agree.simps public-agree-def typeof-def t-sec-def*)
   hence $(\!| s′;vs′;es′ |\!)$ (*Load-packed-Some-action tp sx (nat-of-int k) a off*)⤳- *i*
$(\!| s′;vs′;[$C wasm-deserialise bs′ t] |\!)$
   using *reduce.load-packed-Some[OF m′-def(1,2,4)]*
   by *fastforce*
  moreover
  have *public-agree (wasm-deserialise bs t) (wasm-deserialise bs′ t)*
  proof (*cases sec*)
   case *Secret*
   thus *?thesis*

<div align="center">265</div>

   using *wasm-deserialise-type t-sec-is*
   unfolding *public-agree-def*
   by *auto*
  next
   case *Public*
   thus *?thesis*
    using *load-packed-Some(2,3) m'-def(2,3,4)*
    unfolding *public-agree-def memory-public-agree-def*
    by *auto*
  qed
  ultimately
  show *?case*
   using *load-packed-Some(5,6) expr-public-agree.intros(2) action-indistinguishable.intros(1)*
   by *fastforce*
next
  case (*load-packed-None s i j m sec sx k off tp t vs sec′ a*)
  have *sec-def*:*sec′ = Public*
   using *types-preserved-load(2)[OF load-packed-None(11)] exists-v-typeof*
   by *fastforce*
  hence *es′ = [$C ConstInt32 sec′ k, $Load t (Some (tp, sx)) a off]*
   using *exprs-public-agree-imp-publics-agree1-const1[OF load-packed-None(4)]*
   by (*fastforce simp add: expr-public-agree.simps public-agree-def typeof-def t-sec-def*)
  hence $(\!|s′;vs′;es′|\!)$ (*Load-packed-None-action tp sx (nat-of-int k) a off*)⤳- *i*
$(\!|s′;vs′;[Trap]|\!)$
   using *reduce.load-packed-None memories-public-agree-helper[OF load-packed-None(1,6,7,9,2)]*
     load-packed-None(2,3) load-packed-size memory-public-agree-imp-eq-length
   by *auto*
  thus *?case*
   using *load-packed-None(5,6) exprs-public-agree-refl[of [Trap]] action-indistinguishable.intros(1)*
   by *fastforce*
next
  case (*store-Some t v s i j m sec k off mem′ vs sec′ a*)
  obtain *v′ v″ m′* **where** *helpers*:*t-sec t = sec*
           sec′ = Public
           types-agree t v
           es′= [$C v′, $C v″, $Store t None 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)
           s.mem s′ ! j = (m′, sec)
   using *store-helper[OF store-Some(2,3,5,7,8,10,11,12)]*
   by *auto*
  obtain *mem″* **where** *store-def*:*store m′ (nat-of-int k) off (bits v″) (t-length t)
= Some mem″*
           memory-public-agree (mem′,sec) (mem″,sec)
   using *store-m-imp-store-m′*

    store-Some(*3,4*) *helpers*(*1,3,6,10,11*)
  by *blast*
 hence *store-public-agree*
   (*s*⦇*s.mem* := *s.mem s*[*j* := (*mem′, sec*)]⦈) (*s′*⦇*s.mem* := *s.mem s′* [*j* :=
(*mem″, sec*)]⦈)
  using *store-Some*
  unfolding *store-public-agree-def*
  by (*simp add*: *list-all2-update-cong*)
 thus *?case*
  using *reduce.store-Some*[*OF - helpers*(*7,11*) *store-def*(*1*)]
   public-agree-imp-types-agree-insecure[*OF store-Some*(*1*) *helpers*(*6*)] *helpers*(*4,5*)
    exprs-public-agree-refl[*of* [*Trap*]]
    store-Some(*6*) *action-indistinguishable.intros*(*1*)
  by *fastforce*
next
 case (*store-None t v s i j m sec k off vs sec′ a*)
 obtain *v′ v″ m′* **where** *helpers*:*t-sec t* = *sec*
          sec′ = *Public*
          types-agree t v
          es′= [\$*C v′*, \$*C v″*, \$*Store t None 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*)
          s.mem s′ ! *j* = (*m′, sec*)
  using *store-helper*[*OF store-None*(*2,3,5,7,8,10,11,12*)]
  by *auto*
 have *store-def*:*store m′* (*nat-of-int k*) *off* (*bits v″*) (*t-length t*) = *None*
  using *store-size1 store-None*(*3,4*) *helpers*(*10,11*)
  unfolding *memory-public-agree-def*
  by *fastforce*
 show *?case*
  using *reduce.store-None*[*OF - helpers*(*7,11*) *store-def*]
   public-agree-imp-types-agree-insecure[*OF store-None*(*1*) *helpers*(*6*)] *helpers*(*4,5*)
    exprs-public-agree-refl[*of* [*Trap*]]
    store-None(*6,7*) *action-indistinguishable.intros*(*1*)
  by *blast*
next
 case (*store-packed-Some t v s i j m sec k off tp mem vs sec′ a*)
 obtain *v′ v″ m′* **where** *helpers*:*t-sec t* = *sec*
          sec′ = *Public*
          types-agree t v
          es′= [\$*C v′*, \$*C v″*, \$*Store t* (*Some tp*) *a off*]
          v′ = (*ConstInt32 sec′ k*)
          public-agree v v″
          smem-ind s′ i = *Some j*
          j < *length* (*s.mem s′*)

267

$$\textit{memories-public-agree} \ (s.mem \ s) \ (s.mem \ s')$$
$$\textit{memory-public-agree} \ ((s.mem \ s)!j) \ ((s.mem \ s')!j)$$
$$s.mem \ s' \ ! \ j = (m', \ sec)$$
 using *store-helper*[*OF store-packed-Some*(*2,3,5,7,8,10,11,12*)]
 by *auto*
 obtain *mem'* **where** *store-def*:*store-packed m'* (*nat-of-int k*) *off* (*bits v''*) (*tp-length tp*) = *Some mem'*
$$\textit{memory-public-agree} \ (mem,sec) \ (mem',sec)$$
 using *store-packed-m-imp-store-packed-m'*
  store-packed-Some(*3,4*) *helpers*(*1,3,6,10,11*)
 by *blast*
 hence *store-public-agree*
  ($s(\![s.mem := s.mem \ s[j := (mem, \ sec)]\!]\!)$) ($s'(\![s.mem := s.mem \ s' \ [j := (mem', \ sec)]\!]\!)$)
 using *store-packed-Some*
 unfolding *store-public-agree-def*
 by (*simp add: list-all2-update-cong*)
 thus *?case*
 using *reduce.store-packed-Some*[*OF - helpers*(*7,11*) *store-def*(*1*)]
  public-agree-imp-types-agree-insecure[*OF store-packed-Some*(*1*) *helpers*(*6*)]
helpers(*4,5*)
  exprs-public-agree-refl[*of* [*Trap*]]
  store-packed-Some(*6*) *action-indistinguishable.intros*(*1*)
 by *fastforce*
next
 case (*store-packed-None t v s i j m sec k off tp vs sec' a*)
 obtain *v' v'' m'* **where** *helpers*:*t-sec t = sec*
$$sec' = Public$$
$$\textit{types-agree} \ t \ v$$
$$es'= [\$C \ v', \ \$C \ v'', \ \$Store \ t \ (Some \ tp) \ a \ off]$$
$$v' = (ConstInt32 \ sec' \ k)$$
$$\textit{public-agree} \ v \ v''$$
$$\textit{smem-ind} \ s' \ i = Some \ j$$
$$j < length \ (s.mem \ s')$$
$$\textit{memories-public-agree} \ (s.mem \ s) \ (s.mem \ s')$$
$$\textit{memory-public-agree} \ ((s.mem \ s)!j) \ ((s.mem \ s')!j)$$
$$s.mem \ s' \ ! \ j = (m', \ sec)$$
 using *store-helper*[*OF store-packed-None*(*2,3,5,7,8,10,11,12*)]
 by *auto*
 have *store-def*:*store-packed m'* (*nat-of-int k*) *off* (*bits v''*) (*tp-length tp*) = *None*
 using *store-packed-size1 store-packed-None*(*3,4*) *helpers*(*10,11*)
 unfolding *memory-public-agree-def*
 by *fastforce*
 show *?case*
 using *reduce.store-packed-None*[*OF - helpers*(*7,11*) *store-def*]
  public-agree-imp-types-agree-insecure[*OF store-packed-None*(*1*) *helpers*(*6*)]
helpers(*4,5*)
  exprs-public-agree-refl[*of* [*Trap*]]
  store-packed-None(*6,7*) *action-indistinguishable.intros*(*1*)

by *blast*
next
 case (*current-memory s i j m sec n vs*)
 have *es′* = [$*Current-memory*]
  using *exprs-public-agree-imp-publics-agree1-const0*[*OF current-memory(4)*]
  by (*fastforce simp add: expr-public-agree.simps*)
 thus *?case*
  using *reduce.current-memory*[*of s′ i j - sec n vs′*] *action-indistinguishable.intros(1)*
      memories-public-agree-helper[*OF current-memory(1,6,7,9,2)*]
     current-memory(2,5,6) memory-public-agree-imp-eq-length current-memory(3)
      exprs-public-agree-refl[*of* [$*C ConstInt32 Public (int-of-nat n)*]]
  by *fastforce*
next
 case (*grow-memory s i j m sec n c mem′ vs sec′*)
 hence *sec′* = *Public*
  using *types-preserved-grow-memory*
  by *blast*
 moreover
 hence *es′* = [$*C ConstInt32 Public c*, $*Grow-memory*]
  using *exprs-public-agree-imp-publics-agree1-const1*[*OF grow-memory(5)*]
  unfolding *public-agree-def typeof-def t-sec-def*
  by (*fastforce simp add: expr-public-agree.simps*)
 moreover
 obtain *m′* **where** *mem-agree*:*smem-ind s′ i = Some j*
             j < length (s.mem s′)
             memories-public-agree (s.mem s) (s.mem s′)
             memory-public-agree ((s.mem s)!j) ((s.mem s′)!j)
             s.mem s′ ! j = (m′, sec)
  using *memories-public-agree-helper*[*OF grow-memory(1,7,8,10,2)*]
  by *auto*
 moreover
 hence *mem-size m′ = mem-size m*
     sec = Public ⟹ m = m′
  using *grow-memory(2,3)*
  unfolding *memory-public-agree-def*
  by *auto*
 then obtain *mem″* **where** *mem″-def*:*mem-size m′ = mem-size m*
                 mem-grow m′ (nat-of-int c) = mem″
                 mem-size mem′ = mem-size mem″
                 sec = Public ⟹ mem′ = mem″
  using *mem-grow-size grow-memory(4)*
  by *metis*
 ultimately
 have (⦇*s′*;*vs′*;*es′*⦈) (*Grow-memory-Some-action n (nat-of-int c)*)⤳-*i* (⦇*s′*⦇*s.mem*
:= *s.mem s′*[*j := (mem″, sec)*]⦈⦈;*vs′*;[$*C ConstInt32 Public (int-of-nat n)*]⦈)
  using *reduce.grow-memory grow-memory(3)*
  by *fastforce*
 moreover
 have *memory-public-agree (mem′, sec) (mem″, sec)*

269

    using *mem″-def(3,4)*
    unfolding *memory-public-agree-def*
    by (*cases sec*) *auto*
  hence *store-public-agree* (*s*⦇*s.mem* := *s.mem s*[*j* := (*mem′, sec*)]⦈) (*s′*⦇*s.mem* := *s.mem s′*[*j* := (*mem″, sec*)]⦈)
    using *mem-agree(1) list-all2-update-cong grow-memory(7)*
    unfolding *store-public-agree-def*
    by *fastforce*
  ultimately
  show *?case*
  using *exprs-public-agree-refl*[*of* [$*C ConstInt32 Public* (*int-of-nat n*)]] *grow-memory(6)*
       action-indistinguishable.intros(1)
    by *fastforce*
next
  case (*grow-memory-fail s i j m sec n vs sec′ c*)
  hence *sec′ = Public*
    using *types-preserved-grow-memory*
    by *blast*
  hence *es′* = [$*C ConstInt32 Public c*, $*Grow-memory*]
    using *exprs-public-agree-imp-publics-agree1-const1*[*OF grow-memory-fail(4)*]
    unfolding *public-agree-def typeof-def t-sec-def*
    by (*fastforce simp add*: *expr-public-agree.simps*)
  moreover
  obtain *m′* **where** *mem-agree*:*smem-ind s′ i = Some j*
                   j < length (*s.mem s′*)
                   memories-public-agree (*s.mem s*) (*s.mem s′*)
                   memory-public-agree ((*s.mem s*)!*j*) ((*s.mem s′*)!*j*)
                   s.mem s′ ! *j* = (*m′, sec*)
    using *memories-public-agree-helper*[*OF grow-memory-fail(1,6,7,9,2)*]
    by *auto*
  hence *mem-size m′ = n*
    using *grow-memory-fail(2,3) mem-grow-size*
    unfolding *memory-public-agree-def*
    by *fastforce*
  ultimately
  have ⦇*s′*;*vs′*;*es′*⦈ (*Grow-memory-None-action n* (*nat-of-int c*))↝-*i* ⦇*s′*;*vs′*;[$*C ConstInt32 Public int32-minus-one*]⦈)
    using *reduce.grow-memory-fail*[*OF mem-agree(1,5)*]
    by *blast*
  thus *?case*
  using *exprs-public-agree-refl*[*of* [$*C ConstInt32 Public int32-minus-one*]] *grow-memory-fail(5,6)*
       action-indistinguishable.intros(1)
    by *fastforce*
next
  case (*label s vs es a i s-a vs-a es-a k lholed les les-a s′ vs′ les′*)
  obtain *lholed′ es′* **where** *lholed′-def*:*lholed-public-agree lholed lholed′*
                      exprs-public-agree es es′
                      Lfilled k lholed′ es′ les′
    using *exprs-public-agree-imp-lholed-public-agree*[*OF label(2,5)*]

270

by *blast*
obtain *a′ s′-a vs′-a es′-a* **where** *es′-a-def*:⦇*s′;vs′;es′*⦈ *a′⤳- i* ⦇*s′-a;vs′-a;es′-a*⦈
                          exprs-public-agree es-a es′-a
                          publics-agree vs-a vs′-a
                          store-public-agree s-a s′-a
                          a ∼-a a′
  using *label(4)[OF lholed′-def(2) label(6,7,8,9,10)]*
      types-exist-lfilled-weak[OF label(2,12)] label(1,11)
  by *fastforce*
obtain *les′-a* **where** *Lfilled k lholed′ es′-a les′-a exprs-public-agree les-a les′-a*
 using *lholed-public-agree-imp-exprs-public-agree[OF lholed′-def(1) label(3) es′-a-def(2)]*
  by *blast*
thus *?case*
  using *reduce.label[OF es′-a-def(1) lholed′-def(3)] es′-a-def(3,4,5)*
  by *blast*
next
  case (*local s vs es a i s-a vs-a es-a v0s n j s′ vs′ es′*)
  obtain *vs″ es″* **where** *es′-def*:*es′ = [Local n i vs″ es″]*
                             publics-agree vs vs″
                             exprs-public-agree es es″
  using *local(3) expr-public-agree-local[OF exprs-public-agree-imp-expr-public-agree]*
  by (*metis* (*no-types, lifting*) *list-all2-Cons1 list-all2-Nil*)
  obtain *tls C′* **where** *tls-def*:*i < length* (*inst s*)
                  C′ = (s-inst S ! i)⦇*trust-t := Untrusted, local := local* (*s-inst*
S ! i) *@ map typeof vs, label := label* (*s-inst S ! i*), *return := Some tls*⦈
                  S·C′ ⊢ es : ([] -> tls)
    using *e-type-local[OF local(10)] store-typing-imp-inst-length-eq[OF local(6)]*
local(9)
  by *fastforce*
  obtain *a′ s′-a vs′-a es′-a* **where** ⦇*s′;vs″;es″*⦈ *a′⤳- i* ⦇*s′-a;vs′-a;es′-a*⦈
                          exprs-public-agree es-a es′-a
                          publics-agree vs-a vs′-a
                          store-public-agree s-a s′-a
                          a ∼-a a′
  using *local(2)[OF es′-def(3,2) local(5,6) - tls-def local(1)]*
  by *fastforce*
  thus *?case*
  using *reduce.local es′-def(1) expr-public-agree.intros(7) local(4)*
  by *fastforce*
qed

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 S
      tvs = map typeof vs
      i < length (*inst s*)

$\mathcal{C} = ((s\text{-}inst\ \mathcal{S})!i)(\!\!|\mkern-2mu trust\text{-}t := Untrusted,\ local := (local\ ((s\text{-}inst\ \mathcal{S})!i)\ @$
$tvs),\ label := arb\text{-}labs,\ return := arb\text{-}return|\!\!)$
$\qquad \mathcal{S}\cdot\mathcal{C} \vdash es : (ts \rightarrow ts')$
  shows $\exists\ as'.\ (r\text{-}actions\ (\!\!|s';vs';es'|\!\!)\ i\ as') \wedge list\text{-}all2\ action\text{-}indistinguishable\ as$
as'
  using *assms*
proof (*induction s vs es i as arbitrary: s' vs' es' arb-labs arb-return* $\mathcal{C}$ *rule:*
reduction-actions.induct)
  case (*1 es s vs*)
  have *const-list es'* $\vee$ *es'* = [*Trap*]
    using *1*(*1*)
  proof (*rule disjE*)
    assume *const-list es*
    thus *?thesis*
      using *exprs-public-agree-const-list 1*(*2*)
      by *simp*
  next
    assume *local-assms*:*es* = [*Trap*]
    thus *?thesis*
      using *1*(*2*)
      by (*simp add*: *list-all2-Cons1 expr-public-agree.simps*)
  qed
  thus *?case*
    using *reduction-actions.intros*(*1*)
    by *fastforce*
next
  case (*2 s vs es a i s-a vs-a es-a as s' vs' es'*)
  have *store-typing s-a* $\mathcal{S}$ *i* < *length* (*inst s-a*)
  using *store-preserved1*[*OF 2*(*1,7,11*)] *2*(*7,8,9,10*) *store-typing-imp-inst-length-eq*
    by *fastforce*+
  moreover
  have $\mathcal{S}\cdot\mathcal{C} \vdash$ *es-a* : (*ts* -> *ts'*) *tvs* = *map typeof vs-a*
    using *types-preserved-e1*[*OF 2*(*1,7,8,9,10,11*)] *2*(*8*)
    by *auto*
  moreover
  obtain *a' s'-a vs'-a es'-a* **where** $(\!\!|s';vs';es'|\!\!)\ a' \rightsquigarrow$- *i* $(\!\!|s'\text{-}a;vs'\text{-}a;es'\text{-}a|\!\!)$
                              exprs-public-agree es-a es'-a
                              publics-agree vs-a vs'-a
                              store-public-agree s-a s'-a
                              $a \sim$-*a a'*
    using *exprs-public-agree-imp-reduce*[*OF 2*(*1,4,5,6,7,8,9,10,11*)]
    by *blast*
  ultimately
  show *?case*
    using *2*(*3,10*) *reduction-actions.intros*(*2*)
    by *fastforce*
qed

lemma *function-actions-indistinguishable-secrets*:

272

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 $\mathcal{S}$
      $\mathcal{S}{\cdot}Untrusted{\cdot}rs \Vdash\text{-}i\ vs;es : ts′$
shows $\exists\ as′.\ (r\text{-}actions\ (|s′;vs′;es′|)\ i\ as′) \wedge\ list\text{-}all2\ action\text{-}indistinguishable\ as$
as′
proof −
  obtain *tvs rs $\mathcal{C}$* **where** *$\mathcal{C}$-def:tvs = map typeof vs*
                    i < length (inst s)
                        $\mathcal{C} = (s\text{-}inst\ \mathcal{S}\ !\ i)(|trust\text{-}t := Untrusted,\ local := local$
(s-inst $\mathcal{S}$! i) @ tvs, label := label (s-inst $\mathcal{S}$! i), return := rs|)
                    $\mathcal{S}{\cdot}\mathcal{C} \vdash es : ([]\ \text{->}\ ts′)$
    using *assms(6) store-typing-imp-inst-length-eq[OF assms(5)]*
    unfolding *s-typing.simps*
    by *fastforce*
  thus *?thesis*
    using *actions-indistinguishable-secrets[OF assms(1,2,3,4,5) $\mathcal{C}$-def]*
    by *fastforce*
qed

theorem *config-actions-indistinguishable-secrets*:
  assumes $\vdash\text{-}i\ s;vs;es : (Untrusted,ts)$
      r-actions $(|s;vs;es|)$ i as
      exprs-public-agree es es′
      publics-agree vs vs′
      store-public-agree s s′
  shows $\exists\ as′.\ (r\text{-}actions\ (|s′;vs′;es′|)\ i\ as′) \wedge\ list\text{-}all2\ action\text{-}indistinguishable\ as$
as′
proof −
  obtain $\mathcal{S}$ **where** *store-typing s $\mathcal{S}$ $\mathcal{S}{\cdot}Untrusted{\cdot}None \Vdash\text{-}i\ vs;es : ts$*
    using *assms(1) config-typing.simps*
    by *blast*
  thus *?thesis*
    using *assms(2,3,4,5) function-actions-indistinguishable-secrets*
    by *blast*
qed

lemma *config-indistinguishable-imp-reduce*:
  assumes $(|s;vs;es|)\ a \rightsquigarrow\text{-}\ i\ (|s\text{-}a;vs\text{-}a;es\text{-}a|)$
      $(s,vs,es) \sim\text{-}c\ (s′,vs′,es′)$
      $\vdash\text{-}i\ s;vs;es : (Untrusted,ts)$
  shows $\exists\ a′\ s′\text{-}a\ vs′\text{-}a\ es′\text{-}a.\ (|s′;vs′;es′|)\ a′ \rightsquigarrow\text{-}\ i\ (|s′\text{-}a;vs′\text{-}a;es′\text{-}a|) \wedge$
                    $((s\text{-}a,vs\text{-}a,es\text{-}a) \sim\text{-}c\ (s′\text{-}a,vs′\text{-}a,es′\text{-}a)) \wedge$
                    $(a \sim\text{-}a\ a′)$
proof −
  obtain $\mathcal{S}$ **where** *$\mathcal{S}$-def:store-typing s $\mathcal{S}$*
                  $\mathcal{S}{\cdot}Untrusted{\cdot}None \Vdash\text{-}i\ vs;es : ts$

```
    using assms(3)
    unfolding config-typing.simps
    by blast
  have config-agree:exprs-public-agree es es′
                    publics-agree vs vs′
                    store-public-agree s s′
    using assms(2)
    by simp-all
  obtain tvs C where C-def:tvs = map typeof vs
                     i < length (inst s)
                     C = (s-inst S ! i) (|trust-t := Untrusted, local := local (s-inst
S ! i) @ tvs, label := label (s-inst S ! i), return := None|)
                     S·C ⊢ es : ([] -> ts)
    using S-def(2) store-typing-imp-inst-length-eq[OF S-def(1)]
    unfolding s-typing.simps
    by fastforce
  show ?thesis
    using exprs-public-agree-imp-reduce[OF assms(1) config-agree S-def(1) C-def]
    by fastforce
qed

definition config-bisimulation :: ((s × v list × e list) × nat) rel ⇒ bool where
  config-bisimulation R ≡
    ∀ (((s1,vs1,es1),i1),((s2,vs2,es2),i2)) ∈ R.
    (∀ s1′ vs1′ es1′ a. (|s1;vs1;es1|) a⤳-i1 (|s1′;vs1′;es1′|) ⟶ (∃ s2′ vs2′ es2′ a′.
(|s2;vs2;es2|) a′⤳-i2 (|s2′;vs2′;es2′|) ∧ (((s1′,vs1′,es1′),i1),((s2′,vs2′,es2′),i2)) ∈
R ∧ (a ∼-a a′)))
         ∧ (∀ s2′ vs2′ es2′ a. (|s2;vs2;es2|) a⤳-i2 (|s2′;vs2′;es2′|) ⟶ (∃ s1′ vs1′ es1′
a′. (|s1;vs1;es1|) a′⤳-i1 (|s1′;vs1′;es1′|) ∧ (((s1′,vs1′,es1′),i1),((s2′,vs2′,es2′),i2))
∈ R ∧ (a ∼-a a′)))

definition config-bisimilar :: ((s × v list × e list) × nat) rel where
  config-bisimilar ≡ ⋃ { R. config-bisimulation R }

lemma config-bisimilar-ex-config-bisimulation:
  assumes (((s,vs,es),i), ((s′,vs′,es′),i′)) ∈ config-bisimilar
  shows ∃ R. config-bisimulation R ∧ (((s,vs,es),i), ((s′,vs′,es′),i′)) ∈ R
  using assms
  unfolding config-bisimilar-def
  by simp

definition typed-indistinguishable-pairs :: ((s × v list × e list) × nat) rel where
  typed-indistinguishable-pairs ≡
    { (((s,vs,es),i1),((s′,vs′,es′),i2)). ((s,vs,es) ∼-c (s′,vs′,es′)) ∧ i1 = i2
                                  ∧ (∃ ts. ⊢-i1 s;vs;es : (Untrusted,ts))}

lemma config-bisimulation-typed-indistinguishable-pairs1:
  assumes (((s1,vs1,es1),i1), ((s2,vs2,es2),i2)) ∈ typed-indistinguishable-pairs
          (|s1;vs1;es1|) a1⤳-i1 (|s1′;vs1′;es1′|)
```

274

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

proof $-$
  have *bisim-is*:$(s1,vs1,es1) \sim\text{-}c\ (s2,vs2,es2)$
        $i1 = i2$
        $(\exists\, ts.\ \vdash\text{-}i1\ s1;vs1;es1 : (Untrusted,ts))$
    using *assms*(1)
    unfolding *typed-indistinguishable-pairs-def*
    by *auto*
  show *?thesis*
   using *config-indistinguishable-imp-reduce*[*OF assms*(2) *bisim-is*(1)] *bisim-is*(2,3)
preservation[*OF - assms*(2)]
    unfolding *typed-indistinguishable-pairs-def*
    by *fastforce*
qed

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

proof $-$
  obtain *ts* **where** *bisim-is*:$(s1,vs1,es1) \sim\text{-}c\ (s2,vs2,es2)$
           $i1 = i2$
           $(\vdash\text{-}i1\ s1;vs1;es1 : (Untrusted,ts))$
    using *assms*(1)
    unfolding *typed-indistinguishable-pairs-def*
    by *auto*
  have *c2-type*:$(s2,vs2,es2) \sim\text{-}c\ (s1,vs1,es1)$
      $(\vdash\text{-}i2\ s2;vs2;es2 : (Untrusted,ts))$
    using *exprs-public-agree-imp-config-typing*[*OF bisim-is*(3)] *bisim-is*(1)
      config-indistinguishable-symm[*OF bisim-is*(1)] *bisim-is*(2)
    by *fastforce+*
  show *?thesis*
   using *config-indistinguishable-imp-reduce*[*OF assms*(2) *config-indistinguishable-symm*[*OF*
bisim-is(1)] *c2-type*(2)] *bisim-is*(2)
      preservation[*OF bisim-is*(3)]
    unfolding *typed-indistinguishable-pairs-def*
    by (*fastforce simp only*: *config-indistinguishable-symm*)
qed

lemma *config-bisimulation-typed-indistinguishable-pairs*:
 config-bisimulation typed-indistinguishable-pairs
proof $-$
 {

    fix *s1 vs1 es1 i1 s2 vs2 es2 i2*
   assume *assms*:*(((s1,vs1,es1),i1),((s2,vs2,es2),i2)) ∈ typed-indistinguishable-pairs*
    have *(∀ s1 ′ vs1 ′ es1 ′ a. (|s1;vs1;es1|) a⤳-i1 (|s1 ′;vs1 ′;es1 ′|) ⟶*
         (∃ s2 ′ vs2 ′ es2 ′ a′. (|s2;vs2;es2|) a′⤳-i2 (|s2 ′;vs2 ′;es2 ′|)
      ∧ (((s1 ′,vs1 ′,es1 ′),i1),((s2 ′,vs2 ′,es2 ′),i2)) ∈ typed-indistinguishable-pairs
      ∧ (a ∼-a a′)))
      ∧ (∀ s2 ′ vs2 ′ es2 ′ a. (|s2;vs2;es2|) a⤳-i2 (|s2 ′;vs2 ′;es2 ′|) ⟶
         (∃ s1 ′ vs1 ′ es1 ′ a′. (|s1;vs1;es1|) a′⤳-i1 (|s1 ′;vs1 ′;es1 ′|)
      ∧ (((s1 ′,vs1 ′,es1 ′),i1),((s2 ′,vs2 ′,es2 ′),i2)) ∈ typed-indistinguishable-pairs
      ∧ (a ∼-a a′)))
     using *config-bisimulation-typed-indistinguishable-pairs1* [*OF assms*]
        config-bisimulation-typed-indistinguishable-pairs2 [*OF assms*]
    by *fastforce*
 }
 thus *?thesis*
  unfolding *config-bisimulation-def*
  by *fastforce*
qed

theorem *config-indistinguishable-imp-config-bisimilar*:
 assumes *(s,vs,es) ∼-c (s′,vs′,es′)*
    ⊢-i s;vs;es : (Untrusted,ts)
 shows *(((s,vs,es),i), ((s′,vs′,es′),i)) ∈ config-bisimilar*
proof −
 have *(((s,vs,es),i), ((s′,vs′,es′),i)) ∈ typed-indistinguishable-pairs*
  using *assms*
  unfolding *typed-indistinguishable-pairs-def*
  by *fastforce*
 thus *?thesis*
  using *config-bisimulation-typed-indistinguishable-pairs*
  unfolding *config-bisimilar-def*
  by *blast*
qed

inductive *reduce-relpowp :: s ⇒ v list ⇒ e list ⇒ action list ⇒ nat ⇒ s ⇒ v list*
⇒ e list ⇒ bool *((|-;-;-|) -ˆˆ⤳′- - (|-;-;-|) 60)* **where**
 (|s;vs;es|) [] ˆˆ⤳-i (|s;vs;es|)
| [[(|s;vs;es|) a⤳-i (|s ′′;vs ′′;es ′′|); (|s ′′;vs ′′;es ′′|) as ˆˆ⤳-i (|s′;vs′;es′|)]] ⟹ (|s;vs;es|)
(a#as) ˆˆ⤳-i (|s′;vs′;es′|)

theorem *config-indistinguishable-trace-noninterference*:
 assumes *(|s;vs;es|) as ˆˆ⤳-i (|s-as;vs-as;es-as|)*
    (s,vs,es) ∼-c (s′,vs′,es′)
    ⊢-i s;vs;es : (Untrusted,ts)
 shows *∃ s′-as vs′-as es′-as as′. (|s′;vs′;es′|) as′ˆˆ⤳-i (|s′-as;vs′-as;es′-as|) ∧*
            list-all2 action-indistinguishable as as′ ∧
        config-indistinguishable (s-as,vs-as,es-as) (s′-as,vs′-as,es′-as)
 using *assms*
proof (*induction s vs es as i s-as vs-as es-as arbitrary: s′ vs′ es′ rule: reduce-relpowp.induct*)

```
  case (1 s vs es i)
  thus ?case
    using reduce-relpowp.intros(1)
    by fastforce
next
  case (2 s vs es a i s-a vs-a es-a as s-aas vs-aas es-aas)
  obtain a' s'-a vs'-a es'-a where es'-a-def:(|s';vs';es'|) a'⤳- i (|s'-a;vs'-a;es'-a|)
                                  config-indistinguishable (s-a, vs-a, es-a) (s'-a,
vs'-a, es'-a)

                                  a ∼-a a'
    using config-indistinguishable-imp-reduce[OF 2(1,4,5)]
    by blast
  have ⊢- i s-a;vs-a;es-a : (Untrusted, ts)
    using preservation[OF 2(5,1)]
    by −
  thus ?case
   using 2(3)[OF es'-a-def(2)] reduce-relpowp.intros(2)[OF es'-a-def(1)] es'-a-def(2,3)
    by fastforce
qed


lemma rep-config-untrusted-quot-typing:
  assumes ((s,vs,es),i) = (rep-config-untrusted-quot x)
  shows ∃ ts. ⊢-i s;vs;es : (Untrusted,ts)
proof −
  have ((s,vs,es),i) ∼-cp ((s,vs,es),i)
    using Quotient3-config-untrusted-quot assms
    unfolding Quotient3-def
    by metis
  thus ?thesis
    unfolding config-untrusted-equiv-def
    by simp
qed

end
```

13   Constant Time (coinductive)

theory *Wasm-Constant-Time* **imports** *Wasm-Secret* **begin**

lemma *equivp-observation*: *equivp (llist-all2 action-indistinguishable)*
  using *equivp-action-indistinguishable reflp-llist-all2 symp-llist-all2 transp-llist-all2*
  unfolding *equivp-reflp-symp-transp*
  by *blast*

quotient-type (**overloaded**) *observation = action llist / llist-all2 action-indistinguishable*
  using *equivp-observation*
  by *blast*

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 ≡ λ*c 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 −
  def *transX* ≡ λ (*s,vs,es*) *a as*. (*SOME* ((*s′,vs′,es′*),*a′*). (|*s;vs;es*|) *a′*⤳- *i* (|*s′;vs′;es′*|) ∧ (*a* ∼-*a a′*) ∧ *X* (*s′,vs′,es′*) *i as*)
  def *realtr* ≡ λ*c b*. *unfold-llist*
    (λ(*c, abs*). *lnull abs*)
    (λ(*c, abs*). *snd* (*transX c* (*lhd abs*) (*ltl abs*)))
    (λ(*c, abs*). (*fst* (*transX c* (*lhd abs*) (*ltl abs*)), (*ltl abs*)))
    (*c, b*)
  have *realtr-simps*:
    ⋀*c*. *realtr c LNil* = *LNil*
    ⋀*c abs*. *lnull* (*realtr c abs*) ⟷ *lnull abs*
    ⋀*c abs*. ¬ *lnull abs* ⟹ *lhd* (*realtr c abs*) = *snd* (*transX c* (*lhd abs*) (*ltl abs*))
    ⋀*c abs*. ¬ *lnull abs* ⟹ *ltl* (*realtr c abs*) = *realtr* (*fst* (*transX c* (*lhd abs*) (*ltl*

abs))) (*ltl abs*)
$\bigwedge$*c tl abs. realtr c* (*LCons tl abs*) =
   LCons
    (*snd* (*transX c tl abs*))
     (*realtr* (*fst* (*transX c tl abs*)) *abs*)
  by (*simp-all add: realtr-def*)
have *config-is-trace xa i* (*realtr xa xb*)
  using *base step*
proof (*coinduction arbitrary: xa xb*)
  case *config-is-trace*
  show *?case*
  proof (*cases xb*)
    case *LNil*
    thus *?thesis*
     using *config-is-trace realtr-simps*(*1*)
     by (*cases xa*) *fastforce*
    next
    case (*LCons aa aas*)
    obtain *a as xa′* **where** *transX-is:realtr xa xb = LCons a as*
                       a = (*snd* (*transX xa aa aas*))
                       xa′ = (*fst* (*transX xa aa aas*))
                       as = (*realtr xa′ aas*)
     using *LCons realtr-simps*(*5*)
     by *simp*
    obtain *s′ vs′ es′* **where** *xa′-def:xa′* = (*s′,vs′,es′*)
     by (*metis prod.collapse*)
    obtain *s vs es* **where** *xa-def:xa* = (*s, vs, es*)
                    $\exists$ *a′ s′ vs′ es′ a tr.*
                    (|*s;vs;es*|) *a′⤳- i* (|*s′;vs′;es′*|) $\wedge$
                    (*aa ∼-a a′*) $\wedge$ *X* (*s′, vs′, es′*) *i aas*
     using *config-is-trace LCons*
     by *fastforce*
    hence (|*s;vs;es*|) *a ⤳- i* (|*s′;vs′;es′*|) $\wedge$ (*aa ∼-a a*) $\wedge$ *X* (*s′,vs′,es′*) *i aas*
     using *transX-is*(*2,3*) *xa′-def*
     unfolding *transX-def*
     by (*metis* (*mono-tags, lifting*) *prod.collapse someI split-conv*)
    thus *?thesis*
     using *transX-is config-is-trace*(*2*) *xa-def*(*1*) *xa′-def*
     by *force*
  qed
 qed
 moreover
have *llist-all2 action-indistinguishable xb* (*realtr xa xb*)
  using *base step*
proof (*coinduction arbitrary: xa xb*)
  case *LNil*
  thus *?case*
   by (*simp add: realtr-simps*(*2*))
  next

case *LCons*
  obtain *aa aas* **where** *xb-def*:*xb = LCons aa aas*
    by (*metis LCons(3) lhd-LCons-ltl*)
  obtain *a as xa′* **where** *transX-is*:*realtr xa xb = LCons a as*
                           a = (snd (transX xa aa aas))
                           xa′ = (fst (transX xa aa aas))
                           as = (realtr xa′ aas)
    using *realtr-simps(5) xb-def*
    by *simp*
    obtain *s′ vs′ es′* **where** *xa′-def*:*xa′ = (s′,vs′,es′)*
      by (*metis prod.collapse*)
    obtain *s vs es* **where** *xa-def*:*xa = (s, vs, es)*
                   $\exists$ *a′ s′ vs′ es′ a tr.*
                    (|*s*;*vs*;*es*|) *a′*⤳- *i* (|*s′*;*vs′*;*es′*|) $\wedge$
                    (aa ∼-a a′) $\wedge$ *X (s′, vs′, es′) i aas*
    using *xb-def LCons(1,2)*
    by *fastforce*
    hence (|*s*;*vs*;*es*|) *a* ⤳- *i* (|*s′*;*vs′*;*es′*|) $\wedge$ *(aa ∼-a a)* $\wedge$ *X (s′,vs′,es′) i aas*
    using *transX-is(2,3) xa′-def*
    unfolding *transX-def*
    by (*metis (mono-tags, lifting) prod.collapse someI split-conv*)
  thus *?case*
    using *LCons(2) transX-is(1,4) xb-def xa′-def*
    by *auto*
  qed
  ultimately
  show *?thesis*
    unfolding *ct-prop-def*
    by *blast*
qed

lemma *config-indistinguishable-imp-reduce2*:
  assumes *(s,vs,es)* ∼-c *(s′,vs′,es′)*
         ⊢-*i s*;*vs*;*es* : *(Untrusted,ts)*
         config-is-trace (s,vs,es) i tr
  shows *ct-prop (s′,vs′,es′) i tr*
  using *assms*
proof (*coinduction arbitrary*: *s vs es s′ vs′ es′ tr rule*: *ct-prop-coinduct-weak*)
  case (*ct-prop s vs es s′ vs′ es′ tr*)
  show *?case*
    using *ct-prop(3)*
  proof (*cases rule*: *config-is-trace.cases*)
    case *base*
    thus *?thesis*
      using *ct-prop(1,2)*
    by (*metis config-indistinguishable-imp-config-typing config-indistinguishable-imp-reduce*
config-indistinguishable-symm)
    next
    case (*step a s′ vs′ es′ tr*)

    thus *?thesis*
      using *ct-prop(1,2)*
    by (*simp del*: *config-indistinguishable.simps*) (*meson config-indistinguishable-imp-reduce*
preservation)
  qed
qed

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')
$i\ tr'$
  using *config-indistinguishable-imp-reduce2 assms*
  unfolding *ct-prop-def*
  by *auto*

lemma *program-actions-set2-indistinguishable-secrets-co*:
  assumes $tr \in (config\text{-}trace\text{-}set\ (s,\ vs,\ es)\ i)$
      $(s,vs,es) \sim\text{-}c\ (s',vs',es')$
      $\vdash\text{-}i\ s;vs;es : (Untrusted,ts)$
  shows $\exists\ tr' \in (config\text{-}trace\text{-}set\ (s',\ vs',\ es')\ i).$ *llist-all2 action-indistinguishable*
tr tr'
proof $-$
  have *config-is-trace* $(s,vs,es)\ i\ tr$
    using *assms(1)*
    unfolding *config-trace-set-def*
    by *fastforce*
  then obtain tr' **where** *config-is-trace* $(s',vs',es')\ i\ tr' \wedge$ *llist-all2 action-indistinguishable*
tr tr'
    using *config-indistinguishable-imp-reduce3*[*OF assms(2,3)*]
    by *fastforce*
  thus *?thesis*
    unfolding *config-trace-set-def*
    by *fastforce*
qed

lemma *program-actions2-indistinguishable-secrets-abs-set-co*:
  assumes $t \in (image\ abs\text{-}observation\ (config\text{-}trace\text{-}set\ (s,\ vs,\ es)\ i))$
      $(s,vs,es) \sim\text{-}c\ (s',vs',es')$
      $\vdash\text{-}i\ s;vs;es : (Untrusted,ts)$
  shows $t \in (image\ abs\text{-}observation\ (config\text{-}trace\text{-}set\ (s',\ vs',\ es')\ i))$
proof $-$
  obtain tr **where** *config-is-trace* $(s,vs,es)\ i\ tr \wedge (abs\text{-}observation\ tr) = t$
    using *assms(1)*
    unfolding *config-trace-set-def*
    by *fastforce*
  then obtain tr' **where** *config-is-trace* $(s',vs',es')\ i\ tr' \wedge (abs\text{-}observation\ tr') =$
t

    using *Quotient3-observation config-indistinguishable-imp-reduce3*[*OF assms*(*2,3*)]
     unfolding *Quotient3-def*
     by *metis*
   thus *?thesis*
     unfolding *config-trace-set-def*
     by *fastforce*
qed

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* $(s',\ vs',\ es')\ i$))
  using *program-actions2-indistinguishable-secrets-abs-set-co*[*OF - assms*(*1,2*)]
   config-indistinguishable-imp-config-typing[*OF assms*(*2,1*)]
    program-actions2-indistinguishable-secrets-abs-set-co[*OF - config-indistinguishable-symm*[*OF*
assms(*1*)]]
  by *fastforce*

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

lift-definition *config-untrusted-quot-obs-set* :: *config-untrusted-quot* $\Rightarrow$ *observation set* **is** $(\lambda c.\ config\text{-}obs\text{-}set\ c)$
proof −
 fix *prod1 prod2*
 assume *assms*:*prod1* $\sim\text{-}cp$ *prod2*
 show *config-obs-set prod1* = *config-obs-set prod2*
 proof (*cases prod1*; *cases prod2*)
  fix *c1 i1 c2 i2*
  assume *prod-assms*:*prod1* = (*c1,i1*) *prod2* = (*c2,i2*)
  show *?thesis*
  proof (*cases c1*; *cases c2*)
   fix $s\ vs\ es\ s'\ vs'\ es'$
   assume *config-assms*:*c1* = (*s,vs,es*) *c2* = (s',vs',es')
   obtain *ts* **where** *ts-def*:$(s,vs,es) \sim\text{-}c\ (s',vs',es')$
               $\vdash\text{-}i1\ s;vs;es : (Untrusted,ts)$
               i1 = *i2*
    using *assms prod-assms config-assms*
    unfolding *config-untrusted-equiv-def*
    by *fastforce*
   thus *?thesis*
  using *assms prod-assms config-assms program-actions2-indistinguishable-secrets-abs-set-equiv-co*
    unfolding *config-obs-set-def*
    by *simp*
  qed
 qed
qed

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

theorem *config-untrusted-constant-time*:
  assumes ⊢-i *s;vs;es* : (*Untrusted,ts*)
  shows *constant-time* ((s,vs,es),i)
  using *program-actions2-indistinguishable-secrets-abs-set-equiv-co assms*
  unfolding *constant-time-def config-obs-set-def*
  by *simp*

lift-definition *config-untrusted-quot-constant-time* :: *config-untrusted-quot* ⇒ *bool*
is *constant-time*
proof −
  fix *prod1 prod2*
  assume *assms:prod1* ∼-cp *prod2*
  show *constant-time prod1* = *constant-time prod2*
  proof (*cases prod1*; *cases prod2*)
    fix *c i c' i'*
    assume *local-assms:prod1* = (*c,i*) *prod2* = (*c',i'*)
    show *?thesis*
    proof (*cases c*; *cases c'*)
      fix *s vs es s' vs' es'*
      assume *inner-assms:c* = (*s,vs,es*) *c'* = (*s',vs',es'*)
      thus *?thesis*
        using *assms local-assms config-untrusted-constant-time*
        unfolding *config-untrusted-equiv-def*
      by (*simp del*: *config-indistinguishable.simps*) (*metis config-indistinguishable-imp-config-typing*)
    qed
  qed
qed

lemma *config-untrusted-quot-constant-time-trivial*:
  config-untrusted-quot-constant-time = (λx. *True*)
  using *config-untrusted-constant-time rep-config-untrusted-quot-typing*
  unfolding *config-untrusted-quot-constant-time.rep-eq*
  by (*metis prod.exhaust*)

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 ⊢-i *s;vs;es* : (*Untrusted,ts*)
  shows *constant-time-traces* ((s,vs,es),i)
  using *assms*
  unfolding *trace-set-equiv-def rel-set-def constant-time-traces-def*

apply (*simp del*: *config-indistinguishable.simps*)
  apply (*metis observation.abs-eq-iff config-indistinguishable-imp-config-typing config-indistinguishable-symm program-actions-set2-indistinguishable-secrets-co*)
  done
end

14 Constant Time (inductive)

theory *Wasm-Constant-Time-Ind* **imports** *Wasm-Secret* **begin**

definition *config-actions* :: [*s, v list, e list, nat, action list*] $\Rightarrow$ *bool* (*p'-actions* $(\!|\text{-};\text{-};\text{-}|\!)$ - - *60*) **where**
  config-actions s vs es i as $\equiv$ ($\exists\, s'\; vs'\; es'.\; (\!|s;vs;es|\!)\; as\,\hat{}\,\hat{}\leadsto\text{-}i\; (\!|s';vs';es'|\!)$)

definition *config-trace-set-ind* :: [($s \times v\; list \times e\; list$), *nat*] $\Rightarrow$ (*action list*) *set* **where**
  config-trace-set-ind $\equiv \lambda(s,vs,es)\; i.\; Collect\; (config\text{-}actions\; s\; vs\; es\; i)$

lemma *config-actions-indistinguishable-secrets-ind*:
  assumes (*p-actions* $(\!|s;vs;es|\!)\; i\; as$)
        (s,vs,es) $\sim$-c (s',vs',es')
        $\vdash$-i $s;vs;es$: (*Untrusted,ts*)
  shows $\exists\, as'.$ (*p-actions* $(\!|s';vs';es'|\!)\; i\; as'$) $\land$ *list-all2 action-indistinguishable as as'*
  using *assms(1,3) config-indistinguishable-trace-noninterference*[*OF - assms(2)*]
        config-indistinguishable-imp-config-typing[*OF - assms(2)*]
  unfolding *config-actions-def*
  by *blast*

lemma *config-actions-indistinguishable-secrets-abs-ind*:
  assumes (*p-actions* $(\!|s;vs;es|\!)\; i\; as$)
        (s,vs,es) $\sim$-c (s',vs',es')
        $\vdash$-i $s;vs;es$: (*Untrusted,ts*)
  shows $\exists\, as'.$ (*p-actions* $(\!|s';vs';es'|\!)\; i\; as'$) $\land$ ($\$\mathcal{A}\; as$) = ($\$\mathcal{A}\; as'$)
  using *Quotient3-observation config-actions-indistinguishable-secrets-ind*[*OF assms*]
  unfolding *Quotient3-def*
  by *metis*

lemma *config-trace-set-ind-indistinguishable-secrets-ind*:
  assumes *as* $\in$ (*config-trace-set-ind* (s,vs,es) i)
        (s,vs,es) $\sim$-c (s',vs',es')
        $\vdash$-i $s;vs;es$: (*Untrusted,ts*)
  shows $\exists\, as' \in$ (*config-trace-set-ind* (s',vs',es') i). *list-all2 action-indistinguishable as as'*
proof −
  have (*p-actions* $(\!|s;vs;es|\!)\; i\; as$)
    using *assms(1)*
    unfolding *config-trace-set-ind-def*
    by *fastforce*

then obtain *as′* **where** (*p-actions* (|*s′*;*vs′*;*es′*|) *i as′*) ∧ *list-all2 action-indistinguishable as as′*
  using *config-actions-indistinguishable-secrets-ind*[*OF - assms*(*2*,*3*)]
  by *fastforce*
thus *?thesis*
  unfolding *config-trace-set-ind-def*
  by *fastforce*
qed

lemma *config-actions-indistinguishable-secrets-abs-set-ind*:
  assumes $t \in$ (*image abs-obs* (*config-trace-set-ind* (*s*,*vs*,*es*) *i*))
      (*s*,*vs*,*es*) ∼-*c* (*s′*,*vs′*,*es′*)
      ⊢-*i s*;*vs*;*es* : (*Untrusted*,*ts*)
  shows $t \in$ (*image abs-obs* (*config-trace-set-ind* (*s′*,*vs′*,*es′*) *i*))
proof −
  obtain *as* **where** (*p-actions* (|*s*;*vs*;*es*|) *i as*) ∧ ($A *as*) = *t*
    using *assms*(*1*)
    unfolding *config-trace-set-ind-def*
    by *fastforce*
  then obtain *as′* **where** (*p-actions* (|*s′*;*vs′*;*es′*|) *i as′*) ∧ ($A *as′*) = *t*
    using *config-actions-indistinguishable-secrets-abs-ind*[*OF - assms*(*2*)] *assms*(*3*)
    by *fastforce*
  thus *?thesis*
    unfolding *config-trace-set-ind-def*
    by *fastforce*
qed

lemma *config-actions-indistinguishable-secrets-abs-set-equiv-ind*:
  assumes (*s*,*vs*,*es*) ∼-*c* (*s′*,*vs′*,*es′*)
      ⊢-*i s*;*vs*;*es* : (*Untrusted*,*ts*)
  shows (*image abs-obs* (*config-trace-set-ind* (*s*,*vs*,*es*) *i*)) = (*image abs-obs* (*config-trace-set-ind* (*s′*,*vs′*,*es′*) *i*))
  using *config-actions-indistinguishable-secrets-abs-set-ind*[*OF - assms*(*1*,*2*)]
    config-indistinguishable-imp-config-typing[*OF assms*(*2*,*1*)]
   config-actions-indistinguishable-secrets-abs-set-ind[*OF - config-indistinguishable-symm*[*OF assms*(*1*)]]
  by *fastforce*

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

lift-definition *config-untrusted-quot-obs-set-ind* :: *config-untrusted-quot* ⇒ *observation set* **is** ($\lambda c.$ *config-obs-set-ind c*)
proof −
  fix *prod1 prod2*
  assume *assms*:*prod1* ∼-*cp prod2*
  show *config-obs-set-ind prod1* = *config-obs-set-ind prod2*
  proof (*cases prod1*; *cases prod2*)
    fix *c1 i1 c2 i2*

```
    assume prod-assms:prod1 = (c1,i1) prod2 = (c2,i2)
    show ?thesis
    proof (cases c1; cases c2)
      fix s vs es s′ vs′ es′
      assume config-assms:c1 = (s,vs,es) c2 = (s′,vs′,es′)
      obtain ts where ts-def:(s,vs,es) ∼-c (s′,vs′,es′)
                        ⊢-i1 s;vs;es : (Untrusted,ts)
                        i1 = i2
      using assms prod-assms config-assms
      unfolding config-untrusted-equiv-def
      by fastforce
      thus ?thesis
      using assms prod-assms config-assms config-actions-indistinguishable-secrets-abs-set-equiv-ind
      unfolding config-obs-set-ind-def
      by simp
    qed
  qed
qed
```

```
definition constant-time-ind :: ((s × v list × e list) × nat) ⇒ bool where
  constant-time-ind = (λ(c, i). ∀ c′. (c ∼-c c′) ⟶ ((config-obs-set-ind (c,i)) =
(config-obs-set-ind (c′,i))))
```

```
theorem config-untrusted-constant-time-ind:
  assumes ⊢-i s;vs;es : (Untrusted,ts)
  shows constant-time-ind ((s,vs,es),i)
    using config-actions-indistinguishable-secrets-abs-set-equiv-ind assms
    unfolding constant-time-ind-def config-obs-set-ind-def
    by simp
```

```
lift-definition config-untrusted-quot-constant-time-ind :: config-untrusted-quot ⇒
bool is constant-time-ind
proof −
  fix prod1 prod2
  assume assms:prod1 ∼-cp prod2
  show constant-time-ind prod1 = constant-time-ind prod2
  proof (cases prod1; cases prod2)
    fix c i c′ i′
    assume local-assms:prod1 = (c,i) prod2 = (c′,i′)
    show ?thesis
    proof (cases c; cases c′)
      fix s vs es s′ vs′ es′
      assume inner-assms:c = (s,vs,es) c′ = (s′,vs′,es′)
      thus ?thesis
        using assms local-assms config-untrusted-constant-time-ind
        unfolding config-untrusted-equiv-def
      by (simp del: config-indistinguishable.simps) (metis config-indistinguishable-imp-config-typing)
    qed
  qed
```

286

qed

lemma *config-untrusted-quot-constant-time-trivial-ind*:
  config-untrusted-quot-constant-time-ind = ($\lambda x.$ *True*)
  using *config-untrusted-constant-time-ind rep-config-untrusted-quot-typing*
  unfolding *config-untrusted-quot-constant-time-ind.rep-eq*
  by (*metis prod.exhaust*)

definition *trace-set-equiv* = *rel-set* (*list-all2 action-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)
  using *assms*
  unfolding *trace-set-equiv-def rel-set-def constant-time-traces-ind-def*
  apply (*simp del*: *config-indistinguishable.simps*)
  apply (*metis config-indistinguishable-imp-config-typing config-indistinguishable-symm*
config-trace-set-ind-indistinguishable-secrets-ind observation.abs-eq-iff)
  done
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* $\Rightarrow$ *leakage* **where**
action-leakage a =
  (*case a of*
  Unop-i32-action op′ $\Rightarrow$ *Arith-leakage* (*Unop-i32-leakage op′*)
| *Unop-i64-action op′* $\Rightarrow$ *Arith-leakage* (*Unop-i64-leakage op′*)
| *Unop-f32-action op′ c* $\Rightarrow$ *Arith-leakage* (*Unop-f32-leakage op′ c*)
| *Unop-f64-action op′ c* $\Rightarrow$ *Arith-leakage* (*Unop-f64-leakage op′ c*)
| *Binop-i32-Some-action op′ c1 c2* $\Rightarrow$ *Arith-leakage* (*if* (*safe-binop-i op′*)
                                 then Binop-i32-Some-safe-leakage op′
                                 else Binop-i32-Some-leakage op′ c1 c2)
| *Binop-i32-None-action op′ c1 c2* $\Rightarrow$ *Arith-leakage* (*if* (*safe-binop-i op′*)

288

$\qquad$ then *Binop-i32-None-safe-leakage op'*

$\qquad$ else *Binop-i32-None-leakage op' c1 c2*)

| *Binop-i64-Some-action op' c1 c2* $\Rightarrow$ *Arith-leakage* (*if* (*safe-binop-i op'*)

$\qquad$ then *Binop-i64-Some-safe-leakage op'*

$\qquad$ else *Binop-i64-Some-leakage op' c1 c2*)

| *Binop-i64-None-action op' c1 c2* $\Rightarrow$*Arith-leakage* (*if* (*safe-binop-i op'*)

$\qquad$ then *Binop-i64-None-safe-leakage op'*

$\qquad$ else *Binop-i64-None-leakage op' c1 c2*)

| *Binop-f32-Some-action op' c1 c2* $\Rightarrow$ *Arith-leakage* (*Binop-f32-Some-leakage op'*
c1 c2)

| *Binop-f32-None-action op' c1 c2* $\Rightarrow$ *Arith-leakage* (*Binop-f32-None-leakage op'*
c1 c2)

| *Binop-f64-Some-action op' c1 c2* $\Rightarrow$ *Arith-leakage* (*Binop-f64-Some-leakage op'*
c1 c2)

| *Binop-f64-None-action op' c1 c2* $\Rightarrow$ *Arith-leakage* (*Binop-f64-None-leakage op'*
c1 c2)

| *Testop-i32-action op'* $\Rightarrow$ *Arith-leakage* (*Testop-i32-leakage op'*)

| *Testop-i64-action op'* $\Rightarrow$ *Arith-leakage* (*Testop-i64-leakage op'*)

| *Relop-i32-action op'* $\Rightarrow$ *Arith-leakage* (*Relop-i32-leakage op'*)

| *Relop-i64-action op'* $\Rightarrow$ *Arith-leakage* (*Relop-i64-leakage op'*)

| *Relop-f32-action op' c1 c2* $\Rightarrow$ *Arith-leakage* (*Relop-f32-leakage op' c1 c2*)

| *Relop-f64-action op' c1 c2* $\Rightarrow$ *Arith-leakage* (*Relop-f64-leakage op' c1 c2*)

| *Convert-Some-action t1 t2 c* $\Rightarrow$ (*if is-int-t t1* $\wedge$ *is-int-t t2*

$\qquad$ then *Convert-Some-int-leakage t1 t2*

$\qquad$ else *Convert-Some-leakage t1 t2 c*)

| *Convert-None-action t1 t2 c* $\Rightarrow$ (*if is-int-t t1* $\wedge$ *is-int-t t2*

$\qquad$ then *Convert-None-int-leakage t1 t2*

$\qquad$ else *Convert-None-leakage t1 t2 c*)

| *Reinterpret-action* $\Rightarrow$ *Empty-leakage*

| *Classify-action* $\Rightarrow$ *Empty-leakage*

| *Declassify-action* $\Rightarrow$ *Empty-leakage*

| *Unreachable-action* $\Rightarrow$ *Empty-leakage*

| *Nop-action* $\Rightarrow$ *Empty-leakage*

| *Drop-action* $\Rightarrow$ *Empty-leakage*

| *Select-action sec c* $\Rightarrow$ *if* (*sec = Secret*) *then Select-leakage None else Select-leakage*
(*Some c*)

| *Block-action* $\Rightarrow$ *Empty-leakage*

| *Loop-action* $\Rightarrow$ *Empty-leakage*

| *If-false-action c* $\Rightarrow$ *If-false-leakage c*

| *If-true-action c* $\Rightarrow$ *If-true-leakage c*

| *Label-const-action* $\Rightarrow$ *Empty-leakage*

| *Label-trap-action* $\Rightarrow$ *Empty-leakage*

| *Br-action* $\Rightarrow$ *Empty-leakage*

| *Br-if-false-action c* $\Rightarrow$ *Br-if-false-leakage c*

| *Br-if-true-action c* $\Rightarrow$ *Br-if-true-leakage c*

| *Br-table-action c* $\Rightarrow$ *Br-table-leakage c*

| *Br-table-length-action c* $\Rightarrow$ *Br-table-length-leakage c*

| *Local-const-action* $\Rightarrow$ *Empty-leakage*

| *Local-trap-action* $\Rightarrow$ *Empty-leakage*

| *Return-action* ⇒ *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′*
  using *assms*
  unfolding *memory-public-agree-def*
  by (*cases m*; *cases m′*) *auto*

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′*
  using *assms*
proof (*induction ms arbitrary*: *ms′*)
  case *Nil*
  thus *?case*
    by *simp*
next
  case (*Cons a ms*)
  obtain *a′ ms″* **where** *ms′* = *a′#ms″*
                  memory-public-agree a a′
                  memories-public-agree ms ms″

    using *Cons(2)*
    by (*metis list-all2-Cons1*)
  thus *?case*
    using *Cons(1) memory-agree-filter*
    by (*fastforce simp add*: *memory-public-agree-def*)
qed

lemma *action-indistinguishable-imp-action-leakage-eq*:
  assumes $a \sim$-$a\ a'$
       action-leakage a = obs
  shows *action-leakage a' = obs*
  using *assms*
proof (*induction rule*: *action-indistinguishable.induct*)
  case (*host-Some s s' vcs vcs' s-o s'-o vcs-o vcs'-o tf f hs hs'*)
  have *filter* (λ(*m,sec*). *sec = Public*) (*mem s*) = *filter* (λ(*m,sec*). *sec = Public*) (*mem s'*)
    using *host-Some(1) store-public-agree-def memories-agree-filter*
    by *simp*
  thus *?case*
    using *host-Some(5)*
    by (*auto simp add*: *action-leakage-def*)
next
  case (*host-None s s' vcs vcs' tf f hs hs'*)
  have *filter* (λ(*m,sec*). *sec = Public*) (*mem s*) = *filter* (λ(*m,sec*). *sec = Public*) (*mem s'*)
    using *host-None(1) store-public-agree-def memories-agree-filter*
    by *simp*
  thus *?case*
    using *host-None(3)*
    by (*auto simp add*: *action-leakage-def*)
qed (*auto simp add*: *action-leakage-def*)

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.