WebAssembly

Conrad Watt

May 26, 2024

Abstract

This is a mechanised specification of the WebAssembly language, drawn mainly from the previously published paper formalisation [1]. Also included is a full proof of soundness of the type system, together with a verified type checker and interpreter. We include only a partial procedure for the extraction of the type checker and interpreter here. For more details, please see our paper [2].

Contents

1 WebAssembly Core AST **2**

2 Syntactic Typeclasses **5**

3 WebAssembly Base Definitions **7**

4 Host Properties **24**

5 Auxiliary Type System Properties **25**

6 Lemmas for Soundness Proof **54**
6.1 Preservation . . . 54
6.2 Progress . . . 92

7 Soundness Theorems **126**

8 Augmented Type Syntax for Concrete Checker **127**

9 Executable Type Checker **150**

10 Correctness of Type Checker **154**
10.1 Soundness . . . 154
10.2 Completeness . . . 172

11 WebAssembly Interpreter **189**

1 WebAssembly Core AST

theory *Wasm-Ast*
  imports
    Main
    Native-Word.Uint8
    Word-Lib.Reversed-Bit-Lists
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* ⇒ *nat* ⇒ *bytes* ⇒ *bytes* **is** (λ*a n as. takefill* (*Abs-uint8 a*) *n as*) **.**
lift-definition *bytes-replicate* :: *nat* ⇒ *byte* ⇒ *bytes* **is** (λ*n b. replicate n* (*Abs-uint8 b*)) **.**
definition *msbyte* :: *bytes* ⇒ *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* ⇒ *nat* ⇒ *nat* ⇒ *bytes* **is** (λ*m n l. take l* (*drop n m*)) **.**
lift-definition *write-bytes* :: *mem* ⇒ *nat* ⇒ *bytes* ⇒ *mem* **is** (λ*m n bs.* (*take n m*) @ *bs* @ (*drop* (*n* + *length bs*) *m*)) **.**
lift-definition *mem-append* :: *mem* ⇒ *bytes* ⇒ *mem* **is** *append* **.**

— host

typedecl *host*
typedecl *host-state*

datatype — value types
  $t = T\text{-}i32 \mid T\text{-}i64 \mid T\text{-}f32 \mid T\text{-}f64$

datatype — packed types
  $tp = Tp\text{-}i8 \mid Tp\text{-}i16 \mid Tp\text{-}i32$

datatype — mutability
  $mut = T\text{-}immut \mid T\text{-}mut$

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

datatype — function types
  $tf = Tf\ t\ list\ t\ list\ (\text{-}\ '\text{->}\ \text{-}\ 60)$

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

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

datatype
  $sx = S \mid U$

datatype
  $unop\text{-}i = Clz \mid Ctz \mid Popcnt$

datatype
  $unop\text{-}f = Neg \mid Abs \mid Ceil \mid Floor \mid Trunc \mid Nearest \mid Sqrt$

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

datatype
binop-f = *Addf* | *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*

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

datatype — basic instructions
b-e =
Unreachable
| *Nop*
| *Drop*
| *Select*
| *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 list b-e list
| *Func-host tf host*

record *inst* = — instances
types :: *tf 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 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*
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 ⇒ 'a
  fixes float-floor   :: 'a ⇒ 'a
  fixes float-trunc   :: 'a ⇒ 'a
  fixes float-nearest :: 'a ⇒ 'a
  fixes float-sqrt    :: 'a ⇒ 'a

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

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

3 WebAssembly Base Definitions

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

instantiation i32 :: wasm-int begin instance .. 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
```

$f32\text{-}convert\text{-}si32 :: i32 \Rightarrow f32$
$f32\text{-}convert\text{-}ui64 :: i64 \Rightarrow f32$
$f32\text{-}convert\text{-}si64 :: i64 \Rightarrow f32$

$f64\text{-}convert\text{-}ui32 :: i32 \Rightarrow f64$
$f64\text{-}convert\text{-}si32 :: i32 \Rightarrow f64$
$f64\text{-}convert\text{-}ui64 :: i64 \Rightarrow f64$
$f64\text{-}convert\text{-}si64 :: i64 \Rightarrow f64$

$wasm\text{-}wrap :: i64 \Rightarrow i32$
$wasm\text{-}extend\text{-}u :: i32 \Rightarrow i64$
$wasm\text{-}extend\text{-}s :: i32 \Rightarrow i64$
$wasm\text{-}demote :: f64 \Rightarrow f32$
$wasm\text{-}promote :: f32 \Rightarrow f64$

$serialise\text{-}i32 :: i32 \Rightarrow bytes$
$serialise\text{-}i64 :: i64 \Rightarrow bytes$
$serialise\text{-}f32 :: f32 \Rightarrow bytes$
$serialise\text{-}f64 :: f64 \Rightarrow bytes$
$wasm\text{-}bool :: bool \Rightarrow i32$
$int32\text{-}minus\text{-}one :: i32$

definition $mem\text{-}size :: mem \Rightarrow nat$ **where**
$mem\text{-}size\ m = length\ (Rep\text{-}mem\ m)$

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

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

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

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

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

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

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 - ⇒ *T-i32*
        | *ConstInt64* - ⇒ *T-i64*
        | *ConstFloat32* - ⇒ *T-f32*
        | *ConstFloat64* - ⇒ *T-f64*)

definition *option-projl* :: (′*a* × ′*b*) *option* ⇒ ′*a option* **where**
  option-projl x = *map-option fst x*

definition *option-projr* :: (′*a* × ′*b*) *option* ⇒ ′*b option* **where**
  option-projr x = *map-option snd x*

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

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

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

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

definition *is-mut* :: *tg* ⇒ *bool* **where**

is-mut tg = (*tg-mut tg* = *T-mut*)

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

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

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

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

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

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

definition *app-relop-f* :: *relop-f* ⇒ *'f::wasm-float* ⇒ *'f::wasm-float* ⇒ *bool* **where**
app-relop-f rop c1 c2 = (*case rop of*
Eqf ⇒ *float-eq c1 c2*
| *Nef* ⇒ *float-ne c1 c2*
| *Ltf* ⇒ *float-lt c1 c2*
| *Gtf* ⇒ *float-gt c1 c2*
| *Lef* ⇒ *float-le c1 c2*
| *Gef* ⇒ *float-ge c1 c2*)

definition *types-agree* :: *t* ⇒ *v* ⇒ *bool* **where**
types-agree t v = (*typeof v* = *t*)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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 ⇒ (*case* (*cvt-i32 sx v*) *of Some c* ⇒ *Some* (*ConstInt32 c*) | *None* ⇒ *None*)
| *T-i64* ⇒ (*case* (*cvt-i64 sx v*) *of Some c* ⇒ *Some* (*ConstInt64 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 ⇒ *ConstInt32 0*
| *T-i64* ⇒ *ConstInt64 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 $t = T\text{-}i32 \lor t = T\text{-}i64$
using *assms*
by (*cases t*) (*auto simp add*: *is-int-t-def*)

lemma *is-float-t-exists*:
assumes *is-float-t t*
shows $t = T\text{-}f32 \lor t = T\text{-}f64$
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*(*13*−*16*))

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* (λ*v*. $*C v*)
by (*meson b-e.inject*(*16*) *e.inject*(*1*) *injI*)

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*:$\exists$ *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 $\exists$ *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 $\exists$ *lholed′ ves es-c*. *const-list ves* $\wedge$ *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* $\Longrightarrow$ (*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*] ⇒ *bool* (- ⊢ - : - *60*) **where**
— *num ops*
const:$\mathcal{C}$ ⊢ [*C v*] : ([] -> [(*typeof v*)])
| *unop-i*:*is-int-t t* $\Longrightarrow$ $\mathcal{C}$ ⊢ [*Unop-i t* -] : ([*t*] -> [*t*])
| *unop-f*:*is-float-t t* $\Longrightarrow$ $\mathcal{C}$ ⊢ [*Unop-f t* -] : ([*t*] -> [*t*])
| *binop-i*:*is-int-t t* $\Longrightarrow$ $\mathcal{C}$ ⊢ [*Binop-i t iop*] : ([*t*,*t*] -> [*t*])
| *binop-f*:*is-float-t t* $\Longrightarrow$ $\mathcal{C}$ ⊢ [*Binop-f t* -] : ([*t*,*t*] -> [*t*])
| *testop*:*is-int-t t* $\Longrightarrow$ $\mathcal{C}$ ⊢ [*Testop t* -] : ([*t*] -> [*T-i32*])
| *relop-i*:*is-int-t t* $\Longrightarrow$ $\mathcal{C}$ ⊢ [*Relop-i t* -] : ([*t*,*t*] -> [*T-i32*])
| *relop-f*:*is-float-t t* $\Longrightarrow$ $\mathcal{C}$ ⊢ [*Relop-f t* -] : ([*t*,*t*] -> [*T-i32*])
— *convert*
| *convert*:⟦(*t1* ≠ *t2*); (*sx* = *None*) = ((*is-float-t t1* ∧ *is-float-t t2*) ∨ (*is-int-t t1* ∧ *is-int-t t2* ∧ (*t-length t1* < *t-length t2*)))⟧ $\Longrightarrow$ $\mathcal{C}$ ⊢ [*Cvtop t1 Convert t2 sx*] : ([*t2*] -> [*t1*])
— *reinterpret*
| *reinterpret*:⟦(*t1* ≠ *t2*); *t-length t1* = *t-length t2*⟧ $\Longrightarrow$ $\mathcal{C}$ ⊢ [*Cvtop t1 Reinterpret t2 None*] : ([*t2*] -> [*t1*])
— *unreachable*, *nop*, *drop*, *select*
| *unreachable*:$\mathcal{C}$ ⊢ [*Unreachable*] : (*ts* -> *ts'*)
| *nop*:$\mathcal{C}$ ⊢ [*Nop*] : ([] -> [])

| *drop*:$\mathcal{C} \vdash$ [*Drop*] : ([*t*] -> [])
| *select*:$\mathcal{C} \vdash$ [*Select*] : ([*t*,*t*,*T-i32*] -> [*t*])
— *block*
| *block*:$[\![$*tf* = (*tn* -> *tm*); $\mathcal{C}$(|*label* := ([*tm*] @ (*label* $\mathcal{C}$))|) $\vdash$ *es* : (*tn* -> *tm*)$]\!] \Longrightarrow \mathcal{C} \vdash$ [*Block tf es*] : (*tn* -> *tm*)
— *loop*
| *loop*:$[\![$*tf* = (*tn* -> *tm*); $\mathcal{C}$(|*label* := ([*tn*] @ (*label* $\mathcal{C}$))|) $\vdash$ *es* : (*tn* -> *tm*)$]\!] \Longrightarrow \mathcal{C} \vdash$ [*Loop tf es*] : (*tn* -> *tm*)
— *if then else*
| *if-wasm*:$[\![$*tf* = (*tn* -> *tm*); $\mathcal{C}$(|*label* := ([*tm*] @ (*label* $\mathcal{C}$))|) $\vdash$ *es1* : (*tn* -> *tm*); $\mathcal{C}$(|*label* := ([*tm*] @ (*label* $\mathcal{C}$))|) $\vdash$ *es2* : (*tn* -> *tm*)$]\!] \Longrightarrow \mathcal{C} \vdash$ [*If tf es1 es2*] : (*tn* @ [*T-i32*] -> *tm*)
— *br*
| *br*:$[\![$*i* < *length*(*label* $\mathcal{C}$); (*label* $\mathcal{C}$)!*i* = *ts*$]\!] \Longrightarrow \mathcal{C} \vdash$ [*Br i*] : (*t1s* @ *ts* -> *t2s*)
— *br-if*
| *br-if*:$[\![$*i* < *length*(*label* $\mathcal{C}$); (*label* $\mathcal{C}$)!*i* = *ts*$]\!] \Longrightarrow \mathcal{C} \vdash$ [*Br-if i*] : (*ts* @ [*T-i32*] -> *ts*)
— *br-table*
| *br-table*:$[\![$*list-all* (λ*i*. *i* < *length*(*label* $\mathcal{C}$) $\wedge$ (*label* $\mathcal{C}$)!*i* = *ts*) (*is*@[*i*])$]\!] \Longrightarrow \mathcal{C} \vdash$ [*Br-table is i*] : (*t1s* @ *ts* @ [*T-i32*] -> *t2s*)
— *return*
| *return*:$[\![$(*return* $\mathcal{C}$) = *Some ts*$]\!] \Longrightarrow \mathcal{C} \vdash$ [*Return*] : (*t1s* @ *ts* -> *t2s*)
— *call*
| *call*:$[\![$*i* < *length*(*func-t* $\mathcal{C}$); (*func-t* $\mathcal{C}$)!*i* = *tf*$]\!] \Longrightarrow \mathcal{C} \vdash$ [*Call i*] : *tf*
— *call-indirect*
| *call-indirect*:$[\![$*i* < *length*(*types-t* $\mathcal{C}$); (*types-t* $\mathcal{C}$)!*i* = (*t1s* -> *t2s*); (*table* $\mathcal{C}$) $\neq$ *None*$]\!] \Longrightarrow \mathcal{C} \vdash$ [*Call-indirect i*] : (*t1s* @ [*T-i32*] -> *t2s*)
— *get-local*
| *get-local*:$[\![$*i* < *length*(*local* $\mathcal{C}$); (*local* $\mathcal{C}$)!*i* = *t*$]\!] \Longrightarrow \mathcal{C} \vdash$ [*Get-local i*] : ([] -> [*t*])
— *set-local*
| *set-local*:$[\![$*i* < *length*(*local* $\mathcal{C}$); (*local* $\mathcal{C}$)!*i* = *t*$]\!] \Longrightarrow \mathcal{C} \vdash$ [*Set-local i*] : ([*t*] -> [])
— *tee-local*
| *tee-local*:$[\![$*i* < *length*(*local* $\mathcal{C}$); (*local* $\mathcal{C}$)!*i* = *t*$]\!] \Longrightarrow \mathcal{C} \vdash$ [*Tee-local i*] : ([*t*] -> [*t*])
— *get-global*
| *get-global*:$[\![$*i* < *length*(*global* $\mathcal{C}$); *tg-t* ((*global* $\mathcal{C}$)!*i*) = *t*$]\!] \Longrightarrow \mathcal{C} \vdash$ [*Get-global i*] : ([] -> [*t*])
— *set-global*
| *set-global*:$[\![$*i* < *length*(*global* $\mathcal{C}$); *tg-t* ((*global* $\mathcal{C}$)!*i*) = *t*; *is-mut* ((*global* $\mathcal{C}$)!*i*)$]\!] \Longrightarrow \mathcal{C} \vdash$ [*Set-global i*] : ([*t*] -> [])
— *load*
| *load*:$[\![$(*memory* $\mathcal{C}$) = *Some n*; *load-store-t-bounds a* (*option-projl tp-sx*) *t*$]\!] \Longrightarrow \mathcal{C} \vdash$ [*Load t tp-sx a off*] : ([*T-i32*] -> [*t*])
— *store*
| *store*:$[\![$(*memory* $\mathcal{C}$) = *Some n*; *load-store-t-bounds a tp t*$]\!] \Longrightarrow \mathcal{C} \vdash$ [*Store t tp a off*] : ([*T-i32*,*t*] -> [])
— *current-memory*
| *current-memory*:(*memory* $\mathcal{C}$) = *Some n* $\Longrightarrow \mathcal{C} \vdash$ [*Current-memory*] : ([] -> [*T-i32*])
— *Grow-memory*
| *grow-memory*:(*memory* $\mathcal{C}$) = *Some n* $\Longrightarrow \mathcal{C} \vdash$ [*Grow-memory*] : ([*T-i32*] -> [*T-i32*])

— *empty program*
| *empty*:𝒞 ⊢ [] : ([] -> [])
— *composition*
| *composition*:⟦𝒞 ⊢ *es* : (*t1s* -> *t2s*); 𝒞 ⊢ [*e*] : (*t2s* -> *t3s*)⟧ ⟹ 𝒞 ⊢ *es* @ [*e*] : (*t1s* -> *t3s*)
— *weakening*
| *weakening*:𝒞 ⊢ *es* : (*t1s* -> *t2s*) ⟹ 𝒞 ⊢ *es* : (*ts* @ *t1s* -> *ts* @ *t2s*)

inductive *cl-typing* :: [*s-context*, *cl*, *tf*] ⇒ *bool* **where**
⟦*i* < *length* (*s-inst* 𝒮); ((*s-inst* 𝒮)!*i*) = 𝒞; *tf* = (*t1s* -> *t2s*); 𝒞⦇*local* := (*local* 𝒞) @ *t1s* @ *ts*, *label* := ([*t2s*] @ (*label* 𝒞)), *return* := *Some t2s*⦈ ⊢ *es* : ([] -> *t2s*)⟧ ⟹ *cl-typing* 𝒮 (*Func-native i tf ts es*) (*t1s* -> *t2s*)
| *cl-typing* 𝒮 (*Func-host tf h*) *tf*

inductive *e-typing* :: [*s-context*, *t-context*, *e list*, *tf*] ⇒ *bool* (-•- ⊢ - : - *60*)
and *s-typing* :: [*s-context*, (*t list*) *option*, *nat*, *v list*, *e list*, *t list*] ⇒ *bool* (-•- ⊩'- - -;- : - *60*) **where**

𝒞 ⊢ *b-es* : *tf* ⟹ 𝒮•𝒞 ⊢ $∗*b-es* : *tf*

| ⟦𝒮•𝒞 ⊢ *es* : (*t1s* -> *t2s*); 𝒮•𝒞 ⊢ [*e*] : (*t2s* -> *t3s*)⟧ ⟹ 𝒮•𝒞 ⊢ *es* @ [*e*] : (*t1s* -> *t3s*)

| 𝒮•𝒞 ⊢ *es* : (*t1s* -> *t2s*) ⟹𝒮•𝒞 ⊢ *es* : (*ts* @ *t1s* -> *ts* @ *t2s*)

| 𝒮•𝒞 ⊢ [*Trap*] : *tf*

| ⟦𝒮•*Some ts* ⊩-*i vs*;*es* : *ts*; *length ts* = *n*⟧ ⟹ 𝒮•𝒞 ⊢ [*Local n i vs es*] : ([] -> *ts*)

| ⟦*cl-typing* 𝒮 *cl tf*⟧ ⟹ 𝒮•𝒞 ⊢ [*Callcl cl*] : *tf*

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

| ⟦*i* < (*length* (*s-inst* 𝒮)); *tvs* = *map typeof vs*; 𝒞 =((*s-inst* 𝒮)!*i*)⦇*local* := (*local* ((*s-inst* 𝒮)!*i*) @ *tvs*), *return* := *rs*⦈; 𝒮•𝒞 ⊢ *es* : ([] -> *ts*); (*rs* = *Some ts*) ∨ *rs* = *None*⟧ ⟹ 𝒮•*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*;

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

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

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

definition $mem\text{-}agree\ bs\ m = (\lambda\ bs\ m.\ m \leq mem\text{-}size\ bs)\ bs\ m$

inductive $store\text{-}typing :: [s, s\text{-}context] \Rightarrow bool$ **where**
$\llbracket\mathcal{S} = (\!|s\text{-}inst = \mathcal{C}s, s\text{-}funcs = tfs, s\text{-}tab = ns, s\text{-}mem = ms, s\text{-}globs = tgs|\!)$; $list\text{-}all2\ (inst\text{-}typing\ \mathcal{S})\ insts\ \mathcal{C}s$; $list\text{-}all2\ (cl\text{-}typing\ \mathcal{S})\ fs\ tfs$; $list\text{-}all\ (tab\text{-}agree\ \mathcal{S})\ (concat\ tclss)$; $list\text{-}all2\ (\lambda\ tcls\ n.\ n \leq length\ tcls)\ tclss\ ns$; $list\text{-}all2\ mem\text{-}agree\ bss\ ms$; $list\text{-}all2\ glob\text{-}agree\ gs\ tgs\rrbracket \Longrightarrow store\text{-}typing\ (\!|s.inst = insts, s.funcs = fs, s.tab = tclss, s.mem = bss, s.globs = gs|\!)\ \mathcal{S}$

inductive $config\text{-}typing :: [nat, s, v\ list, e\ list, t\ list] \Rightarrow bool$ ($\vdash'$- - -;-;- : - *60*) **where**
$\llbracket store\text{-}typing\ s\ \mathcal{S}; \mathcal{S}\cdot None \Vdash\text{-}i\ vs;es : ts\rrbracket \Longrightarrow \vdash\text{-}i\ s;vs;es : ts$

inductive $reduce\text{-}simple :: [e\ list, e\ list] \Rightarrow bool$ ((\|-\|) ⇝ (\|-\|) *60*) **where**
— *integer unary ops*
$unop\text{-}i32$:(\|[\$$C$ ($ConstInt32\ c$), \$($Unop\text{-}i\ T\text{-}i32\ iop$)]\|) ⇝ (\|[\$$C$ ($ConstInt32$ ($app\text{-}unop\text{-}i\ iop\ c$))]\|)
| $unop\text{-}i64$:(\|[\$$C$ ($ConstInt64\ c$), \$($Unop\text{-}i\ T\text{-}i64\ iop$)]\|) ⇝ (\|[\$$C$ ($ConstInt64$ ($app\text{-}unop\text{-}i\ iop\ c$))]\|)
— *float unary ops*
| $unop\text{-}f32$:(\|[\$$C$ ($ConstFloat32\ c$), \$($Unop\text{-}f\ T\text{-}f32\ fop$)]\|) ⇝ (\|[\$$C$ ($ConstFloat32$ ($app\text{-}unop\text{-}f\ fop\ c$))]\|)
| $unop\text{-}f64$:(\|[\$$C$ ($ConstFloat64\ c$), \$($Unop\text{-}f\ T\text{-}f64\ fop$)]\|) ⇝ (\|[\$$C$ ($ConstFloat64$ ($app\text{-}unop\text{-}f\ fop\ c$))]\|)
— *int32 binary ops*
| $binop\text{-}i32\text{-}Some$:$\llbracket app\text{-}binop\text{-}i\ iop\ c1\ c2 = (Some\ c)\rrbracket \Longrightarrow$ (\|[\$$C$ ($ConstInt32\ c1$), \$$C$ ($ConstInt32\ c2$), \$($Binop\text{-}i\ T\text{-}i32\ iop$)]\|) ⇝ (\|[\$$C$ ($ConstInt32\ c$)]\|)
| $binop\text{-}i32\text{-}None$:$\llbracket app\text{-}binop\text{-}i\ iop\ c1\ c2 = None\rrbracket \Longrightarrow$ (\|[\$$C$ ($ConstInt32\ c1$), \$$C$ ($ConstInt32\ c2$), \$($Binop\text{-}i\ T\text{-}i32\ iop$)]\|) ⇝ (\|[$Trap$]\|)
— *int64 binary ops*
| $binop\text{-}i64\text{-}Some$:$\llbracket app\text{-}binop\text{-}i\ iop\ c1\ c2 = (Some\ c)\rrbracket \Longrightarrow$ (\|[\$$C$ ($ConstInt64\ c1$), \$$C$ ($ConstInt64\ c2$), \$($Binop\text{-}i\ T\text{-}i64\ iop$)]\|) ⇝ (\|[\$$C$ ($ConstInt64\ c$)]\|)
| $binop\text{-}i64\text{-}None$:$\llbracket app\text{-}binop\text{-}i\ iop\ c1\ c2 = None\rrbracket \Longrightarrow$ (\|[\$$C$ ($ConstInt64\ c1$), \$$C$ ($ConstInt64\ c2$), \$($Binop\text{-}i\ T\text{-}i64\ iop$)]\|) ⇝ (\|[$Trap$]\|)
— *float32 binary ops*
| $binop\text{-}f32\text{-}Some$:$\llbracket app\text{-}binop\text{-}f\ fop\ c1\ c2 = (Some\ c)\rrbracket \Longrightarrow$ (\|[\$$C$ ($ConstFloat32\ c1$), \$$C$ ($ConstFloat32\ c2$), \$($Binop\text{-}f\ T\text{-}f32\ fop$)]\|) ⇝ (\|[\$$C$ ($ConstFloat32\ c$)]\|)

| *binop-f32-None*:⟦*app-binop-f fop c1 c2* = *None*⟧ $\Longrightarrow$ ⦇[$*C* (*ConstFloat32 c1*), $*C* (*ConstFloat32 c2*), $(*Binop-f T-f32 fop*)]⦈ $\leadsto$ ⦇[*Trap*]⦈

— *float64 binary ops*

| *binop-f64-Some*:⟦*app-binop-f fop c1 c2* = (*Some c*)⟧ $\Longrightarrow$ ⦇[$*C* (*ConstFloat64 c1*), $*C* (*ConstFloat64 c2*), $(*Binop-f T-f64 fop*)]⦈ $\leadsto$ ⦇[$*C* (*ConstFloat64 c*)]⦈

| *binop-f64-None*:⟦*app-binop-f fop c1 c2* = *None*⟧ $\Longrightarrow$ ⦇[$*C* (*ConstFloat64 c1*), $*C* (*ConstFloat64 c2*), $(*Binop-f T-f64 fop*)]⦈ $\leadsto$ ⦇[*Trap*]⦈

— *testops*

| *testop-i32*:⦇[$*C* (*ConstInt32 c*), $(*Testop T-i32 testop*)]⦈ $\leadsto$ ⦇[$*C* *ConstInt32* (*wasm-bool* (*app-testop-i testop c*))]⦈

| *testop-i64*:⦇[$*C* (*ConstInt64 c*), $(*Testop T-i64 testop*)]⦈ $\leadsto$ ⦇[$*C* *ConstInt32* (*wasm-bool* (*app-testop-i testop c*))]⦈

— *int relops*

| *relop-i32*:⦇[$*C* (*ConstInt32 c1*), $*C* (*ConstInt32 c2*), $(*Relop-i T-i32 iop*)]⦈ $\leadsto$ ⦇[$*C* (*ConstInt32* (*wasm-bool* (*app-relop-i iop c1 c2*)))]⦈

| *relop-i64*:⦇[$*C* (*ConstInt64 c1*), $*C* (*ConstInt64 c2*), $(*Relop-i T-i64 iop*)]⦈ $\leadsto$ ⦇[$*C* (*ConstInt32* (*wasm-bool* (*app-relop-i iop c1 c2*)))]⦈

— *float relops*

| *relop-f32*:⦇[$*C* (*ConstFloat32 c1*), $*C* (*ConstFloat32 c2*), $(*Relop-f T-f32 fop*)]⦈ $\leadsto$ ⦇[$*C* (*ConstInt32* (*wasm-bool* (*app-relop-f fop c1 c2*)))]⦈

| *relop-f64*:⦇[$*C* (*ConstFloat64 c1*), $*C* (*ConstFloat64 c2*), $(*Relop-f T-f64 fop*)]⦈ $\leadsto$ ⦇[$*C* (*ConstInt32* (*wasm-bool* (*app-relop-f fop c1 c2*)))]⦈

— *convert*

| *convert-Some*:⟦*types-agree t1 v*; *cvt t2 sx v* = (*Some v'*)⟧ $\Longrightarrow$ ⦇[$(*C v*), $(*Cvtop t2 Convert t1 sx*)]⦈ $\leadsto$ ⦇[$(*C v'*)]⦈

| *convert-None*:⟦*types-agree t1 v*; *cvt t2 sx v* = *None*⟧ $\Longrightarrow$ ⦇[$(*C v*), $(*Cvtop t2 Convert t1 sx*)]⦈ $\leadsto$ ⦇[*Trap*]⦈

— *reinterpret*

| *reinterpret*:*types-agree t1 v* $\Longrightarrow$ ⦇[$(*C v*), $(*Cvtop t2 Reinterpret t1 None*)]⦈ $\leadsto$ ⦇[$(*C* (*wasm-deserialise* (*bits v*) *t2*))]⦈

— *unreachable*

| *unreachable*:⦇[$ *Unreachable*]⦈ $\leadsto$ ⦇[*Trap*]⦈

— *nop*

| *nop*:⦇[$ *Nop*]⦈ $\leadsto$ ⦇[]⦈

— *drop*

| *drop*:⦇[$(*C v*), ($ *Drop*)]⦈ $\leadsto$ ⦇[]⦈

— *select*

| *select-false*:*int-eq n 0* $\Longrightarrow$ ⦇[$(*C v1*), $(*C v2*), $*C* (*ConstInt32 n*), ($ *Select*)]⦈ $\leadsto$ ⦇[$(*C v2*)]⦈

| *select-true*:*int-ne n 0* $\Longrightarrow$ ⦇[$(*C v1*), $(*C v2*), $*C* (*ConstInt32 n*), ($ *Select*)]⦈ $\leadsto$ ⦇[$(*C v1*)]⦈

— *block*

| *block*:⟦*const-list vs*; *length vs* = *n*; *length t1s* = *n*; *length t2s* = *m*⟧ $\Longrightarrow$ ⦇*vs* @ [$(*Block* (*t1s* -> *t2s*) *es*)]⦈ $\leadsto$ ⦇[*Label m* [] (*vs* @ ($* *es*))]⦈

— *loop*

| *loop*:⟦*const-list vs*; *length vs* = *n*; *length t1s* = *n*; *length t2s* = *m*⟧ $\Longrightarrow$ ⦇*vs* @ [$(*Loop* (*t1s* -> *t2s*) *es*)]⦈ $\leadsto$ ⦇[*Label n* [$(*Loop* (*t1s* -> *t2s*) *es*)] (*vs* @ ($* *es*))]⦈

— *if*

| *if-false*:*int-eq n 0* $\Longrightarrow$ ⦇[$*C* (*ConstInt32 n*), $(*If tf e1s e2s*)]⦈ $\leadsto$ ⦇[$(*Block tf e2s*)]⦈

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

inductive reduce :: [s, v list, e list, nat, s, v list, e list] ⇒ bool (⦇-;-;-⦈ ⇝′- - ⦇-;-;-⦈ 60) **where**
— lifting basic reduction
basic:⦇e⦈ ⇝ ⦇e′⦈ ⟹ ⦇s;vs;e⦈ ⇝-i ⦇s;vs;e′⦈
— call
| call:⦇s;vs;[$(Call j)]⦈ ⇝-i ⦇s;vs;[Callcl (sfunc s i j)]⦈
— call-indirect
| call-indirect-Some:⟦stab s i (nat-of-int c) = Some cl; stypes s i j = tf; cl-type cl = tf⟧ ⟹ ⦇s;vs;[$C (ConstInt32 c), $(Call-indirect j)]⦈ ⇝-i ⦇s;vs;[Callcl cl]⦈
| call-indirect-None:⟦(stab s i (nat-of-int c) = Some cl ∧ stypes s i j ≠ cl-type cl) ∨ stab s i (nat-of-int c) = None⟧ ⟹ ⦇s;vs;[$C (ConstInt32 c), $(Call-indirect j)]⦈ ⇝-i ⦇s;vs;[Trap]⦈
— call
| callcl-native:⟦cl = Func-native j (t1s -> t2s) ts es; ves = ($$∗ vcs); length vcs = n; length ts = k; length t1s = n; length t2s = m; (n-zeros ts = zs) ⟧ ⟹ ⦇s;vs;ves @ [Callcl cl]⦈ ⇝-i ⦇s;vs;[Local m j (vcs@zs) [$(Block ([] -> t2s) es)]]⦈
| callcl-host-Some:⟦cl = Func-host (t1s -> t2s) f; ves = ($$∗ 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]⦈ ⇝-i ⦇s′;vs;($$∗ vcs′)⦈
| callcl-host-None:⟦cl = Func-host (t1s -> t2s) f; ves = ($$∗ vcs); length vcs = n; length t1s = n; length t2s = m⟧ ⟹ ⦇s;vs;ves @ [Callcl cl]⦈ ⇝-i ⦇s;vs;[Trap]⦈
— get-local
| get-local:⟦length vi = j⟧ ⟹ ⦇s;(vi @ [v] @ vs);[$(Get-local j)]⦈ ⇝-i ⦇s;(vi @ [v]

@ *vs*);[$(*C v*)]⦈

— *set-local*

| *set-local*:⟦*length vi* = *j*⟧ ⟹ ⦇*s*;(*vi* @ [*v*] @ *vs*);[$(*C v′*), $(*Set-local j*)]⦈ ↝-*i* ⦇*s*;(*vi* @ [*v′*] @ *vs*);[]⦈

— *get-global*

| *get-global*:⦇*s*;*vs*;[$(*Get-global j*)]⦈ ↝-*i* ⦇*s*;*vs*;[$ *C*(*sglob-val s i j*)]⦈

— *set-global*

| *set-global*:*supdate-glob s i j v* = *s′* ⟹ ⦇*s*;*vs*;[$(*C v*), $(*Set-global j*)]⦈ ↝-*i* ⦇*s′*;*vs*;[]⦈

— *load*

| *load-Some*:⟦*smem-ind s i* = *Some j*; ((*mem s*)!*j*) = *m*; *load m* (*nat-of-int k*) *off* (*t-length t*) = *Some bs*⟧ ⟹ ⦇*s*;*vs*;[$*C* (*ConstInt32 k*), $(*Load t None a off*)]⦈ ↝-*i* ⦇*s*;*vs*;[$*C* (*wasm-deserialise bs t*)]⦈

| *load-None*:⟦*smem-ind s i* = *Some j*; ((*mem s*)!*j*) = *m*; *load m* (*nat-of-int k*) *off* (*t-length t*) = *None*⟧ ⟹ ⦇*s*;*vs*;[$*C* (*ConstInt32 k*), $(*Load t None a off*)]⦈ ↝-*i* ⦇*s*;*vs*;[*Trap*]⦈

— *load packed*

| *load-packed-Some*:⟦*smem-ind s i* = *Some j*; ((*mem s*)!*j*) = *m*; *load-packed sx m* (*nat-of-int k*) *off* (*tp-length tp*) (*t-length t*) = *Some bs*⟧ ⟹ ⦇*s*;*vs*;[$*C* (*ConstInt32 k*), $(*Load t* (*Some* (*tp*, *sx*)) *a off*)]⦈ ↝-*i* ⦇*s*;*vs*;[$*C* (*wasm-deserialise bs t*)]⦈

| *load-packed-None*:⟦*smem-ind s i* = *Some j*; ((*mem s*)!*j*) = *m*; *load-packed sx m* (*nat-of-int k*) *off* (*tp-length tp*) (*t-length t*) = *None*⟧ ⟹ ⦇*s*;*vs*;[$*C* (*ConstInt32 k*), $(*Load t* (*Some* (*tp*, *sx*)) *a off*)]⦈ ↝-*i* ⦇*s*;*vs*;[*Trap*]⦈

— *store*

| *store-Some*:⟦*types-agree t v*; *smem-ind s i* = *Some j*; ((*mem s*)!*j*) = *m*; *store m* (*nat-of-int k*) *off* (*bits v*) (*t-length t*) = *Some mem′*⟧ ⟹ ⦇*s*;*vs*;[$*C* (*ConstInt32 k*), $*C v*, $(*Store t None a off*)]⦈ ↝-*i* ⦇*s*⦇*mem*:= ((*mem s*)[*j* := *mem′*])⦈;*vs*;[]⦈

| *store-None*:⟦*types-agree t v*; *smem-ind s i* = *Some j*; ((*mem s*)!*j*) = *m*; *store m* (*nat-of-int k*) *off* (*bits v*) (*t-length t*) = *None*⟧ ⟹ ⦇*s*;*vs*;[$*C* (*ConstInt32 k*), $*C v*, $(*Store t None a off*)]⦈ ↝-*i* ⦇*s*;*vs*;[*Trap*]⦈

— *store packed*

| *store-packed-Some*:⟦*types-agree t v*; *smem-ind s i* = *Some j*; ((*mem s*)!*j*) = *m*; *store-packed m* (*nat-of-int k*) *off* (*bits v*) (*tp-length tp*) = *Some mem′*⟧ ⟹ ⦇*s*;*vs*;[$*C* (*ConstInt32 k*), $*C v*, $(*Store t* (*Some tp*) *a off*)]⦈ ↝-*i* ⦇*s*⦇*mem*:= ((*mem s*)[*j* := *mem′*])⦈;*vs*;[]⦈

| *store-packed-None*:⟦*types-agree t v*; *smem-ind s i* = *Some j*; ((*mem s*)!*j*) = *m*; *store-packed m* (*nat-of-int k*) *off* (*bits v*) (*tp-length tp*) = *None*⟧ ⟹ ⦇*s*;*vs*;[$*C* (*ConstInt32 k*), $*C v*, $(*Store t* (*Some tp*) *a off*)]⦈ ↝-*i* ⦇*s*;*vs*;[*Trap*]⦈

— *current-memory*

| *current-memory*:⟦*smem-ind s i* = *Some j*; ((*mem s*)!*j*) = *m*; *mem-size m* = *n*⟧ ⟹ ⦇*s*;*vs*;[$(*Current-memory*)]⦈ ↝-*i* ⦇*s*;*vs*;[$*C* (*ConstInt32* (*int-of-nat n*))]⦈

— *grow-memory*

| *grow-memory*:⟦*smem-ind s i* = *Some j*; ((*mem s*)!*j*) = *m*; *mem-size m* = *n*; *mem-grow m* (*nat-of-int c*) = *mem′*⟧ ⟹ ⦇*s*;*vs*;[$*C* (*ConstInt32 c*), $(*Grow-memory*)]⦈ ↝-*i* ⦇*s*⦇*mem*:= ((*mem s*)[*j* := *mem′*])⦈;*vs*;[$*C* (*ConstInt32* (*int-of-nat n*))]⦈

— *grow-memory fail*

| *grow-memory-fail*:⟦*smem-ind s i* = *Some j*; ((*mem s*)!*j*) = *m*; *mem-size m* = *n*⟧ ⟹ ⦇*s*;*vs*;[$*C* (*ConstInt32 c*),$(*Grow-memory*)]⦈ ↝-*i* ⦇*s*;*vs*;[$*C* (*ConstInt32 int32-minus-one*)]⦈

— *inductive label reduction*
| *label*:$[\![(\!| s;vs;es |\!) \leadsto\text{-}i\ (\!| s';vs';es' |\!);\ Lfilled\ k\ lholed\ es\ les;\ Lfilled\ k\ lholed\ es'\ les']\!] \Longrightarrow (\!| s;vs;les |\!) \leadsto\text{-}i\ (\!| s';vs';les' |\!)$
— *inductive local reduction*
| *local*:$[\![(\!| s;vs;es |\!) \leadsto\text{-}i\ (\!| s';vs';es' |\!)]\!] \Longrightarrow (\!| s;v0s;[Local\ n\ i\ vs\ es] |\!) \leadsto\text{-}j\ (\!| s';v0s;[Local\ n\ i\ vs'\ es'] |\!)$

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* ⟹ *host-apply s* (*t1s* -> *t2s*) *f vs hs* = *Some* (*s′*, *vs′*) ⟹ *store-extension s s′*
and *host-apply-respect-type*:*list-all2 types-agree t1s vs* ⟹ *host-apply s* (*t1s* -> *t2s*) *f vs hs* = *Some* (*s′*, *vs′*) ⟹ *list-all2 types-agree t2s vs′*
end

5 Auxiliary Type System Properties

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

lemma *typeof-i32*:
assumes *typeof v* = *T-i32*
shows ∃ *c*. *v* = *ConstInt32 c*
using *assms*
unfolding *typeof-def*
by (*cases v*) *auto*

lemma *typeof-i64*:
assumes *typeof v* = *T-i64*
shows ∃ *c*. *v* = *ConstInt64 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 *exists-v-typeof*: $\exists\, v\ v.\ typeof\ v = t$
proof (*cases t*)
  case *T-i32*
  fix *v*
  have *typeof* (*ConstInt32 v*) = *t*
    using *T-i32*
    unfolding *typeof-def*
    by *simp*
  thus *?thesis*
    using *T-i32*
    by *fastforce*
next
  case *T-i64*
  fix *v*
  have *typeof* (*ConstInt64 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 $\geq$ *l*
  shows $\exists$ *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** $\mathcal{S}$•$\mathcal{C}$ ⊢ $*b-es : *tf* **shows** $\mathcal{C}$ ⊢ *b-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* $\mathcal{S}$
  shows *length* (*s.mem s*) = *length* (*s-mem* $\mathcal{S}$)
  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* -> *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* -> *ts'*)) = (*ts* = *ts'*)
proof (*safe*)
assume $\mathcal{C} \vdash$ [] : (*ts* -> *ts'*)
thus *ts* = *ts'*
by *blast*
next
assume *ts* = *ts'*
thus $\mathcal{C} \vdash$ [] : (*ts'* -> *ts'*)
using *b-e-typing.empty b-e-typing.weakening*
by *fastforce*
qed

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

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

load-store-t-bounds a (*option-projl tp-sx*) *t*
using *assms*
by (*induction* [*e*] (*ts* -> *ts'*) *arbitrary*: *ts ts' rule*: *b-e-typing.induct*, *auto*)

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

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

lemma *b-e-type-grow-memory*:
assumes $\mathcal{C}$ ⊢ [*e*] : (*ts* -> *ts'*)
e = *Grow-memory*
shows ∃ *ts''*. *ts* = *ts''*@[*T-i32*] ∧ *ts* = *ts'* ∧ (∃ *n*. (*memory* $\mathcal{C}$) = *Some n*)
using *assms*
by (*induction* [*e*] (*ts* -> *ts'*) *arbitrary*: *ts ts' rule*: *b-e-typing.induct*) *auto*

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

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

lemma *b-e-type-binop-relop*:
assumes $\mathcal{C}$ ⊢ [*e*] : (*ts* -> *ts'*)
e = *Binop-i t iop* ∨ *e* = *Binop-f t fop* ∨ *e* = *Relop-i t irop* ∨ *e* = *Relop-f t frop*
shows ∃ *ts''*. *ts* = *ts''*@[*t*,*t*] ∧ *ts'* = *ts''*@[*arity-2-result*(*e*)]
e = *Binop-f t fop* ⟹ *is-float-t t*
e = *Relop-f t frop* ⟹ *is-float-t t*
using *assms*

unfolding *arity-2-result-def*
by (*induction* [*e*] (*ts* -> *ts'*) *arbitrary*: *ts ts' rule*: *b-e-typing.induct*, *auto*)

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

definition *arity-1-result* :: *b-e* ⇒ *t* **where**
arity-1-result op1 = (*case op1 of*
Unop-i t - ⇒ *t*
| *Unop-f t* - ⇒ *t*
| *Testop t* - ⇒ *T-i32*
| *Cvtop t1 Convert* - - ⇒ *t1*
| *Cvtop t1 Reinterpret* - - ⇒ *t1*)

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

lemma *b-e-type-cvtop*:
assumes $\mathcal{C} \vdash$ [*e*] : (*ts* -> *ts'*)
e = *Cvtop t1 cvtop t sx*
shows ∃ *ts''*. *ts* = *ts''*@[*t*] ∧ *ts'* = *ts''*@[*arity-1-result e*]
cvtop = *Convert* ⟹ (*t1* ≠ *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-length t1* = *t-length t*
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 ∃ *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*
shows ∃ *ts'' t*. *ts* = *ts''*@[*t*,*t*,*T-i32*] ∧ *ts'* = *ts''*@[*t*]

using *assms*
by (*induction* [*e*] (*ts* -> *ts'*) *arbitrary*: *ts ts' rule*: *b-e-typing.induct*, *auto*)

lemma *b-e-type-call*:
assumes $\mathcal{C} \vdash [e] : (ts \text{ -> } ts')$
$e = Call\ i$
shows $i < length\ (func\text{-}t\ \mathcal{C})$
$\exists ts''\ tf1\ tf2.\ ts = ts''@tf1 \wedge ts' = ts''@tf2 \wedge (func\text{-}t\ \mathcal{C})!i = (tf1 \text{ -> } tf2)$
using *assms*
by (*induction* [*e*] (*ts* -> *ts'*) *arbitrary*: *ts ts' rule*: *b-e-typing.induct*, *auto*)

lemma *b-e-type-call-indirect*:
assumes $\mathcal{C} \vdash [e] : (ts \text{ -> } ts')$
$e = Call\text{-}indirect\ i$
shows $i < length\ (types\text{-}t\ \mathcal{C})$
$\exists ts''\ tf1\ tf2.\ ts = ts''@tf1@[T\text{-}i32] \wedge ts' = ts''@tf2 \wedge (types\text{-}t\ \mathcal{C})!i = (tf1$
$\text{-> } tf2)$
using *assms*
by (*induction* [*e*] (*ts* -> *ts'*) *arbitrary*: *ts ts' rule*: *b-e-typing.induct*, *auto*)

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

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

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

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

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

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

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

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

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

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

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

lemma *b-e-type-comp*:
  assumes $\mathcal{C} \vdash es@[e] : (t1s \text{ -> } t4s)$
  shows $\exists ts'.\ (\mathcal{C} \vdash es : (t1s \text{ -> } ts')) \wedge (\mathcal{C} \vdash [e] : (ts' \text{ -> } t4s))$
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 $\mathcal{S}\cdot\mathcal{C} \vdash [\$e1, \$e2] : (t1s \text{ -> } t2s)$
  shows $\exists ts'.\ (\mathcal{C} \vdash [e1] : (t1s \text{ -> } ts')) \wedge (\mathcal{C} \vdash [e2] : (ts' \text{ -> } t2s))$
  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 $\mathcal{C} \vdash [e1] : (t1s \text{ -> } ts')$ $\mathcal{C} \vdash [e2] : (ts' \text{ -> } t2s)$
  shows $\mathcal{S}\cdot\mathcal{C} \vdash [\$e1, \$e2] : (ts@t1s \text{ -> } ts@t2s)$
  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 $\mathcal{C} \vdash [C\ v1,\ C\ v2] : (t1s\ \text{->}\ t2s)$
shows $t2s = t1s\ @\ [typeof\ v1,\ typeof\ v2]$
proof −
obtain ts' **where** $ts'\text{-}def{:}\mathcal{C} \vdash [C\ v1] : (t1s\ \text{->}\ ts')$
$\mathcal{C} \vdash [C\ v2] : (ts'\ \text{->}\ t2s)$
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 $\mathcal{S}{\cdot}\mathcal{C} \vdash es@[e] : (t1s\ \text{->}\ t3s)$
shows $\exists\, ts'.\ (\mathcal{S}{\cdot}\mathcal{C} \vdash es : (t1s\ \text{->}\ ts')) \wedge (\mathcal{S}{\cdot}\mathcal{C} \vdash [e] : (ts'\ \text{->}\ t3s))$
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* $es@[e]$ $(t1s\ \text{->}\ t3s)$ *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 $(\$*es'') = es\ (\$e'') = e$
using *1*(*2*) *inj-basic map-injective*
by *auto*
moreover
have $\mathcal{C} \vdash (es''\ @\ [e'']) : (t1s\ \text{->}\ t3s)$ **using** *1*(*1*)
using *inj-basic map-injective b-e-defs*
by *blast*
then obtain *t2s* **where** $\mathcal{C} \vdash es'' : (t1s\ \text{->}\ t2s)\ \mathcal{C} \vdash [e''] : (t2s\ \text{->}\ t3s)$
using *b-e-type-comp*
by *blast*
ultimately
show *?case*
using *e-typing-s-typing.intros*(*1*)
by *fastforce*
next
case (*3* $\mathcal{S}$ $\mathcal{C}$ *t1s t2s ts*)
thus *?case*

```
    using e-typing-s-typing.intros(3)
    by fastforce
  qed auto
qed

lemma e-type-comp-conc:
  assumes 𝒮•𝒞 ⊢ es : (t1s -> t2s)
          𝒮•𝒞 ⊢ es' : (t2s -> t3s)
  shows 𝒮•𝒞 ⊢ es@es' : (t1s -> 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 𝒮•𝒞 ⊢ xs : (t2s -> ts') 𝒮•𝒞 ⊢ [x] : (ts' -> 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 𝒞 ⊢ es : (t1s -> t2s)
          𝒞 ⊢ es' : (t2s -> t3s)
  shows 𝒞 ⊢ es@es' : (t1s -> t3s)
proof −
  fix 𝒮
  have 1:𝒮•𝒞 ⊢ $*es : (t1s -> t2s)
    using e-typing-s-typing.intros(1)[OF assms(1)]
    by fastforce
  have 2:𝒮•𝒞 ⊢ $*es' : (t2s -> 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 𝒮•𝒞 ⊢ es@es' : (ts -> ts')
  shows ∃ts''. (𝒮•𝒞 ⊢ es : (ts -> ts'')) ∧ (𝒮•𝒞 ⊢ es' : (ts'' -> 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 $\mathcal{S}$•$\mathcal{C}$ ⊢ *es*@*es′*@*es″* : (*t1s* -> *t2s*)
  shows ∃ *ts′ ts″*. ($\mathcal{S}$•$\mathcal{C}$ ⊢ *es* : (*t1s* -> *ts′*))
              ∧ ($\mathcal{S}$•$\mathcal{C}$ ⊢ *es′* : (*ts′* -> *ts″*))
              ∧ ($\mathcal{S}$•$\mathcal{C}$ ⊢ *es″* : (*ts″* -> *t2s*))
proof −
  obtain *ts′* **where** $\mathcal{S}$•$\mathcal{C}$ ⊢ *es* : (*t1s* -> *ts′*) $\mathcal{S}$•$\mathcal{C}$ ⊢ *es′*@*es″* : (*ts′* -> *t2s*)
    using *assms*(*1*) *e-type-comp-conc1*
    by *fastforce*
  moreover
  then obtain *ts″* **where** $\mathcal{S}$•$\mathcal{C}$ ⊢ *es′* : (*ts′* -> *ts″*) $\mathcal{S}$•$\mathcal{C}$ ⊢ *es″* : (*ts″* -> *t2s*)
    using *e-type-comp-conc1*
    by *fastforce*
  ultimately
  show *?thesis*
    by *fastforce*
qed

lemma *b-e-type-value-list*:
  assumes ($\mathcal{C}$ ⊢ *es*@[*C v*] : (*ts* -> *ts′*@[*t*]))
  shows ($\mathcal{C}$ ⊢ *es* : (*ts* -> *ts′*))
      (*typeof v* = *t*)
proof −
  obtain *ts″* **where** ($\mathcal{C}$ ⊢ *es* : (*ts* -> *ts″*)) ($\mathcal{C}$ ⊢ [*C v*] : (*ts″* -> *ts′* @ [*t*]))
    using *b-e-type-comp assms*
    by *blast*
  thus ($\mathcal{C}$ ⊢ *es* : (*ts* -> *ts′*)) (*typeof v* = *t*)
    using *b-e-type-value*[*of* $\mathcal{C}$ *C v ts″ ts′* @ [*t*]]
    by *auto*
qed

lemma *e-type-label*:
  assumes $\mathcal{S}$•$\mathcal{C}$ ⊢ [*Label n es0 es*] : (*ts* -> *ts′*)
  shows ∃ *tls t2s*. (*ts′* = (*ts*@*t2s*))
          ∧ *length tls* = *n*

$\wedge$ ($\mathcal{S}$•$\mathcal{C}$ $\vdash$ *es0* : (*tls* -> *t2s*))
$\wedge$ ($\mathcal{S}$•$\mathcal{C}$(|*label* := [*tls*] @ (*label* $\mathcal{C}$)|) $\vdash$ *es* : ([] -> *t2s*))
using *assms*
proof (*induction* $\mathcal{S}$ $\mathcal{C}$ [*Label n es0 es*] (*ts* -> *ts'*) *arbitrary*: *ts ts'*)
case (*1* $\mathcal{C}$ *b-es* $\mathcal{S}$)
then show *?case*
by (*simp add*: *map-eq-Cons-conv*)
next
case (*2* $\mathcal{S}$ $\mathcal{C}$ *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* $\mathcal{S}$ $\mathcal{C}$ *t1s t2s ts*)
then show *?case*
by *simp*
next
case (*7* $\mathcal{S}$ $\mathcal{C}$ *t2s*)
then show *?case*
by *fastforce*
qed

lemma *e-type-callcl-native*:
assumes $\mathcal{S}$•$\mathcal{C}$ $\vdash$ [*Callcl cl*] : (*t1s'* -> *t2s'*)
cl = *Func-native i tf ts es*
shows $\exists$ *t1s t2s ts-c*. (*t1s'* = *ts-c* @ *t1s*)
$\wedge$ (*t2s'* = *ts-c* @ *t2s*)
$\wedge$ *tf* = (*t1s* -> *t2s*)
$\wedge$ *i* < *length* (*s-inst* $\mathcal{S}$)
$\wedge$ (((*s-inst* $\mathcal{S}$)!*i*)(|*local* := (*local* ((*s-inst* $\mathcal{S}$)!*i*)) @ *t1s* @ *ts*, *label* := ([*t2s*] @ (*label* ((*s-inst* $\mathcal{S}$)!*i*))), *return* := *Some t2s*|) $\vdash$ *es* : ([] -> *t2s*))
using *assms*
proof (*induction* $\mathcal{S}$ $\mathcal{C}$ [*Callcl cl*] (*t1s'* -> *t2s'*) *arbitrary*: *t1s' t2s'*)
case (*1* $\mathcal{C}$ *b-es* $\mathcal{S}$)
thus *?case*
by *auto*
next
case (*2* $\mathcal{S}$ $\mathcal{C}$ *es t1s t2s e t3s*)
have $\mathcal{C}$ $\vdash$ [] : (*t1s* -> *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* $\mathcal{S}$ $\mathcal{C}$ *t1s t2s ts*)
thus *?case*
by *fastforce*
next
case (*6* $\mathcal{S}$ $\mathcal{C}$)

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

lemma e-type-callcl-host:
  assumes 𝒮•𝒞 ⊢ [Callcl cl] : (t1s' -> t2s')
          cl = Func-host tf f
  shows ∃ t1s t2s ts-c. (t1s' = ts-c @ t1s)
                      ∧ (t2s' = ts-c @ t2s)
                      ∧ tf = (t1s -> 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 ⊢ [] : (t1s -> 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 𝒮•𝒞 ⊢ [Callcl cl] : (t1s' -> t2s')
  shows ∃ t1s t2s ts-c. (t1s' = ts-c @ t1s)
                      ∧ (t2s' = ts-c @ t2s)
                      ∧ cl-type cl = (t1s -> 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 $\mathcal{S}$•*rs* ⊩-*i vs*;*es* : *ts*
shows *i* < *length* (*s-inst* $\mathcal{S}$)
(*rs* = *Some ts*) ∨ *rs* = *None*
($\mathcal{S}$•((*s-inst* $\mathcal{S}$)!*i*)⦇*local* := (*local* ((*s-inst* $\mathcal{S}$)!*i*)) @ (*map typeof vs*), *return* := *rs*⦈ ⊢ *es* : ([] -> *ts*))
using *assms*
by (*induction vs es ts*, *auto*)

lemma *e-type-local*:
assumes $\mathcal{S}$•$\mathcal{C}$ ⊢ [*Local n i vs es*] : (*ts* -> *ts'*)
shows ∃ *tls*. *i* < *length* (*s-inst* $\mathcal{S}$)
∧ *length tls* = *n*
∧ ($\mathcal{S}$•((*s-inst* $\mathcal{S}$)!*i*)⦇*local* := (*local* ((*s-inst* $\mathcal{S}$)!*i*)) @ (*map typeof vs*), *return* := *Some tls*⦈ ⊢ *es* : ([] -> *tls*))
∧ *ts'* = *ts* @ *tls*
using *assms*
proof (*induction* $\mathcal{S}$ $\mathcal{C}$ [*Local n i vs es*] (*ts* -> *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*
by *simp*
qed (*auto simp add*: *unlift-b-e s-typing.simps*)

lemma *e-type-local-shallow*:
assumes $\mathcal{S}$•$\mathcal{C}$ ⊢ [*Local n i vs es*] : (*ts* -> *ts'*)
shows ∃ *tls*. *length tls* = *n* ∧ *ts'* = *ts*@*tls* ∧ ($\mathcal{S}$•(*Some tls*) ⊩-*i vs*;*es* : *tls*)
using *assms*
proof (*induction* $\mathcal{S}$ $\mathcal{C}$ [*Local n i vs es*] (*ts* -> *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* ∘ *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* ∘ *EConst*) *vs*
by *blast*
moreover
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 ∃t. (𝒮•𝒞 ⊢ [e] : (ts -> ts@[t]))
  using assms
proof (cases e)
  case (Basic x1)
  then show ?thesis
    using assms
  proof (cases x1)
    case (EConst x23)
      hence 𝒞 ⊢ [x1] : ([] -> [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
          𝒮•𝒞 ⊢ [e] : (ts -> ts')
  shows  ∃t. (ts' = ts@[t]) ∧ (𝒮•𝒞' ⊢ [e] : ([] -> [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 𝒮•𝒞' ⊢ [e] : ([] -> [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 𝒮•𝒞 ⊢ [$C v] : ([] -> [t])
  shows typeof v = t
```

using *assms*
proof −
have $\mathcal{C} \vdash [C\ v] : ([] \text{ -> } [t])$
using *unlift-b-e assms*
by *fastforce*
thus *?thesis*
by (*induction* $[C\ v]$ $([] \text{ -> } [t])$ *rule*: *b-e-typing.induct*, *auto*)
qed

lemma *e-type-const-list*:
assumes *const-list vs*
$\mathcal{S}{\bullet}\mathcal{C} \vdash vs : (ts \text{ -> } ts')$
shows $\exists\, tvs.\ ts' = ts\ @\ tvs \wedge length\ vs = length\ tvs \wedge (\mathcal{S}{\bullet}\mathcal{C}' \vdash vs : ([] \text{ -> } tvs))$
using *assms*
proof (*induction vs arbitrary*: *ts ts′ rule*: *List.rev-induct*)
case *Nil*
have $\mathcal{S}{\bullet}\mathcal{C}' \vdash [] : ([] \text{ -> } [])$
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*:$\mathcal{S}{\bullet}\mathcal{C} \vdash xs : (ts \text{ -> } ts'')$ $\mathcal{S}{\bullet}\mathcal{C} \vdash [x] : (ts'' \text{ -> } ts')$
using *snoc*(*3*) *e-type-comp*
by *fastforce*
then obtain *ts-b* **where** *ts-b-def*:$ts'' = ts\ @\ ts\text{-}b$ *length xs* = *length ts-b* $\mathcal{S}{\bullet}\mathcal{C}' \vdash$ $xs : ([] \text{ -> } ts\text{-}b)$
using *snoc*(*1*) *v-lists*(*1*)
unfolding *const-list-def*
by *fastforce*
then obtain *t* **where** *t-def*:$ts' = ts\ @\ ts\text{-}b\ @\ [t]$ $\mathcal{S}{\bullet}\mathcal{C}' \vdash [x] : ([] \text{ -> } [t])$
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 $\mathcal{S}{\bullet}\mathcal{C}' \vdash (xs@[x]) : ([] \text{ -> } ts\text{-}b@[t])$
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*
          $\mathcal{S}\bullet\mathcal{C} \vdash vs : ([] \text{ -> } ts@[t])$
  shows $\exists\, vs1\ v2.\ (\mathcal{S}\bullet\mathcal{C} \vdash vs1 : ([] \text{ -> } ts))$
          $\wedge\ (\mathcal{S}\bullet\mathcal{C} \vdash [v2] : (ts \text{ -> } ts@[t]))$
          $\wedge\ (vs = vs1@[v2])$
          $\wedge$ *const-list vs1*
          $\wedge$ *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*:$\mathcal{S}\bullet\mathcal{C} \vdash vs' : ([] \text{ -> } ts')$ $\mathcal{S}\bullet\mathcal{C} \vdash [v] : (ts' \text{ -> } ts@[t])$
    using *vs-def assms*(*2*) *e-type-comp*[*of* $\mathcal{S}$ $\mathcal{C}$ *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* $\mathcal{S}$ $\mathcal{C}$ [*C c*]] *b-e-type-value*
    by *fastforce*
  thus *?thesis* **using** *ts′-def vs-def consts-def*
    by *simp*
qed

lemma *e-type-const-list-cons*:
  assumes *const-list vs*
          $\mathcal{S}\bullet\mathcal{C} \vdash vs : ([] \text{ -> } (ts1@ts2))$
  shows $\exists\, vs1\ vs2.\ (\mathcal{S}\bullet\mathcal{C} \vdash vs1 : ([] \text{ -> } ts1))$
          $\wedge\ (\mathcal{S}\bullet\mathcal{C} \vdash vs2 : (ts1 \text{ -> } (ts1@ts2)))$
          $\wedge\ vs = vs1@vs2$
          $\wedge$ *const-list vs1*
          $\wedge$ *const-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 𝒮•𝒞 ⊢ [] : (ts1 -> 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:(𝒮•𝒞 ⊢ vs1 : ([] -> ts1 @ ts2'))
                                (𝒮•𝒞 ⊢ [v2] : (ts1 @ ts2' -> 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 𝒮 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 ∃ vs. ves = $$* 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
          𝒮•𝒞 ⊢ lfilled : (ts -> ts')
```

shows $\exists$ *t1s t2s* $\mathcal{C}'$ *arb-label*. ($\mathcal{S}$•$\mathcal{C}$(|*label* := *arb-label*@(*label* $\mathcal{C}$)|) ⊢ *es* : (*t1s* -> *t2s*))
using *assms*
proof (*induction arbitrary*: $\mathcal{C}$ *ts ts'* *rule*: *Lfilled.induct*)
case (*L0 vs lholed es' es*)
hence $\mathcal{S}$•($\mathcal{C}$(|*label* := *label* $\mathcal{C}$|)) ⊢ *vs* @ *es* @ *es'* : (*ts* -> *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** $\mathcal{S}$•$\mathcal{C}$ ⊢ [*Label n es' lfilledk*] : (*ts''* -> *ts'''*)
using *e-type-comp-conc2*[*OF LN*(*5*)]
by *fastforce*
then obtain *t1s t2s ts* **where** *test*:$\mathcal{S}$•$\mathcal{C}$(|*label* := [*ts*] @ (*label* $\mathcal{C}$)|) ⊢ *lfilledk* : (*t1s* -> *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*
$\mathcal{S}$•$\mathcal{C}$ ⊢ *lfilled* : (*ts* -> *ts'*)
shows $\exists$ *t1s t2s* $\mathcal{C}'$ *arb-label arb-return*. ($\mathcal{S}$•$\mathcal{C}$(|*label* := *arb-label*, *return* := *arb-return*|) ⊢ *es* : (*t1s* -> *t2s*))
proof −
have $\exists$ *t1s t2s* $\mathcal{C}'$ *arb-label*. ($\mathcal{S}$•$\mathcal{C}$(|*label* := *arb-label*, *return* := (*return* $\mathcal{C}$)|) ⊢ *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* $\mathcal{S}$
$i <$ *length* (*s-inst* $\mathcal{S}$)
$j <$ *length* (*func-t* ((*s-inst* $\mathcal{S}$)!*i*))
shows (*sfunc-ind s i j*) $<$ *length* (*s-funcs* $\mathcal{S}$)
cl-typing $\mathcal{S}$ (*sfunc s i j*) ((*s-funcs* $\mathcal{S}$)!(*sfunc-ind s i j*))
((*s-funcs* $\mathcal{S}$)!(*sfunc-ind s i j*)) = (*func-t* ((*s-inst* $\mathcal{S}$)!*i*))!*j*
proof −
have *funcs-agree*:*list-all2* (*cl-typing* $\mathcal{S}$) (*funcs s*) (*s-funcs* $\mathcal{S}$)
using *assms*(*1*)
unfolding *store-typing.simps*

by *auto*
have *list-all2* (*funci-agree* (*s-funcs* $\mathcal{S}$)) (*inst.funcs* ((*inst* *s*)!*i*)) (*func-t* ((*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 *funci-agree* (*s-funcs* $\mathcal{S}$) ((*inst.funcs* ((*inst* *s*)!*i*))!*j*) ((*func-t* ((*s-inst* $\mathcal{S}$)!*i*))!*j*)
using *assms*(*3*) *list-all2-nthD2*
by *blast*
thus (*sfunc-ind s i j*) < *length* (*s-funcs* $\mathcal{S}$)
((*s-funcs* $\mathcal{S}$)!(*sfunc-ind s i j*)) = (*func-t* ((*s-inst* $\mathcal{S}$)!*i*))!*j*
unfolding *funci-agree-def sfunc-ind-def*
by *auto*
thus *cl-typing* $\mathcal{S}$ (*sfunc s i j*) ((*s-funcs* $\mathcal{S}$)!(*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* $\mathcal{S}$
i < *length* (*s-inst* $\mathcal{S}$)
j < *length* (*global* ((*s-inst* $\mathcal{S}$)!*i*))
shows (*sglob-ind s i j*) < *length* (*s-globs* $\mathcal{S}$)
glob-agree (*sglob s i j*) ((*s-globs* $\mathcal{S}$)!(*sglob-ind s i j*))
((*s-globs* $\mathcal{S}$)!(*sglob-ind s i j*)) = (*global* ((*s-inst* $\mathcal{S}$)!*i*))!*j*
proof −
have *globs-agree*:*list-all2 glob-agree* (*globs s*) (*s-globs* $\mathcal{S}$)
using *assms*(*1*)
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* $\mathcal{S}$

$i < \mathit{length}\ (\mathit{s\text{-}inst}\ \mathcal{S})$
$\mathit{smem\text{-}ind}\ s\ i = \mathit{Some}\ j$
shows $j < \mathit{length}\ (\mathit{s\text{-}mem}\ \mathcal{S})$
$\mathit{mem\text{-}agree}\ ((\mathit{mem}\ s)!j)\ ((\mathit{s\text{-}mem}\ \mathcal{S})!j)$
$\exists x.\ ((\mathit{s\text{-}mem}\ \mathcal{S})!j) = x \wedge (\mathit{memory}\ ((\mathit{s\text{-}inst}\ \mathcal{S})!i)) = \mathit{Some}\ x$
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 $j < \mathit{length}\ (\mathit{s\text{-}mem}\ \mathcal{S})$
$\exists x.\ ((\mathit{s\text{-}mem}\ \mathcal{S})!j) = x \wedge (\mathit{memory}\ ((\mathit{s\text{-}inst}\ \mathcal{S})!i)) = \mathit{Some}\ x$
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* $\mathcal{S}$
$i < \mathit{length}\ (\mathit{s\text{-}inst}\ \mathcal{S})$
$\mathit{smem\text{-}ind}\ s\ i = \mathit{None}$
shows $(\mathit{memory}\ ((\mathit{s\text{-}inst}\ \mathcal{S})!i)) = \mathit{None}$
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-mem-exists*:
assumes $i < \mathit{length}\ (\mathit{s\text{-}inst}\ \mathcal{S})$
store-typing s $\mathcal{S}$
shows *Option.is-none* (*memory* ((*s-inst* $\mathcal{S}$)!*i*)) = *Option.is-none* (*inst.mem* ((*inst s*)!*i*))
proof −

obtain j **where** *j-def*:$j = (inst.mem\ ((inst\ s)!i))$
  by *blast*
obtain m **where** *m-def*:$m = (memory\ ((s\text{-}inst\ \mathcal{S})!i))$
  by *blast*
have *inst-typ*:*inst-typing* $\mathcal{S}$ $((inst\ s)!i)$ $((s\text{-}inst\ \mathcal{S})!i)$
  using *assms*
  unfolding *store-typing.simps list-all2-conv-all-nth*
  by *auto*
thus *?thesis*
  unfolding *inst-typing.simps memi-agree-def*
  by *auto*
qed

lemma *store-preserved-mem*:
assumes *store-typing* s $\mathcal{S}$
$s' = s(\!|s.mem := (s.mem\ s)[i := mem']|\!)$
mem-size $mem' \geq$ *mem-size orig-mem*
$((s.mem\ s)!i) =$ *orig-mem*
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** $\mathcal{S} = (\!|s\text{-}inst = \mathcal{C}s,\ s\text{-}funcs = tfs,\ s\text{-}tab = ns,\ s\text{-}mem = ms,\ s\text{-}globs = tgs|\!)$
  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 ($\lambda cls\ n.\ n \leq length\ cls$) $clss'$ *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 m. m* $\leq$ *mem-size bs*) *bss′ ms*
proof −
have *length bss* = *length bss′*
using *assms*(*2*) *s-S-defs*
by (*simp*)
moreover

have *initial-mem*:*list-all2* (λ *bs m. m* $\leq$ *mem-size bs*) *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 m. m* $\leq$ *mem-size bs*) (*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 m. m* $\leq$ *mem-size bs*) (*bss*!*n*) (*ms*!*n*)
using *initial-mem local-assms*
unfolding *list-all2-conv-all-nth*
by *simp*
thus (λ *bs m. m* $\leq$ *mem-size bs*) (*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 $\mathcal{S}$•$\mathcal{C}$ ⊢ [\$*C v*] : ([] -> [*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 $\mathcal{S}$•$\mathcal{C}$ ⊢ \$\$∗ *vs* : ([] -> *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 $\mathcal{S}\bullet\mathcal{C} \vdash$ [\$*C v*] : ([] -> [*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* (λ*v*. *C v*) *vs*) : (*ts'* -> *ts'*@*ts*)
shows *list-all2 types-agree ts vs*
using *assms*
proof (*induction* (*map* (λ*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 $\mathcal{S}\bullet\mathcal{C} \vdash$ (\$\$* *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* $\mathcal{S}$) *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*(*2*)
    unfolding *store-typing.simps*
    by *auto*
  moreover
  have *list-all2* (λ *bs m*. *m* $\leq$ *mem-size bs*) *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
qed
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⦈ ⇝-i ⦇s′;vs′;es′⦈
          store-typing s 𝒮
          𝒮•𝒞 ⊢ es : (ts -> ts′)
```

$\mathcal{C} = ((s\text{-}inst\ \mathcal{S})!i)(\!|local := local\ ((s\text{-}inst\ \mathcal{S})!i)\ @\ (map\ typeof\ vs),\ label := arb\text{-}label,\ return := arb\text{-}return|\!)$
$i < length\ (s\text{-}inst\ \mathcal{S})$
shows *store-typing s′ $\mathcal{S}$*
using *assms*
proof (*induction arbitrary*: $\mathcal{C}$ *arb-label arb-return ts ts′ rule*: *reduce.induct*)
case (*callcl-host-Some cl t1s t2s f ves vcs n m s hs s′ vcs′ vs i*)
obtain *ts″* **where** *ts″-def*:$\mathcal{S}\bullet\mathcal{C} \vdash ves : (ts \text{ -> } ts'')$ $\mathcal{S}\bullet\mathcal{C} \vdash [Callcl\ cl] : (ts'' \text{ -> } ts')$
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*:$ts'' = ts\ @\ tvs$
$length\ t1s = length\ tvs$
$\mathcal{S}\bullet\mathcal{C} \vdash ves : ([] \text{ -> } tvs)$
using *ts″-def*(*1*) *e-type-const-list*[*of ves* $\mathcal{S}$ $\mathcal{C}$ *ts ts″*] *callcl-host-Some*
by *fastforce*
hence $ts'' = ts\ @\ t1s$
$ts' = ts\ @\ t2s$
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** $\mathcal{S} = (\!|s\text{-}inst = \mathcal{C}s,\ s\text{-}funcs = tfs,\ s\text{-}tab = ns,\ s\text{-}mem = ms,\ s\text{-}globs = tgs|\!)$
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*
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 $\bigwedge j'$. $[\![j' \neq k;\ j' <$ *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 $\bigwedge j'$. $[\![j' \neq k;\ j' <$ *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*:$\mathcal{C} \vdash$ [*C v*] : (*ts* -> *ts''*) $\mathcal{C} \vdash$ [*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 $j < length\ (global\ ((s\text{-}inst\ \mathcal{S})!i))$
using *set-global*(*4*)
by *fastforce*
hence *globs-agree*:$k < length\ (s\text{-}globs\ \mathcal{S})$
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-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 k off mem' vs 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 s i j m k off tp 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 j n mem 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 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-label′*, *return* := *arb-return′*|) $\mathcal{S}$•$\mathcal{C}'$ ⊢ *es* : (*t1s* -> *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 i s′ vs′ es′ v0s n j*)
obtain *tls* **where** *t-local*:($\mathcal{S}$•((*s-inst* $\mathcal{S}$)!*i*)(|*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*|) ⇝-*i* (|*s′*;*vs′*;*es′*|)
store-typing s $\mathcal{S}$
$\mathcal{S}$•*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 $\mathcal{C} \vdash [C\ v,\ e] : (ts\ \text{->}\ ts')$
  using *unlift-b-e assms(1)*
  by *simp*
then obtain ts'' **where** ts''-*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 $(typeof\ v) = t\ e = (Unop\text{-}f\ t\ fop) \Longrightarrow is\text{-}float\text{-}t\ t$
  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 $\mathcal{S} \cdot \mathcal{C} \vdash [\$C\ v,\ \$e] : (ts\ \text{->}\ ts')$
    $e = Cvtop\ t1\ cvtop\ t\ sx$
  shows $(typeof\ v) = t$
    $cvtop = Convert \Longrightarrow (t1 \neq t) \wedge ((sx = None) = ((is\text{-}float\text{-}t\ t1 \wedge is\text{-}float\text{-}t$
$t) \vee (is\text{-}int\text{-}t\ t1 \wedge is\text{-}int\text{-}t\ t \wedge (t\text{-}length\ t1 < t\text{-}length\ t))))$
    $cvtop = Reinterpret \Longrightarrow (t1 \neq t) \wedge t\text{-}length\ t1 = t\text{-}length\ t$
proof −
  have $\mathcal{C} \vdash [C\ v,\ e] : (ts\ \text{->}\ ts')$
    using *unlift-b-e assms(1)*
    by *simp*
  then obtain ts'' **where** ts''-*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 $(typeof\ v) = t$
    $cvtop = Convert \Longrightarrow (t1 \neq t) \wedge (sx = None) = ((is\text{-}float\text{-}t\ t1 \wedge is\text{-}float\text{-}t\ t)$
$\vee (is\text{-}int\text{-}t\ t1 \wedge is\text{-}int\text{-}t\ t \wedge (t\text{-}length\ t1 < t\text{-}length\ t)))$
    $cvtop = Reinterpret \Longrightarrow (t1 \neq t) \wedge t\text{-}length\ t1 = t\text{-}length\ t$
    using *b-e-type-value*[*OF* ts''-*def*(1)] *b-e-type-cvtop*[*OF* ts''-*def*(2) *assms*(2)]
    by *simp-all*
qed

lemma *types-preserved-unop-testop-cvtop*:
  assumes $(\![\$C\ v,\ \$e]\!) \leadsto (\![\$C\ v']\!)$
    $\mathcal{S} \cdot \mathcal{C} \vdash [\$C\ v,\ \$e] : (ts\ \text{->}\ ts')$
    $(e = (Unop\text{-}i\ t\ iop)) \vee (e = (Unop\text{-}f\ t\ fop)) \vee (e = (Testop\ t\ testop)) \vee$
$(e = (Cvtop\ t2\ cvtop\ t\ sx))$
  shows $\mathcal{S} \cdot \mathcal{C} \vdash [\$C\ v'] : (ts\ \text{->}\ ts')$
proof −
  have $\mathcal{C} \vdash [C\ v,\ e] : (ts\ \text{->}\ ts')$
    using *unlift-b-e assms(2)*
    by *simp*
  then obtain ts'' **where** ts''-*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 $ts @ [arity\text{-}1\text{-}result\ e] = ts'\ (typeof\ v) = t$
    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*)
done
hence $\mathcal{C} \vdash$ [*C* v'] : ([] -> [*arity-1-result e*])
using *b-e-typing.const*
by *metis*
ultimately
show $\mathcal{S}\cdot\mathcal{C} \vdash$ [\$*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 $\mathcal{S}\cdot\mathcal{C} \vdash$ [\$*C v1*, \$*C v2*, \$*e*] : (*ts* -> ts')
e = *Binop-i t iop* ∨ *e* = *Binop-f t fop* ∨ *e* = *Relop-i t irop* ∨ *e* = *Relop-f t frop*
shows *typeof v1* = *t*
typeof v2 = *t*
e = *Binop-f t fop* ⟹ *is-float-t t*
e = *Relop-f t frop* ⟹ *is-float-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-f t fop* ⟹ *is-float-t t*
e = *Relop-f t frop* ⟹ *is-float-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 = *Binop-f t fop* ⟹ *is-float-t t*
e = *Relop-f t frop* ⟹ *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, $e]⦈ ⇝ ⦇[$C v']⦈
          𝒮•𝒞 ⊢ [$C v1, $C v2, $e] : (ts -> ts')
          e = Binop-i t iop ∨ e = Binop-f t fop ∨ e = Relop-i t irop ∨ e = Relop-f t frop
  shows 𝒮•𝒞 ⊢ [$C v'] : (ts -> ts')
proof −
  have 𝒞 ⊢ [C v1, C v2, e] : (ts -> ts')
    using unlift-b-e assms(2)
    by simp
  then obtain ts'' where ts''-def:𝒞 ⊢ [C v1, C v2] : (ts -> ts'') 𝒞 ⊢ [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]
    using assms(3) b-e-type-binop-relop[of 𝒞 e ts'' ts' t]
    by blast
  hence 𝒞 ⊢ [C v1] : (ts -> ts-id@[t])
    using ts''-def b-e-type-comp[of 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)
  hence 𝒞 ⊢ [C v'] : ([] -> [arity-2-result e])
    using b-e-typing.const
    by metis
  ultimately show ?thesis
    using e-typing-s-typing.intros(1)
          b-e-typing.weakening[of 𝒞 [C v'] [] [arity-2-result e] ts]
    by fastforce
qed

lemma types-preserved-drop:
  assumes ⦇[$C v, $e]⦈ ⇝ ⦇[]⦈
          𝒮•𝒞 ⊢ [$C v, $e] : (ts -> ts')
          (e = (Drop))
  shows 𝒮•𝒞 ⊢ [] : (ts -> ts')
proof −
  have 𝒞 ⊢ [C v, e] : (ts -> ts')
    using unlift-b-e assms(2)
    by simp
  then obtain ts'' where ts''-def:𝒞 ⊢ [C v] : (ts -> ts'') 𝒞 ⊢ [e] : (ts'' -> 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''*-def assms*(*3*) *b-e-type-drop*
  by *blast*
hence $\mathcal{C} \vdash [] : (ts \text{ -> } ts')$
  using *b-e-type-empty*
  by *simp*
thus *?thesis*
  using *e-typing-s-typing.intros*(*1*)
  by *fastforce*
qed

lemma *types-preserved-select*:
assumes $(\![\$C\ v1,\ \$C\ v2,\ \$C\ vn,\ \$e]\!) \leadsto (\![\$C\ v3]\!)$
$\mathcal{S}\cdot\mathcal{C} \vdash [\$C\ v1,\ \$C\ v2,\ \$C\ vn,\ \$e] : (ts \text{ -> } ts')$
$(e = Select)$
shows $\mathcal{S}\cdot\mathcal{C} \vdash [\$C\ v3] : (ts \text{ -> } ts')$
proof −
have $\mathcal{C} \vdash [C\ v1,\ C\ v2,\ C\ vn,\ e] : (ts \text{ -> } ts')$
  using *unlift-b-e assms*(*2*)
  by *simp*
then obtain *t1s* **where** *t1s-def*:$\mathcal{C} \vdash [C\ v1,\ C\ v2,\ C\ vn] : (ts \text{ -> } t1s)$ $\mathcal{C} \vdash [e] : (t1s \text{ -> } ts')$
  using *b-e-type-comp*[**where** $?e = e$ **and** $?es = [C\ v1,\ C\ v2,\ C\ vn]$]
  by *fastforce*
then obtain *t2s t* **where** *t2s-def*:$t1s = t2s\ @\ [t,\ t,\ (T\text{-}i32)]$ $ts' = t2s@[t]$
  using *b-e-type-select*[*of* $\mathcal{C}$ *e t1s*] *assms*
  by *fastforce*
hence $\mathcal{C} \vdash [C\ v1,\ C\ v2] : (ts \text{ -> } t2s@[t,t])$
  using *t1s-def t2s-def b-e-type-value-list*[*of* $\mathcal{C}$ $[C\ v1,\ C\ v2]$ *vn ts* $t2s@[t,t]$]
  by *fastforce*
hence *v2-t-def*:$\mathcal{C} \vdash [C\ v1] : (ts \text{ -> } t2s@[t])$ $typeof\ v2 = t$
  using *t1s-def t2s-def b-e-type-value-list*[*of* $\mathcal{C}$ $[C\ v1]$ *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 $\mathcal{C} \vdash [C\ v3] : (ts \text{ -> } ts')$
  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 $(\!|vs\ @\ [\$Block\ (tn\ \text{->}\ tm)\ es]|\!) \leadsto (\!|[Label\ m\ []\ (vs\ @\ (\$*\ es))]|\!)$
    $\mathcal{S}\bullet\mathcal{C} \vdash vs\ @\ [\$Block\ (tn\ \text{->}\ tm)\ es] : (ts\ \text{->}\ ts')$
    const-list vs
    length vs = *n*
    length tn = *n*
    length tm = *m*
  shows $\mathcal{S}\bullet\mathcal{C} \vdash [Label\ m\ []\ (vs\ @\ (\$*\ es))] : (ts\ \text{->}\ ts')$
proof −
  obtain $\mathcal{C}'$ **where** *c-def*:$\mathcal{C}' = \mathcal{C}(\!|label := [tm]\ @\ label\ \mathcal{C}|\!)$ **by** *blast*
  obtain ts'' **where** ts''*-def*:$\mathcal{S}\bullet\mathcal{C} \vdash vs : (ts\ \text{->}\ ts'')\ \mathcal{S}\bullet\mathcal{C} \vdash [\$Block\ (tn\ \text{->}\ tm)\ es] : (ts''\ \text{->}\ ts')$
    using *assms*(*2*) *e-type-comp*[*of* $\mathcal{S}$ $\mathcal{C}$ *vs* \$*Block* (*tn* -> *tm*) *es ts ts'*]
    by *fastforce*
  hence $\mathcal{C} \vdash [Block\ (tn\ \text{->}\ tm)\ es] : (ts''\ \text{->}\ ts')$
    using *unlift-b-e*
    by *auto*
  then obtain *ts-c tfn tfm* **where** *ts-c-def*:$(tn\ \text{->}\ tm) = (tfn\ \text{->}\ tfm)$ $ts'' = ts\text{-}c@tfn$ $ts' = ts\text{-}c@tfm$ $(\mathcal{C}(\!|label := [tfm]\ @\ label\ \mathcal{C}|\!) \vdash es : (tn\ \text{->}\ tm))$
    using *b-e-type-block*[*of* $\mathcal{C}$ *Block* (*tn* -> *tm*) *es ts'' ts'* (*tn* -> *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* $\mathcal{S}\bullet\mathcal{C}' \vdash vs : ([]\ \text{->}\ tvs')$
    using *e-type-const-list assms*(*3*,*4*) ts''*-def*(*1*)
    by *fastforce*
  hence $\mathcal{S}\bullet\mathcal{C}' \vdash vs : ([]\ \text{->}\ tn)\ \mathcal{S}\bullet\mathcal{C}' \vdash \$*es : (tn\ \text{->}\ tm)$
    using *ts-c-def tvs'-def tfn-l ts''-def c-def e-typing-s-typing.intros*(*1*)
    by *simp-all*
  hence $\mathcal{S}\bullet\mathcal{C}' \vdash (vs\ @\ (\$*\ es)) : ([]\ \text{->}\ tm)$ **using** *e-type-comp-conc*
    by *simp*
  moreover
  have $\mathcal{S}\bullet\mathcal{C} \vdash [] : (tm\ \text{->}\ tm)$
    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 *types-preserved-if*:
  assumes $(\!|[\$C\ ConstInt32\ n,\ \$If\ tf\ e1s\ e2s]|\!) \leadsto (\!|[\$Block\ tf\ es']|\!)$
    $\mathcal{S}\bullet\mathcal{C} \vdash [\$C\ ConstInt32\ n,\ \$If\ tf\ e1s\ e2s] : (ts\ \text{->}\ ts')$
  shows $\mathcal{S}\bullet\mathcal{C} \vdash [\$Block\ tf\ es'] : (ts\ \text{->}\ ts')$

proof −
  have $\mathcal{C} \vdash$ [*C ConstInt32 n*, *If tf e1s e2s*] : (*ts* -> *ts'*)
    using *unlift-b-e assms*(*2*)
    by *fastforce*
  then obtain *ts-i* **where** *ts-i-def*:$\mathcal{C} \vdash$ [*C ConstInt32 n*] : (*ts* -> *ts-i*) $\mathcal{C} \vdash$ [*If tf e1s e2s*] : (*ts-i* -> *ts'*)
    using *b-e-type-comp*
    by (*metis append-Cons append-Nil*)
  then obtain *ts'' tfn tfm* **where** *ts-def*:*tf* = (*tfn* -> *tfm*)
    ts-i = *ts''*@*tfn* @ [*T-i32*]
    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-i* = *ts* @ [(*T-i32*)]
    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* -> *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* -> *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-local i*]|) $\leadsto$ (|[*v*, *v*, $*Set-local i*]|)
    $\mathcal{S}$•$\mathcal{C} \vdash$ [*v*, $*Tee-local i*] : (*ts* -> *ts'*)
    is-const v
  shows $\mathcal{S}$•$\mathcal{C} \vdash$ [*v*, *v*, $*Set-local i*] : (*ts* -> *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-local i*] : (*ts* -> *ts'*)
    using *unlift-b-e assms*(*2*)
    by *fastforce*
  then obtain *ts''* **where** *ts''-def*:$\mathcal{C} \vdash$ [*C bv*] : (*ts* -> *ts''*) $\mathcal{C} \vdash$ [*Tee-local i*] : (*ts''* -> *ts'*)
    using *b-e-type-comp*[*of* - [*C bv*] *Tee-local i*]
    by *fastforce*
  then obtain *ts-c t* **where** *ts-c-def*:*ts''* = *ts-c*@[*t*] *ts'* = *ts-c*@[*t*] (*local* $\mathcal{C}$)!*i* = *t i*

< *length*(*local* 𝒞)
using *b-e-type-tee-local*[*of* 𝒞 *Tee-local i ts′′ ts′ i*]
by *fastforce*
hence *t-bv*:*t* = *typeof bv ts* = *ts-c*
using *b-e-type-value ts′′-def*
by *fastforce+*
have 𝒞 ⊢ [*Set-local i*] : ([*t*,*t*] -> [*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 𝒞 ⊢ [*C bv*] : ([*t*] -> [*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 𝒞 ⊢ [*C bv*, *C bv*] : ([] -> [*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 𝒞 ⊢ [*C bv*, *C bv*, *Set-local i*] : (*ts* -> *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* @ [$*Loop* (*t1s* -> *t2s*) *es*]|) ⇝ (|[*Label n* [$*Loop* (*t1s* -> *t2s*) *es*] (*vs* @ ($* *es*))]|)
𝒮•𝒞 ⊢ *vs* @ [$*Loop* (*t1s* -> *t2s*) *es*] : (*ts* -> *ts′*)
const-list vs
length vs = *n*
length t1s = *n*
length t2s = *m*
shows 𝒮•𝒞 ⊢ [*Label n* [$*Loop* (*t1s* -> *t2s*) *es*] (*vs* @ ($* *es*))] : (*ts* -> *ts′*)
proof −
obtain *ts′′* **where** *ts′′-def*:𝒮•𝒞 ⊢ *vs* : (*ts* -> *ts′′*) 𝒮•𝒞 ⊢ [$*Loop* (*t1s* -> *t2s*) *es*] : (*ts′′* -> *ts′*)
using *assms*(*2*) *e-type-comp*
by *fastforce*
then have 𝒞 ⊢ [*Loop* (*t1s* -> *t2s*) *es*] : (*ts′′* -> *ts′*)
using *unlift-b-e assms*(*2*)
by *fastforce*
then obtain *ts-c tfn tfm* 𝒞′ **where** *t-loop*:(*t1s* -> *t2s*) = (*tfn* -> *tfm*)
(*ts′′* = *ts-c*@*tfn*)
(*ts′* = *ts-c*@*tfm*)
𝒞′ = 𝒞(|*label* := [*t1s*] @ *label* 𝒞|)
(𝒞′ ⊢ *es* : (*tfn* -> *tfm*))

using *b-e-type-loop*[*of* $\mathcal{C}$ *Loop* (*t1s* -> *t2s*) *es ts''* *ts'*]
by *fastforce*
obtain *tvs* **where** *tvs-def*:*ts''* = *ts* @ *tvs length vs* = *length tvs* $\mathcal{S}$•$\mathcal{C}'$ ⊢ *vs* : ([] -> *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 $\mathcal{S}$•$\mathcal{C}$ ⊢ [$*Loop* (*t1s* -> *t2s*) *es*] : (*t1s* -> *t2s*)
using *t-loop b-e-typing.loop e-typing-s-typing.intros*(*1*)
by *fastforce*
moreover
have $\mathcal{S}$•$\mathcal{C}'$ ⊢ $∗*es* : (*t1s* -> *t2s*)
using *t-loop e-typing-s-typing.intros*(*1*)
by *fastforce*
then have $\mathcal{S}$•$\mathcal{C}'$ ⊢ *vs*@($∗*es*) : ([] -> *t2s*)
using *tvs-eq tvs-def*(*3*) *e-type-comp-conc*
by *blast*
ultimately
have $\mathcal{S}$•$\mathcal{C}$ ⊢ [*Label n* [$*Loop* (*t1s* -> *t2s*) *es*] (*vs* @ ($∗ *es*))] : ([] -> *t2s*)
using *e-typing-s-typing.intros*(*7*)[*of* $\mathcal{S}$ $\mathcal{C}$ [$*Loop* (*t1s* -> *t2s*) *es*] *t1s t2s vs* @ ($∗ *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*]⦈ ⇝ ⦇*vs*⦈
$\mathcal{S}$•$\mathcal{C}$ ⊢ [*Label n es0 vs*] : (*ts* -> *ts'*)
const-list vs
shows $\mathcal{S}$•$\mathcal{C}$ ⊢ *vs* : (*ts* -> *ts'*)
proof −
obtain *tls t2s* **where** *t2s-def*:(*ts'* = (*ts*@*t2s*))
($\mathcal{S}$•$\mathcal{C}$ ⊢ *es0* : (*tls* -> *t2s*))
($\mathcal{S}$•$\mathcal{C}$⦇*label* := [*tls*] @ (*label* $\mathcal{C}$)⦈ ⊢ *vs* : ([] -> *t2s*))
using *assms e-type-label*
by *fastforce*
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 *types-preserved-br-if*:
assumes ⦇[$*C ConstInt32 n*, $*Br-if i*]⦈ ⇝ ⦇*e*⦈

$\mathcal{S}\cdot\mathcal{C} \vdash [\$C\ ConstInt32\ n,\ \$Br\text{-}if\ i] : (ts\ \text{->}\ ts')$
$e = [\$Br\ i] \vee e = []$
shows $\mathcal{S}\cdot\mathcal{C} \vdash e : (ts\ \text{->}\ ts')$
proof −
have $\mathcal{C} \vdash [C\ ConstInt32\ n,\ Br\text{-}if\ i] : (ts\ \text{->}\ ts')$
using *unlift-b-e assms*(*2*)
by *fastforce*
then obtain ts'' **where** ts''-*def*:$\mathcal{C} \vdash [C\ ConstInt32\ n] : (ts\ \text{->}\ ts'')$ $\mathcal{C} \vdash [Br\text{-}if\ i]$: $(ts''\ \text{->}\ ts')$
using *b-e-type-comp*[*of* - [*C ConstInt32 n*] *Br-if i*]
by *fastforce*
then obtain *ts-c ts-b* **where** *ts-bc-def*:$i < length(label\ \mathcal{C})$
$ts'' = ts\text{-}c\ @\ ts\text{-}b\ @\ [T\text{-}i32]$
$ts' = ts\text{-}c\ @\ ts\text{-}b$
$(label\ \mathcal{C})!i = ts\text{-}b$
using *b-e-type-br-if*[*of* $\mathcal{C}$ *Br-if i* ts'' ts' *i*]
by *fastforce*
hence *ts-def*:$ts = ts\text{-}c\ @\ ts\text{-}b$
using ts''-*def*(*1*) *b-e-type-value*
by *fastforce*
show *?thesis*
using *assms*(*3*)
proof (*rule disjE*)
assume $e = [\$Br\ i]$
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* - [] $(ts\ \text{->}\ ts')$]
by *fastforce*
qed
qed

lemma *types-preserved-br-table*:
assumes $(\vert[\$C\ ConstInt32\ c,\ \$Br\text{-}table\ is\ i]\vert) \rightsquigarrow (\vert[\$Br\ i']\vert)$
$\mathcal{S}\cdot\mathcal{C} \vdash [\$C\ ConstInt32\ c,\ \$Br\text{-}table\ is\ i] : (ts\ \text{->}\ ts')$
$(i' = (is\ !\ nat\text{-}of\text{-}int\ c) \wedge length\ is > nat\text{-}of\text{-}int\ c) \vee i' = i$
shows $\mathcal{S}\cdot\mathcal{C} \vdash [\$Br\ i'] : (ts\ \text{->}\ ts')$
proof −
have $\mathcal{C} \vdash [C\ ConstInt32\ c,\ Br\text{-}table\ is\ i] : (ts\ \text{->}\ ts')$
using *unlift-b-e assms*(*2*)
by *fastforce*
then obtain ts'' **where** ts''-*def*:$\mathcal{C} \vdash [C\ ConstInt32\ c] : (ts\ \text{->}\ ts'')$ $\mathcal{C} \vdash [Br\text{-}table$ $is\ i] : (ts''\ \text{->}\ ts')$
using *b-e-type-comp*[*of* - [*C ConstInt32 c*] *Br-table is i*]
by *fastforce*

then obtain *ts-l ts-c* **where** *ts-c-def*:*list-all* ($\lambda i.\ i <$ *length*(*label* $\mathcal{C}$) $\wedge$ (*label* $\mathcal{C}$)!*i* = *ts-l*) (*is*@[*i*])
$ts'' = $ *ts-c* @ *ts-l*@[*T-i32*]
using *b-e-type-br-table*[*of* $\mathcal{C}$ *Br-table is i* ts'' ts']
by *fastforce*
hence *ts-def*:*ts* = *ts-c* @ *ts-l*
using ts''-*def*(*1*) *b-e-type-value*
by *fastforce*
have $\mathcal{C} \vdash$ [*Br* i'] : (*ts* -> 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*]⦈ ⇝ ⦇*es*⦈
$\mathcal{S}\cdot\mathcal{C} \vdash$ [*Local n i vs es*] : (*ts* -> ts')
const-list es
shows $\mathcal{S}\cdot\mathcal{C} \vdash$ *es*: (*ts* -> ts')
proof −
obtain *tls* **where** ($\mathcal{S}\cdot$((*s-inst* $\mathcal{S}$)!*i*)⦇*local* := (*local* ((*s-inst* $\mathcal{S}$)!*i*)) @ (*map typeof vs*), *return* := *Some tls*⦈ $\vdash$ *es* : ([] -> *tls*))
ts' = *ts* @ *tls*
using *e-type-local*[*OF assms*(*2*)]
by *blast+*
moreover
then have $\mathcal{S}\cdot\mathcal{C} \vdash$ *es* : ([] -> *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* : ([] -> *tvs*)
shows *tvs* = *map typeof vs*
using *assms*
proof (*induction ves arbitrary*: *vs tvs rule*: *List.rev-induct*)
case *Nil*
hence $\mathcal{C} \vdash$ [] : ([] -> *tvs*)
using *unlift-b-e*
by *auto*
thus *?case*

using *Nil*
by *auto*
next
case (*snoc a ves*)
obtain *vs′ v′* **where** *vs′-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*:$\mathcal{S}$•$\mathcal{C}$ ⊢ *ves*: ([] -> *tvs′*) $\mathcal{S}$•$\mathcal{C}$ ⊢ [*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*] $\mathcal{S}$•$\mathcal{C}$ ⊢ [*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* $\mathcal{C}$ *C v′* [] [*t*] *v′*]
by *fastforce*
ultimately
show *?case*
using *vs′-def t-def*
by *simp*
qed

lemma *types-preserved-call-indirect-Some*:
assumes $\mathcal{S}$•$\mathcal{C}$ ⊢ [$*C ConstInt32 c*, $*Call-indirect j*] : (*ts* -> *ts′*)
stab s i′ (*nat-of-int c*) = *Some cl*
stypes s i′ j = *tf*
cl-type cl = *tf*
store-typing s $\mathcal{S}$
i′ < *length* (*inst s*)
$\mathcal{C}$ = (*s-inst* $\mathcal{S}$! *i′*) (|*local* := *local* (*s-inst* $\mathcal{S}$! *i′*) @ *tvs*, *label* := *arb-labs*, *return* := *arb-return*|)
shows $\mathcal{S}$•$\mathcal{C}$ ⊢ [*Callcl cl*] : (*ts* -> *ts′*)
proof −
obtain *t1s t2s* **where** *tf-def*:*tf* = (*t1s* -> *t2s*)
using *tf*.*exhaust* **by** *blast*
obtain *ts″* **where** *ts″-def*:$\mathcal{C}$ ⊢ [*C ConstInt32 c*] : (*ts* -> *ts″*)
$\mathcal{C}$ ⊢ [*Call-indirect j*] : (*ts″* -> *ts′*)
using *e-type-comp*[*of* $\mathcal{S}$ $\mathcal{C}$ [$*C ConstInt32 c*] $*Call-indirect j ts ts′*]

```
          assms(1)
          unlift-b-e[of 𝒮 𝒞 [C ConstInt32 c]]
          unlift-b-e[of 𝒮 𝒞 [Call-indirect j]]
    by fastforce
  hence ts'' = ts@[(T-i32)]
    using b-e-type-value
    unfolding typeof-def
    by fastforce
  moreover
  have i' < length (s-inst 𝒮)
    using assms(5,6) store-typing-imp-inst-length-eq
    by fastforce
  hence stypes-eq:types-t (s-inst 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:j < length (types-t 𝒞)
                           ts'' = ts''a @ t1s @ [T-i32]
                           ts' = ts''a @ t2s
                           types-t 𝒞 ! j = (t1s -> 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 𝒮 cl tf'
    using assms(2,5,6) stab-typed-some-imp-cl-typed
    by blast
  hence cl-typing 𝒮 cl tf
    using assms(4)
    unfolding cl-typing.simps cl-type-def
    by auto
  hence 𝒮•𝒞 ⊢ [Callcl cl] : tf
    using e-typing-s-typing.intros(6) assms(6,7) ts''a-def(1)
    by fastforce
  ultimately
  show 𝒮•𝒞 ⊢ [Callcl cl] : (ts -> ts')
    using tf-def e-typing-s-typing.intros(3)
    by auto
qed

lemma types-preserved-call-indirect-None:
  assumes 𝒮•𝒞 ⊢ [$C ConstInt32 c, $Call-indirect j] : (ts -> ts')
  shows 𝒮•𝒞 ⊢ [Trap] : (ts -> ts')
  using e-typing-s-typing.intros(4)
  by blast

lemma types-preserved-callcl-native:
  assumes 𝒮•𝒞 ⊢ ves @ [Callcl cl] : (ts -> ts')
```

cl = *Func-native* *i* (*t1s* -> *t2s*) *tfs* *es*
ves = $$* *vs*
length *vs* = *n*
length *tfs* = *k*
length *t1s* = *n*
length *t2s* = *m*
n-zeros *tfs* = *zs*
store-typing *s* $\mathcal{S}$
shows $\mathcal{S}$•$\mathcal{C}$ ⊢ [*Local* *m* *i* (*vs* @ *zs*) [$*Block* ([] -> *t2s*) *es*]] : (*ts* -> *ts'*)
proof −
obtain *ts''* **where** *ts''-def*:$\mathcal{S}$•$\mathcal{C}$ ⊢ *ves* : (*ts* -> *ts''*) $\mathcal{S}$•$\mathcal{C}$ ⊢ [*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*
$\mathcal{S}$•$\mathcal{C}$ ⊢ *ves* : ([] -> *tvs*)
using *ts''-def*(*1*) *e-type-const-list*[*of* *ves* $\mathcal{S}$ $\mathcal{C}$ *ts* *ts''*] *assms*
by *fastforce*
obtain *ts-c* $\mathcal{C}'$ **where** *ts-c-def*:(*ts''* = *ts-c* @ *t1s*)
(*ts'* = *ts-c* @ *t2s*)
i < *length* (*s-inst* $\mathcal{S}$)
$\mathcal{C}'$ = ((*s-inst* $\mathcal{S}$)!*i*)
($\mathcal{C}'$⦇*local* := (*local* $\mathcal{C}'$) @ *t1s* @ *tfs*, *label* := ([*t2s*] @ (*label* $\mathcal{C}'$)), *return* := *Some* *t2s*⦈ ⊢ *es* : ([] -> *t2s*))
using *e-type-callcl-native*[*OF* *ts''-def*(*2*) *assms*(*2*)]
by *fastforce*
have *inst-typing* $\mathcal{S}$ (*inst* *s* ! *i*) (*s-inst* $\mathcal{S}$! *i*)
using *store-typing-imp-inst-length-eq*[*OF* *assms*(*9*)] *store-typing-imp-inst-typing*[*OF* *assms*(*9*)]
ts-c-def(*3*)
by *simp*
obtain $\mathcal{C}''$ **where** *c''-def*:$\mathcal{C}''$ = $\mathcal{C}'$⦇*local* := (*local* $\mathcal{C}'$) @ *t1s* @ *tfs*, *return* := *Some* *t2s*⦈
by *blast*
hence $\mathcal{C}''$⦇*label* := ([*t2s*] @ (*label* $\mathcal{C}''$))⦈ = $\mathcal{C}'$⦇*local* := (*local* $\mathcal{C}'$) @ *t1s* @ *tfs*, *label* := ([*t2s*] @ (*label* $\mathcal{C}'$)), *return* := *Some* *t2s*⦈
by *fastforce*
hence $\mathcal{S}$•$\mathcal{C}''$ ⊢ [$*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*(*1*)
by *simp-all*
have *1*:*tfs* = *map* *typeof* *zs*

using *n-zeros-typeof assms*(*8*)
by *auto*
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 $\mathcal{S}$•*Some t2s* ⊩-*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*)
by *fastforce*
qed

lemma *types-preserved-callcl-host-some*:
assumes $\mathcal{S}$•$\mathcal{C}$ ⊢ *ves* @ [*Callcl cl*] : (*ts* -> *ts'*)
cl = *Func-host* (*t1s* -> *t2s*) *f*
ves = $$∗ *vcs*
length vcs = *n*
length t1s = *n*
length t2s = *m*
host-apply s (*t1s* -> *t2s*) *f vcs hs* = *Some* (*s'*, *vcs'*)
store-typing s $\mathcal{S}$
shows $\mathcal{S}$•$\mathcal{C}$ ⊢ $$∗ *vcs'* : (*ts* -> *ts'*)
proof −
obtain *ts''* **where** *ts''-def*:$\mathcal{S}$•$\mathcal{C}$ ⊢ *ves* : (*ts* -> *ts''*) $\mathcal{S}$•$\mathcal{C}$ ⊢ [*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*
$\mathcal{S}$•$\mathcal{C}$ ⊢ *ves* : ([] -> *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 $\mathcal{S}$•$\mathcal{C}$ ⊢ $$∗ *vcs'* : ([] -> *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 $\mathcal{S}{\bullet}\mathcal{C} \vdash es\ @\ [e]\ @\ es' : (ts \text{ -> } ts')$
$\bigwedge tes\ tes'.\ ((\mathcal{S}{\bullet}\mathcal{C} \vdash [e] : (tes \text{ -> } tes')) \Longrightarrow (\mathcal{S}{\bullet}\mathcal{C} \vdash [e'] : (tes \text{ -> } tes')))$
shows $\mathcal{S}{\bullet}\mathcal{C} \vdash es\ @\ [e']\ @\ es' : (ts \text{ -> } ts')$
proof −
obtain ts'' **where** $\mathcal{S}{\bullet}\mathcal{C} \vdash es : (ts \text{ -> } ts'')$
$\mathcal{S}{\bullet}\mathcal{C} \vdash [e]\ @\ es' : (ts'' \text{ -> } ts')$
using *e-type-comp-conc1[of - - es [e] @ es'] assms(1)*
by *fastforce*
moreover
then obtain ts''' **where** $\mathcal{S}{\bullet}\mathcal{C} \vdash [e] : (ts'' \text{ -> } ts''')$ $\mathcal{S}{\bullet}\mathcal{C} \vdash es' : (ts''' \text{ -> } ts')$
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
$\mathcal{S}{\bullet}\mathcal{C} \vdash LI : (ts \text{ -> } ts')$
const-list vs
shows $\mathcal{S}{\bullet}\mathcal{C}' \vdash vs : ([] \text{ -> } tcs)$
using *assms*
proof (*induction i arbitrary*: $ts\ ts'\ lholed\ \mathcal{C}\ LI\ \mathcal{C}'$)
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** $\mathcal{S}{\bullet}\mathcal{C} \vdash vs' : (ts \text{ -> } ts'')$
$\mathcal{S}{\bullet}\mathcal{C} \vdash (vs\ @\ [\$Return]) : (ts'' \text{ -> } ts''')$
$\mathcal{S}{\bullet}\mathcal{C} \vdash es' : (ts''' \text{ -> } ts')$
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}{\bullet}\mathcal{C} \vdash vs : (ts'' \text{ -> } ts\text{-}b)$ $\mathcal{S}{\bullet}\mathcal{C} \vdash [\$Return] : (ts\text{-}b$
$\text{-> } ts''')$
using *e-type-comp-conc1*
by *fastforce*

then obtain *ts-c* **where** *ts-c-def*:*ts-b* = *ts-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* -> *ts'''*]
by *fastforce*
obtain *tcs'* **where** *ts-b* = *ts''* @ *tcs'* *length vs* = *length tcs'* $\mathcal{S}$•$\mathcal{C}'$ ⊢ *vs* : ([] -> *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** $\mathcal{S}$•$\mathcal{C}$ ⊢ [*Label n les LK*] : (*ts''* -> *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*
$\mathcal{S}$•$\mathcal{C}$ ⊢ *les* : (*tls* -> *t2s*)
$\mathcal{S}$•$\mathcal{C}$(|*label* := [*tls*] @ *label* $\mathcal{C}$|) ⊢ *LK* : ([] -> *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*]|) ⇝ (|*ves*|)
$\mathcal{S}$•$\mathcal{C}$ ⊢ [*Local n i vls LI*] : (*ts* -> *ts'*)
const-list ves
length ves = *n*
Lfilled j lholed (*ves* @ [$*Return*]) *LI*
shows $\mathcal{S}$•$\mathcal{C}$ ⊢ *ves* : (*ts* -> *ts'*)
proof −
obtain *tls* $\mathcal{C}'$ **where** *l-def*:*i* < *length* (*s-inst* $\mathcal{S}$)
$\mathcal{C}'$ = ((*s-inst* $\mathcal{S}$)!*i*)(|*local* := (*local* ((*s-inst* $\mathcal{S}$)!*i*)) @ (*map typeof vls*), *return* := *Some tls*|)
$\mathcal{S}$•$\mathcal{C}'$ ⊢ *LI* : ([] -> *tls*)
ts' = *ts* @ *tls*
length tls = *n*
using *e-type-local*[*OF assms*(*2*)]
by *blast*

hence $\mathcal{S}\cdot\mathcal{C} \vdash$ *ves* : ([] -> *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* -> *ts*′)
const-list vs
shows $\mathcal{S}\cdot\mathcal{C}' \vdash$ *vs* : ([] -> *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* -> *ts*′′)
$\mathcal{S}\cdot\mathcal{C} \vdash$ (*vs* @ [$*Br* (*0*+*k*)]) : (*ts*′′ -> *ts*′′′)
$\mathcal{S}\cdot\mathcal{C} \vdash$ *es*′ : (*ts*′′′ -> *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*′′ -> *ts-b*) $\mathcal{S}\cdot\mathcal{C} \vdash$ [$*Br* (*0*+*k*)] : (*ts-b* -> *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** $\mathcal{S}\bullet\mathcal{C} \vdash [Label\ n\ les\ LK] : (ts'' \text{ -> } ts''')$
  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** $\mathcal{S}\bullet\mathcal{C}'' \vdash LK : ([] \text{ -> } ts'''')$ $\mathcal{C}'' = \mathcal{C}(\!|label := [lts]$ @ $(label\ \mathcal{C})|\!)$
  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 $(\!|[Label\ n\ es0\ LI]|\!) \rightsquigarrow (\!|vs\ @\ es0|\!)$
  $\mathcal{S}\bullet\mathcal{C} \vdash [Label\ n\ es0\ LI] : (ts \text{ -> } ts')$
  const-list vs
  length vs = *n*
  Lfilled i lholed (*vs* @ [$*Br i*]) *LI*
shows $\mathcal{S}\bullet\mathcal{C} \vdash (vs\ @\ es0) : (ts \text{ -> } ts')$
proof −
obtain *tls t2s* $\mathcal{C}'$ **where** *l-def*:(ts' = (*ts*@*t2s*))
  ($\mathcal{S}\bullet\mathcal{C} \vdash es0 : (tls \text{ -> } t2s)$)
  $\mathcal{C}' = \mathcal{C}(\!|label := [tls]\ @\ (label\ \mathcal{C})|\!)$
  length (*label* $\mathcal{C}'$) > *0*
  (*label* $\mathcal{C}'$)!*0* = *tls*
  length tls = *n*
  ($\mathcal{S}\bullet\mathcal{C}(\!|label := [tls]\ @\ (label\ \mathcal{C})|\!) \vdash LI : ([] \text{ -> } t2s)$)
  using *e-type-label*[*of* $\mathcal{S}$ $\mathcal{C}$ *n es0 LI ts ts'*] *assms*(*2*)
  by *fastforce*
hence $\mathcal{S}\bullet\mathcal{C} \vdash vs : ([] \text{ -> } tls)$
  using *assms*(*3*−*5*) *type-const-br*[*of i lholed vs 0 LI* $\mathcal{C}'$ *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*|) ⇝ (|*es*′|)
store-typing s $\mathcal{S}$
$\mathcal{S}$•$\mathcal{C}$ ⊢ *es* : (*ts* -> *ts*′)
shows $\mathcal{S}$•$\mathcal{C}$ ⊢ *es*′ : (*ts* -> *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 iop c1 c2 c*)
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 iop c1 c2 c*)
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 fop c1 c2 c)
  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 fop c1 c2 c)
  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 t2 sx v')
  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 unreachable
  then show ?thesis
    using e-typing-s-typing.intros(4)
    by simp
next
  case nop
  then have 𝒞 ⊢ [Nop] : (ts -> 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 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 i')
  then show ?thesis
```

using *assms*(*1*, *3*) *types-preserved-br-table*
by *fastforce*
next
case (*br-table-length is′ c 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*⦈ ⇝ ⦇*es′*⦈
store-typing s $\mathcal{S}$
$\mathcal{S}$•*None* ⊩-*i vs*;*es* : *ts*
shows $\mathcal{S}$•*None* ⊩-*i vs*;*es′* : *ts*
proof −
have *i* < (*length* (*s-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-inst* $\mathcal{S}$)!*i*)⦇*local* := (*local* ((*s-inst* $\mathcal{S}$)!*i*) @ *tvs*), *return* := *None*⦈ $\mathcal{S}$•$\mathcal{C}$ ⊢ *es* : ([] -> *ts*)
using *assms*(*3*)
unfolding *s-typing.simps*
by *blast*
have $\mathcal{S}$•$\mathcal{C}$ ⊢ *es′* : ([] -> *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 k*, \$*C v*, \$*Store t tp a off*] : (*ts* -> *ts'*)
shows $\mathcal{S}\cdot\mathcal{C} \vdash$ [] : (*ts* -> *ts'*)
types-agree t v
proof −
obtain *ts'' ts'''* **where** *ts-def*:$\mathcal{S}\cdot\mathcal{C} \vdash$ [\$*C ConstInt32 k*] : (*ts* -> *ts''*)
$\mathcal{S}\cdot\mathcal{C} \vdash$ [\$*C v*] : (*ts''* -> *ts'''*)
$\mathcal{S}\cdot\mathcal{C} \vdash$ [\$*Store t tp a off*] : (*ts'''* -> *ts'*)
using *assms e-type-comp-conc2*[*of* $\mathcal{S}$ $\mathcal{C}$ [\$*C ConstInt32 k*] [\$*C v*] [\$*Store t tp a off*]]
by *fastforce*
then have *ts''* = *ts*@[(*T-i32*)]
using *b-e-type-value*[*of* $\mathcal{C}$ *C ConstInt32 k ts ts''*]
unlift-b-e[*of* $\mathcal{S}$ $\mathcal{C}$ [*C* (*ConstInt32 k*)] (*ts* -> *ts''*)]
unfolding *typeof-def*
by *fastforce*
hence *ts'''* = *ts*@[(*T-i32*), (*typeof v*)]
using *ts-def*(*2*) *b-e-type-value*[*of* $\mathcal{C}$ *C v ts'' ts'''*]
unlift-b-e[*of* $\mathcal{S}$ $\mathcal{C}$ [*C v*] (*ts''* -> *ts'''*)]
by *fastforce*
hence *ts* = *ts'* *types-agree t v*
using *ts-def*(*3*) *b-e-type-store*[*of* $\mathcal{C}$ *Store t tp a off ts''' ts'*]
unlift-b-e[*of* $\mathcal{S}$ $\mathcal{C}$ [*Store t tp a off*] (*ts'''* -> *ts'*)]
unfolding *types-agree-def*
by *fastforce+*
thus $\mathcal{S}\cdot\mathcal{C} \vdash$ [] : (*ts* -> *ts'*) *types-agree t v*
using *b-e-type-empty*[*of* $\mathcal{C}$ *ts ts'*] *e-typing-s-typing.intros*(*1*)
by *fastforce+*
qed

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

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

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

lemma *types-preserved-load*:
  assumes $\mathcal{S}\cdot\mathcal{C} \vdash [\$C\ \mathit{ConstInt32}\ k,\ \$\mathit{Load}\ t\ tp\ a\ \mathit{off}] : (ts\ \text{->}\ ts')$
      $\mathit{typeof}\ v = t$

shows $\mathcal{S}\bullet\mathcal{C} \vdash [\$C\ v] : (ts \text{ -> } ts')$
proof −
obtain ts'' **where** $ts''\text{-}def$:$\mathcal{S}\bullet\mathcal{C} \vdash [\$C\ ConstInt32\ k] : (ts \text{ -> } ts'')$
$\mathcal{S}\bullet\mathcal{C} \vdash [\$Load\ t\ tp\ a\ off] : (ts'' \text{ -> } ts')$
using *e-type-comp assms*
by (*metis append-butlast-last-id butlast.simps*(*2*) *last.simps list.distinct*(*1*))
hence $ts'' = ts@[(T\text{-}i32)]$
using *b-e-type-value unlift-b-e*[*of* $\mathcal{S}$ $\mathcal{C}$ [*C ConstInt32 k*]]
unfolding *typeof-def*
by *fastforce*
hence *ts-def*:$ts' = ts@[t]$ *load-store-t-bounds a* (*option-projl tp*) *t*
using *ts''-def*(*2*) *b-e-type-load unlift-b-e*[*of* $\mathcal{S}$ $\mathcal{C}$ [*Load t tp a off*]]
by *fastforce+*
moreover
hence $\mathcal{C} \vdash [C\ v] : (ts \text{ -> } ts@[t])$
using *assms*(*2*) *b-e-typing.const b-e-typing.weakening*
by *fastforce*
ultimately
show $\mathcal{S}\bullet\mathcal{C} \vdash [\$C\ v] : (ts \text{ -> } ts')$
using *e-typing-s-typing.intros*(*1*)
by *fastforce*
qed

lemma *types-preserved-get-local*:
assumes $\mathcal{S}\bullet\mathcal{C} \vdash [\$Get\text{-}local\ i] : (ts \text{ -> } ts')$
length vi = *i*
(*local* $\mathcal{C}$) = *map typeof* (*vi* @ [*v*] @ *vs*)
shows $\mathcal{S}\bullet\mathcal{C} \vdash [\$C\ v] : (ts \text{ -> } ts')$
proof −
have (*local* $\mathcal{C}$)!*i* = *typeof v*
using *assms*(*2*,*3*)
by (*metis* (*no-types*, *opaque-lifting*) *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-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 $\mathcal{S}\bullet\mathcal{C} \vdash [\$C\ v', \$Set\text{-}local\ i] : (ts \text{ -> } ts')$
length vi = *i*
(*local* $\mathcal{C}$) = *map typeof* (*vi* @ [*v*] @ *vs*)
shows ($\mathcal{S}\bullet\mathcal{C} \vdash [] : (ts \text{ -> } ts')$) ∧ *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*, *opaque-lifting*) *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* -> *ts''*)
$\mathcal{S}\cdot\mathcal{C} \vdash$ [$*Set-local i*] : (*ts''* -> *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-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* $\mathcal{C}$! *j*)
$\mathcal{S}\cdot\mathcal{C} \vdash$ [$*Get-global j*] : (*ts* -> *ts'*)
shows $\mathcal{S}\cdot\mathcal{C} \vdash$ [$*C sglob-val s i j*] : (*ts* -> *ts'*)
proof −
have *ts'* = *ts*@[*tg-t* (*global* $\mathcal{C}$! *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* $\mathcal{C}$ *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* -> *ts'*)
$\bigwedge$*arb-labs ts ts'*.
$\mathcal{S}\cdot(\mathcal{C}$(|*label* := *arb-labs*@(*label* $\mathcal{C}$)|)) $\vdash$ *es* : (*ts* -> *ts'*)
$\Longrightarrow$ $\mathcal{S}\cdot(\mathcal{C}$(|*label* := *arb-labs*@(*label* $\mathcal{C}$)|)) $\vdash$ *es'* : (*ts* -> *ts'*)
shows ($\mathcal{S}\cdot\mathcal{C} \vdash$ *les'* : (*ts* -> *ts'*))
using *assms*
proof (*induction arbitrary*: *ts ts' es'* $\mathcal{C}$ *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* -> *ts''*)
$\mathcal{S}\cdot\mathcal{C} \vdash$ *es* : (*ts''* -> *ts'''*)
$\mathcal{S}\cdot\mathcal{C} \vdash$ *es'* : (*ts'''* -> *ts'*)
using *e-type-comp-conc2 L0(4)*
by *blast*
moreover

hence $(\mathcal{S}\cdot\mathcal{C} \vdash es'' : (ts'' \text{ -> } ts'''))$
using *L0*(*5*)[*of* [] *ts'' ts'''*]
by *fastforce*
ultimately
have $(\mathcal{S}\cdot\mathcal{C} \vdash vs \ @ \ es'' \ @ \ es' : (ts \text{ -> } ts'))$
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'''* $\mathcal{C}$ *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*:$\mathcal{S}\cdot\mathcal{C} \vdash vs : (t1s \text{ -> } ts')$
$\mathcal{S}\cdot\mathcal{C} \vdash [Label\ n\ es'\ lfilledk] : (ts' \text{ -> } ts'')$
$\mathcal{S}\cdot\mathcal{C} \vdash es'' : (ts'' \text{ -> } t2s)$
using *e-type-comp-conc2*[*OF LN*(*6*)]
by *blast*
obtain *tls ts-c* $\mathcal{C}$*-int* **where** *int-def*: $ts'' = ts' \ @ \ ts\text{-}c$
$length\ tls = n$
$\mathcal{S}\cdot\mathcal{C} \vdash es' : (tls \text{ -> } ts\text{-}c)$
$\mathcal{C}\text{-}int = \mathcal{C}(\!|label := [tls] \ @ \ label\ \mathcal{C}|\!)$
$\mathcal{S}\cdot\mathcal{C}\text{-}int \vdash lfilledk : ([] \text{ -> } ts\text{-}c)$
using *e-type-label*[*OF lab-def*(*2*)]
by *blast*
have $(\bigwedge\mathcal{C}'\ arb\text{-}labs'\ ts\ ts'.$
$\mathcal{C}' = \mathcal{C}\text{-}int(\!|label := arb\text{-}labs' \ @ \ label\ \mathcal{C}\text{-}int|\!) \Longrightarrow$
$\mathcal{S}\cdot\mathcal{C}' \vdash es : (ts \text{ -> } ts') \Longrightarrow$
$(\mathcal{S}\cdot\mathcal{C}' \vdash es''' : (ts \text{ -> } ts')))$
proof –
fix $\mathcal{C}''$ *arb-labs'' tts tts'*
assume $\mathcal{C}'' = \mathcal{C}\text{-}int(\!|label := arb\text{-}labs'' \ @ \ label\ \mathcal{C}\text{-}int|\!)$
$\mathcal{S}\cdot\mathcal{C}'' \vdash es : (tts \text{ -> } tts')$
thus $(\mathcal{S}\cdot\mathcal{C}'' \vdash es''' : (tts \text{ -> } tts'))$
using *LN*(*7*)[*of arb-labs''* @ [*tls*] *tts tts'*] *int-def*(*4*)
by *fastforce*
qed
hence $(\mathcal{S}\cdot\mathcal{C}\text{-}int \vdash lfilledk' : ([] \text{ -> } ts\text{-}c))$
using *LN*(*4*)[*OF l'-def*(*1*) *int-def*(*5*)]
by *fastforce*
hence $(\mathcal{S}\cdot\mathcal{C} \vdash [Label\ n\ es'\ lfilledk'] : (ts' \text{ -> } ts''))$
using *int-def e-typing-s-typing.intros*(*3*,*7*)
by (*metis append.right-neutral*)
thus *?case*
using *lab-def e-type-comp-conc l'-def*(*2*)
by *blast*

qed

lemma *types-preserved-e1*:
  assumes $\llparenthesis s;vs;es\rrparenthesis \leadsto$-*i* $\llparenthesis s';vs';es'\rrparenthesis$
    store-typing s $\mathcal{S}$
    tvs = *map typeof vs*
    i < *length* (*inst s*)
    $\mathcal{C}$ = ((*s-inst* $\mathcal{S}$)!*i*)$\llparenthesis$*local* := (*local* ((*s-inst* $\mathcal{S}$)!*i*) @ *tvs*), *label* := *arb-labs*, *return* := *arb-return*$\rrparenthesis$
    $\mathcal{S}$•$\mathcal{C}$ ⊢ *es* : (*ts* -> *ts'*)
  shows ($\mathcal{S}$•$\mathcal{C}$ ⊢ *es'* : (*ts* -> *ts'*)) ∧ (*map typeof vs* = *map typeof vs'*)
  using *assms*
proof (*induction arbitrary*: *tvs* $\mathcal{C}$ *ts ts' arb-labs arb-return rule*: *reduce.induct*)
  case (*basic e e' s vs i*)
  then show *?case*
    using *types-preserved-b-e1*[*OF basic*(*1*,*2*)]
    by *fastforce*
next
  case (*call s vs j i*)
  obtain *ts'' tf1 tf2* **where** *l-func-t*: *length* (*func-t* $\mathcal{C}$) > *j*
    ts = *ts''*@*tf1*
    ts' = *ts''*@*tf2*
    ((*func-t* $\mathcal{C}$)!*j*) = (*tf1* -> *tf2*)
    using *b-e-type-call*[*of* $\mathcal{C}$ *Call j ts ts' j*] *call*(*5*)
      unlift-b-e[*of - -* [*Call j*] (*ts* -> *ts'*)]
    by *fastforce*
  have *i* < *length* (*s-inst* $\mathcal{S}$)
    using *call*(*3*) *store-typing-imp-inst-length-eq*[*OF call*(*1*)]
    by *simp*
  moreover
  have *j* < *length* (*func-t* (*s-inst* $\mathcal{S}$! *i*))
    using *l-func-t*(*1*) *call*(*4*)
    by *simp*
  ultimately
  have *cl-typing* $\mathcal{S}$ (*sfunc s i j*) (*tf1* -> *tf2*)
    using *store-typing-imp-func-agree*[*OF call*(*1*)] *l-func-t*(*4*) *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*)
  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 s i c cl j vs*)
  thus *?case*

using *e-typing-s-typing.intros(4)*
by *blast*
next
case (*callcl-native cl j t1s t2s ts es ves vcs n k m zs s vs i*)
thus *?case*
using *types-preserved-callcl-native*
by *fastforce*
next
case (*callcl-host-Some cl t1s t2s f ves vcs n m s hs s′ vcs′ vs i*)
thus *?case*
using *types-preserved-callcl-host-some*
by *fastforce*
next
case (*callcl-host-None cl t1s t2s f ves vcs n m s hs 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* $\mathcal{S}$)
unfolding *list-all2-conv-all-nth store-typing.simps*
by *fastforce*
then have *local* $\mathcal{C}$ = *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* $\mathcal{S}$)
unfolding *list-all2-conv-all-nth store-typing.simps*
by *fastforce*
hence *local* $\mathcal{C}$ = *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* $\mathcal{C}$) $> j$
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* $\mathcal{C}$)!*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* $\mathcal{C}$! *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 j m k off t bs vs a*)
then show *?case*
using *types-preserved-load*(*1*) *wasm-deserialise-type*
by *blast*
next
case (*load-None s i j m k off t vs a*)
then show *?case*
using *e-typing-s-typing.intros*(*4*)
by *blast*
next
case (*load-packed-Some s i j m sx k off tp bs vs t a*)
then show *?case*
using *types-preserved-load*(*1*) *wasm-deserialise-type*
by *blast*
next
case (*load-packed-None s i j m sx k off tp vs t a*)
then show *?case*
using *e-typing-s-typing.intros*(*4*)
by *blast*
next
case (*store-Some t v s i j m k off mem′ vs a*)
then show *?case*
using *types-preserved-store*
by *blast*
next
case (*store-None t v s i j m k off vs a*)
then show *?case*
using *e-typing-s-typing.intros*(*4*)
by *blast*
next
case (*store-packed-Some t v s i j m k off tp mem′ vs a*)
then show *?case*
using *types-preserved-store*
by *blast*
next
case (*store-packed-None t v s i j m k off tp vs a*)
then show *?case*

using *e-typing-s-typing.intros*(*4*)
by *blast*
next
case (*current-memory s i j m n vs*)
then show *?case*
using *types-preserved-current-memory*
by *fastforce*
next
case (*grow-memory s i j m n c mem′ vs*)
then show *?case*
using *types-preserved-grow-memory*
by *fastforce*
next
case (*grow-memory-fail s i j m n vs c*)
thus *?case*
using *types-preserved-grow-memory*
by *blast*
next
case (*label s vs es i s′ vs′ es′ k lholed les les′*)
{
fix $\mathcal{C}'$ *arb-labs′ ts ts′*
assume *local-assms*:$\mathcal{C}' = \mathcal{C}(\!|$*label* := *arb-labs′*@(*label* $\mathcal{C}$), *return* := (*return* $\mathcal{C}$)$|\!)$
hence ($\mathcal{S}$•$\mathcal{C}'$ ⊢ *es* : (*ts* -> *ts′*)) ⟹ ($\mathcal{S}$•$\mathcal{C}'$ ⊢ *es′* : (*ts* -> *ts′*)) ∧ *map typeof vs* = *map typeof vs′*
using *label*(*4*)[*OF label*(*5*,*6*,*7*)] *label*(*8*)
by *fastforce*
hence ($\mathcal{S}$•$\mathcal{C}(\!|$*label* := *arb-labs′*@(*label* $\mathcal{C}$)$|\!)$ ⊢ *es* : (*ts* -> *ts′*))
⟹ ($\mathcal{S}$•$\mathcal{C}(\!|$*label* := *arb-labs′*@(*label* $\mathcal{C}$)$|\!)$ ⊢ *es′* : (*ts* -> *ts′*)) ∧
map typeof vs = *map typeof vs′*
using *local-assms*
by *simp*
}
hence ⋀*arb-labs′ ts ts′*. $\mathcal{S}$•$\mathcal{C}(\!|$*label* := *arb-labs′*@(*label* $\mathcal{C}$)$|\!)$ ⊢ *es* : (*ts* -> *ts′*)
⟹ ($\mathcal{S}$•$\mathcal{C}(\!|$*label* := *arb-labs′*@(*label* $\mathcal{C}$)$|\!)$ ⊢ *es′* : (*ts* -> *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 i s′ vs′ es′ vs n j*)
obtain $\mathcal{C}'$ *tls* **where** *es-def*:*i* < *length* (*s-inst* $\mathcal{S}$)
length tls = *n*
$\mathcal{C}'$ = (*s-inst* $\mathcal{S}$! *i*) $(\!|$*local* := *local*(*s-inst* $\mathcal{S}$! *i*) @ *map typeof vls*, *label* := *label* (*s-inst* $\mathcal{S}$! *i*), *return* := *Some tls*$|\!)$
$\mathcal{S}$•$\mathcal{C}'$ ⊢ *es* : ([] -> *tls*)
ts′ = *ts* @ *tls*
using *e-type-local*[*OF local*(*7*)]

by *fastforce*
moreover
obtain *ts''* **where** *ts' = ts@ts''* (𝒮•(*Some ts''*) ⊩-*i vls*;*es* : *ts''*)
using *e-type-local-shallow local*(*7*)
by *fastforce*
moreover
have *inst-typing* 𝒮 ((*inst s*)!*i*) ((*s-inst* 𝒮)!*i*) *i* < *length* (*inst s*)
using *local es-def*(*1*)
unfolding *store-typing.simps list-all2-conv-all-nth*
by *fastforce+*
ultimately
have 𝒮•𝒞' ⊢ *es'* : ([] -> *tls*) *map typeof vls = map typeof vs'*
using *local*(*2*)[*OF local*(*3*) - - - *es-def*(*4*), *of map typeof vls Some tls label* (*s-inst* 𝒮 ! *i*)]
by *fastforce+*
hence 𝒮•(*Some tls*) ⊩- *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*⦈ ↝-*i* ⦇*s'*;*vs'*;*es'*⦈
store-typing s 𝒮
𝒮•*None* ⊩-*i vs*;*es* : *ts*
shows 𝒮•*None* ⊩-*i vs'*;*es'* : *ts*
using *assms*
proof −
have *i* < (*length* (*s-inst* 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* 𝒞 **where** *defs*: *tvs = map typeof vs*
𝒞 = ((*s-inst* 𝒮)!*i*)⦇*local* := (*local* ((*s-inst* 𝒮)!*i*) @ *tvs*), *label* := (*label* ((*s-inst* 𝒮)!*i*)), *return* := *None*⦈
𝒮•𝒞 ⊢ *es* : ([] -> *ts*)
using *assms*(*3*)
unfolding *s-typing.simps*
by *fastforce*
have (𝒮•𝒞 ⊢ *es'* : ([] -> *ts*)) ∧ (*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 $\neg(\!|s;vs;es|\!) \leadsto\text{-}\ i\ (\!|s';vs';es'|\!)$
proof −
{
assume $(\!|s;vs;es|\!) \leadsto\text{-}\ i\ (\!|s';vs';es'|\!)$
hence *False*
using *assms*
proof (*induction rule*: *reduce.induct*)
case (*basic e e′ s vs i*)
thus *?thesis*
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 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|) ⇝- i (|s';vs';es'|)
proof -
  {
    assume (|s;vs;es|) ⇝- i (|s';vs';es'|)
    hence False
      using assms
    proof (induction rule: reduce.induct)
      case (basic e 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 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|) ⇝- i (|s';vs';es'|)
proof -
  {
    assume (|s;vs;es|) ⇝- i (|s';vs';es'|)
    hence False
      using assms
    proof (induction rule: reduce.induct)
      case (basic e e' s vs i)
      thus ?case
      by (induction rule: reduce-simple.induct) auto
    next
```

    case (*label s vs es 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*
qed

lemma *terminal-no-progress*:
  assumes *const-list es* ∨ *es* = [*Trap*]
  shows ¬⦇*s*;*vs*;*es*⦈ ⇝- *i* ⦇*s′*;*vs′*;*es′*⦈
  using *const-list-no-progress trap-no-progress assms*
  by *blast*

lemma *progress-L0*:
  assumes ⦇*s*;*vs*;*es*⦈ ⇝- *i* ⦇*s′*;*vs′*;*es′*⦈
          const-list cs
  shows ⦇*s*;*vs*;*cs*@*es*@*es-c*⦈ ⇝- *i* ⦇*s′*;*vs′*;*cs*@*es′*@*es-c*⦈
proof −
  have ⋀*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*⦈ ⇝- *i* ⦇*s′*;*vs′*;*es′*⦈
          const-list cs
  shows ⦇*s*;*vs*;*cs*@*es*⦈ ⇝- *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⦈ $\rightsquigarrow$- 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*)
$\mathcal{S}$•$\mathcal{C}$ ⊢ *es* : ([] -> *ts*)
(*label* $\mathcal{C}$)!*k* = *tvs*
shows ∃ *lholed′ vs* $\mathcal{C}'$. (*Lfilled j lholed′* (*vs*@[\$*Br* (*j*+*k*)]) *es*)
∧ ($\mathcal{S}$•$\mathcal{C}'$ ⊢ *vs* : ([] -> *tvs*))
∧ *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*:$\mathcal{S}$•$\mathcal{C}$ ⊢ *vs* : ([] -> *ts′*)
$\mathcal{S}$•$\mathcal{C}$ ⊢ [\$*Br k*] : (*ts′* -> *ts″*)
$\mathcal{S}$•$\mathcal{C}$ ⊢ *es′* : (*ts″* -> *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*:$\mathcal{S}$•$\mathcal{C}$ ⊢ *vs1* : ([] -> *ts-c*)
$\mathcal{S}$•$\mathcal{C}$ ⊢ *vs2* : (*ts-c* -> (*ts-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 $\mathcal{S}$•$\mathcal{C}$ ⊢ *vs2* : ([] -> *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*:$\mathcal{S}$•$\mathcal{C}$ ⊢ *vs* : ([] -> *t1s*)
$\mathcal{S}$•$\mathcal{C}$ ⊢ [*Label n es′ lfilledk*] : (*t1s* -> *t2s*)
$\mathcal{S}$•$\mathcal{C}$ ⊢ *es″* : (*t2s* -> *ts*)
using *e-type-comp-conc2*[*OF LN*(*5*)]
by *fastforce*

obtain *ts′ ts-l* **where** *ts-l-def*:$\mathcal{S}$•$\mathcal{C}$(|*label* := [*ts′*] @ *label* $\mathcal{C}$|) ⊢ *lfilledk* : ([] -> *ts-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*
$\mathcal{S}$•$\mathcal{C}$′ ⊢ *vs′* : ([] -> *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*)
$\mathcal{S}$•$\mathcal{C}$ ⊢ *es* : ([] -> *ts*)
(*return* $\mathcal{C}$) = *Some tvs*
shows ∃ *lholed′ vs $\mathcal{C}$′*. (*Lfilled j lholed′* (*vs*@[$*Return*]) *es*)
∧ ($\mathcal{S}$•$\mathcal{C}$′ ⊢ *vs* : ([] -> *tvs*))
∧ *const-list vs*
using *assms*
proof (*induction* [$*Return*] *es arbitrary*: *k* $\mathcal{C}$ *ts rule*: *Lfilled.induct*)
case (*L0 vs lholed es′*)
obtain *ts′ ts″* **where** *ts-def*:$\mathcal{S}$•$\mathcal{C}$ ⊢ *vs* : ([] -> *ts′*)
$\mathcal{S}$•$\mathcal{C}$ ⊢ [$*Return*] : (*ts′* -> *ts″*)
$\mathcal{S}$•$\mathcal{C}$ ⊢ *es′* : (*ts″* -> *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*:$\mathcal{S}$•$\mathcal{C}$ ⊢ *vs1* : ([] -> *ts-c*)
$\mathcal{S}$•$\mathcal{C}$ ⊢ *vs2* : (*ts-c* -> (*ts-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 $\mathcal{S}$•$\mathcal{C}$ ⊢ *vs2* : ([] -> *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}$•$\mathcal{C}$ ⊢ *vs* : ([] -> *t1s*)
$\mathcal{S}$•$\mathcal{C}$ ⊢ [*Label n es′ lfilledk*] : (*t1s* -> *t2s*)
$\mathcal{S}$•$\mathcal{C}$ ⊢ *es″* : (*t2s* -> *ts*)

using *e-type-comp-conc2*[*OF LN*(*5*)]
by *fastforce*
obtain *ts′ ts-l* **where** *ts-l-def*:$\mathcal{S}$•$\mathcal{C}$(|*label* := [*ts′*] @ *label* $\mathcal{C}$|) ⊢ *lfilledk* : ([] -> *ts-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*
$\mathcal{S}$•$\mathcal{C}'$ ⊢ *vs′* : ([] -> *tvs*)
const-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*)
$\mathcal{S}$•$\mathcal{C}$ ⊢ *es* : (*ts* -> *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*:$\mathcal{S}$•$\mathcal{C}$ ⊢ *vs* : (*ts* -> *ts″*)
$\mathcal{S}$•$\mathcal{C}$ ⊢ [$*Br k*] : (*ts″* -> *ts‴*)
$\mathcal{S}$•$\mathcal{C}$ ⊢ *es′* : (*ts‴* -> *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*:$\mathcal{S}$•$\mathcal{C}$ ⊢ *vs* : (*ts* -> *t1s*)
$\mathcal{S}$•$\mathcal{C}$ ⊢ [*Label n es′ lfilledk*] : (*t1s* -> *t2s*)
$\mathcal{S}$•$\mathcal{C}$ ⊢ *es″* : (*t2s* -> *ts′*)
using *e-type-comp-conc2*[*OF LN*(*5*)]
by *fastforce*
obtain *ts″ ts-l* **where** *ts-l-def*:$\mathcal{S}$•$\mathcal{C}$(|*label* := [*ts″*] @ *label* $\mathcal{C}$|) ⊢ *lfilledk* : ([] -> *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 ∃ *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 ∃ *v*. *cs* = [$*C v*]
  using *e-type-const-unwrap assms*
  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*
          $\mathcal{S}$•$\mathcal{C}$ ⊢ *cs* : ([] -> [(*T-i32*)])
  shows ∃ *c*. *cs* = [$*C ConstInt32 c*]
proof −
  obtain *v* **where** *cs* = [$*C v*]
    using *const-of-const-list assms*(*1*) *e-type-const-list*[*OF assms*]
    by *fastforce*
  moreover
  hence $\mathcal{C}$ ⊢ [*C v*] : ([] -> [(*T-i32*)])
    using *assms*(*2*) *unlift-b-e*
    by *fastforce*
  hence ∃ *c*. *v* = *ConstInt32 c*
  proof (*induction* [*C v*] ([] -> [(*T-i32*)]) *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*
          $\mathcal{S}$•$\mathcal{C}$ ⊢ *cs* : ([] -> [(*T-i64*)])
  shows ∃ *c*. *cs* = [$*C ConstInt64 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 $\mathcal{C} \vdash [C\ v] : ([] \text{ -> } [(T\text{-}i64)])$
    using *assms(2) unlift-b-e*
    by *fastforce*
  hence $\exists\, c.\ v = ConstInt64\ c$
  proof (*induction* $[C\ v]$ $([] \text{ -> } [(T\text{-}i64)])$ *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*
        $\mathcal{S}\cdot\mathcal{C} \vdash cs : ([] \text{ -> } [T\text{-}f32])$
  shows $\exists\, c.\ cs = [\$C\ ConstFloat32\ c]$
proof −
  obtain v **where** $cs = [\$C\ v]$
    using *const-of-const-list assms(1) e-type-const-list[OF assms]*
    by *fastforce*
  moreover
  hence $\mathcal{C} \vdash [C\ v] : ([] \text{ -> } [T\text{-}f32])$
    using *assms(2) unlift-b-e*
    by *fastforce*
  hence $\exists\, c.\ v = ConstFloat32\ c$
  proof (*induction* $[C\ v]$ $([] \text{ -> } [T\text{-}f32])$ *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*
        $\mathcal{S}\cdot\mathcal{C} \vdash cs : ([] \text{ -> } [T\text{-}f64])$
  shows $\exists\, c.\ cs = [\$C\ ConstFloat64\ c]$
proof −

```
  obtain v where cs = [$C v]
    using const-of-const-list assms(1) e-type-const-list[OF assms]
    by fastforce
  moreover
  hence 𝒞 ⊢ [C v] : ([] -> [T-f64])
    using assms(2) unlift-b-e
    by fastforce
  hence ∃ c. v = ConstFloat64 c
  proof (induction [C v] ([] -> [T-f64]) rule: b-e-typing.induct)
    case (const 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 𝒮•𝒞 ⊢ cs : ([] -> [t])
          is-int-t t
          const-list cs
          e = Unop-i t iop ∨ e = Testop t testop
  shows ∃ a s' vs' es'. ⦇s;vs;cs@([$e])⦈ ⇝-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 re-
duce-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 re-
duce-simple.intros(14)]
    by fastforce
qed (simp-all add: is-int-t-def)

lemma progress-unop-f:
  assumes 𝒮•𝒞 ⊢ cs : ([] -> [t])
          is-float-t t
          const-list cs
          e = Unop-f t iop
  shows ∃ a s' vs' es'. ⦇s;vs;cs@([$e])⦈ ⇝-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 re-duce-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 re-duce-simple.intros*(*14*)]
    by *fastforce*
qed (*simp-all add*: *is-float-t-def*)

lemma *const-list-split-2*:
  assumes *const-list cs*
    $\mathcal{S}\boldsymbol{\cdot}\mathcal{C} \vdash cs$: ([] -> [*t1*, *t2*])
  shows $\exists$ *c1 c2*. ($\mathcal{S}\boldsymbol{\cdot}\mathcal{C} \vdash$ [*c1*] : ([] -> [*t1*]))
        $\wedge$ ($\mathcal{S}\boldsymbol{\cdot}\mathcal{C} \vdash$ [*c2*] : ([] -> [*t2*]))
        $\wedge$ *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* $\mathcal{S}$ $\mathcal{C}$ [*c1*] *c2*] *e-type-const*[*of c2* $\mathcal{S}$ $\mathcal{C}$ - [*t1*,*t2*]]
    unfolding *const-list-def*
    by *fastforce*
qed

lemma *const-list-split-3*:
  assumes *const-list cs*
    $\mathcal{S}\boldsymbol{\cdot}\mathcal{C} \vdash cs$: ([] -> [*t1*, *t2*, *t3*])
  shows $\exists$ *c1 c2 c3*. ($\mathcal{S}\boldsymbol{\cdot}\mathcal{C} \vdash$ [*c1*] : ([] -> [*t1*]))
        $\wedge$ ($\mathcal{S}\boldsymbol{\cdot}\mathcal{C} \vdash$ [*c2*] : ([] -> [*t2*]))
        $\wedge$ ($\mathcal{S}\boldsymbol{\cdot}\mathcal{C} \vdash$ [*c3*] : ([] -> [*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 $\mathcal{S}$•$\mathcal{C}$ ⊢ *cs* : ([] -> [*t*, *t*])
is-int-t t
const-list cs
e = *Binop-i t iop* ∨ *e* = *Relop-i t irop*
shows ∃ *a s′ vs′ es′*. (|*s*;*vs*;*cs*@([$*e*])|) ⇝-*i* (|*s′*;*vs′*;*es′*|)
using *assms*(*2*)
proof (*cases t*)
case (*T-i32*)
hence *cs-def*:∃ *c1 c2*. *cs* = [$*C ConstInt32 c1*,$*C ConstInt32 c2*]
using *const-list-split-2*[*OF assms*(*3*,*1*)] *assms*(*3*) *const-of-i32*
unfolding *const-list-def*
by *blast*
show *?thesis*
proof (*cases e* = *Binop-i t iop*)
case *True*
obtain *c1 c2* **where** *cs* = [$*C ConstInt32 c1*,$*C ConstInt32 c2*]
using *cs-def*
by *blast*
thus *?thesis*
apply (*cases app-binop-i iop c1 c2*)
apply (*metis reduce-simple.intros*(*6*) *reduce.intros*(*1*) *T-i32 True append-Cons append-Nil*)
apply (*metis reduce-simple.intros*(*5*) *reduce.intros*(*1*) *T-i32 True append-Cons append-Nil*)
done
next
case *False*
thus *?thesis*
using *reduce-simple.intros*(*15*) *assms*(*4*) *reduce.intros*(*1*) *cs-def T-i32*
by *fastforce*
qed
next
case (*T-i64*)
hence *cs-def*:∃ *c1 c2*. *cs* = [$*C ConstInt64 c1*,$*C ConstInt64 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** $cs = [\$C\ ConstInt64\ c1, \$C\ ConstInt64\ c2]$
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 $\mathcal{S}{\cdot}\mathcal{C} \vdash cs : ([] \text{ -> } [t,\ t])$
is-float-t t
const-list cs
$e = $ *Binop-f t fop* $\vee$ $e = $ *Relop-f t frop*
shows $\exists\, a\ s'\ vs'\ es'.\ (\!|s;vs;cs@([\$e])|\!) \leadsto\text{-}i\ (\!|s';vs';es'|\!)$
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'*)
$\mathcal{S}$•$\mathcal{C} \vdash$ *cs* : ([] -> *ts*)
($\bigwedge$*lholed*. ¬(*Lfilled 0 lholed* [$*Return*] (*cs*@($*b-es*))))
$\bigwedge$ *i lholed*. ¬(*Lfilled 0 lholed* [$*Br* (*i*)] (*cs*@($*b-es*)))
const-list cs
¬ *const-list* ($* *b-es*)
i < *length* (*s-inst* $\mathcal{S}$)
length (*local* $\mathcal{C}$) = *length* (*vs*)
Option.is-none (*memory* $\mathcal{C}$) = *Option.is-none* (*inst.mem* ((*inst s*)!*i*))
shows $\exists$ *a s' vs' es'*. ⦇*s*;*vs*;*cs*@($*b-es*)⦈ ⇝-*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*(*2*,*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* $\mathcal{C}$ *uy*)
thus *?case*
using *progress-unop-testop-i*[*OF testop*(*2*,*1*)]
by *fastforce*
next
case (*relop-i t* $\mathcal{C}$ *uz*)
thus *?case*
using *progress-binop-relop-i*[*OF relop-i*(*2*,*1*)]
by *fastforce*
next
case (*relop-f t* $\mathcal{C}$ *va*)
thus *?case*
using *progress-binop-relop-f*[*OF relop-f*(*2*,*1*,*5*)]
by *fastforce*
next
case (*convert t1 t2 sx* $\mathcal{C}$)
obtain *v* **where** *cs-def*:*cs* = [$ *C v*] *typeof v* = *t2*
using *const-typeof const-of-const-list*[*OF - convert*(*6*)] *e-type-const-list*[*OF convert*(*6*,*3*)]
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*
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*
unfolding *types-agree-def*
by *fastforce*
qed
next
case (*reinterpret t1 t2* $\mathcal{C}$)
obtain *v* **where** *cs-def*:*cs* = [\$ *C v*] *typeof v* = *t2*
using *const-typeof const-of-const-list*[*OF - reinterpret*(*6*)] *e-type-const-list*[*OF reinterpret*(*6*,*3*)]
by *fastforce*
thus *?case*
using *reduce.intros*(*1*)[*OF reduce-simple.reinterpret*]
unfolding *types-agree-def*
by *fastforce*
next
case (*unreachable* $\mathcal{C}$ *ts ts'*)
thus *?case*
using *reduce.intros*(*1*)[*OF reduce-simple.unreachable*] *progress-L0*[*OF - unreachable*(*4*)]
by *fastforce*
next
case (*nop* $\mathcal{C}$)
thus *?case*
using *reduce.intros*(*1*)[*OF reduce-simple.nop*] *progress-L0*[*OF - nop*(*4*)]
by *fastforce*
next
case (*drop* $\mathcal{C}$ *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* $\mathcal{C}$ *t*)
obtain *v1 v2 v3* **where** *cs-def*:$\mathcal{S}$•$\mathcal{C}$ ⊢ [\$ *C v3*] : ([] -> [*T-i32*])
cs = [\$*C v1*, \$*C v2*, \$ *C v3*]
using *const-list-split-3*[*OF select*(*4*,*1*)] *select*(*4*)
unfolding *const-list-def*
by (*metis list-all-simps*(*1*) *e-type-const-unwrap*)
obtain *c3* **where** *c-def*:*v3* = *ConstInt32 c3*
using *cs-def select*(*4*) *const-of-i32*[*OF - cs-def*(*1*)]
unfolding *const-list-def*
by *fastforce*
have ∃ *a s' vs' es'*. ⦇*s*;*vs*;[\$*C v1*, \$*C v2*, \$ *C ConstInt32 c3*, \$*Select*]⦈ ⇝-*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* $\mathcal{C}$ *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* $\mathcal{C}$ *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* $\mathcal{C}$ *es1 es2*)
  obtain *c1s c2s* **where** *cs-def*:$\mathcal{S}$•$\mathcal{C}$ ⊢ *c1s* : ([] -> *tn*)
        $\mathcal{S}$•$\mathcal{C}$ ⊢ *c2s* : ([] -> [*T-i32*])
        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 c*)]
    using *const-of-i32 cs-def*
    by *fastforce*
  have ∃ *a s′ vs′ es′*. ⦇*s*;*vs*;[$ *C* (*ConstInt32 c*), $ *If tf es1 es2*]⦈ ⇝-*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 - [] [$Br i]]
    by fastforce
next
  case (br-if j 𝒞 ts)
  obtain cs1 cs2 where cs-def:𝒮•𝒞 ⊢ cs1 : ([] -> ts)
                              𝒮•𝒞 ⊢ cs2 : ([] -> [T-i32])
                              const-list cs1
                              const-list cs2
                              cs = cs1 @ cs2
    using e-type-const-list-cons[OF br-if(6,3)] e-type-const-list
    by fastforce
  obtain c where c-def:cs2 = [$C ConstInt32 c]
    using const-of-i32[OF cs-def(4,2)]
    by blast
  have ∃ a s' vs' es'. ⦇s;vs;cs2@($* [Br-if j])⦈ ⇝-i ⦇s';vs';es'⦈
  proof (cases int-eq c 0)
    case True
    thus ?thesis
      using c-def reduce.intros(1)[OF reduce-simple.br-if-false]
      by fastforce
  next
    case False
    thus ?thesis
      using c-def reduce.intros(1)[OF reduce-simple.br-if-true]
      by fastforce
  qed
  thus ?case
    using cs-def(5) progress-L0[OF - cs-def(3), of s vs cs2 @ ($* [Br-if j]) - - - -
[]]
    by fastforce
next
  case (br-table 𝒞 ts is i' t1s t2s)
  obtain cs1 cs2 where cs-def:𝒮•𝒞 ⊢ cs1 : ([]-> (t1s @ ts))
                              𝒮•𝒞 ⊢ cs2 : ([] -> [T-i32])
                              const-list cs1
                              const-list cs2
                              cs = cs1 @ cs2
    using e-type-const-list-cons[OF br-table(5), of 𝒮 𝒞 (t1s @ ts) [T-i32]]
          e-type-const-list[of - 𝒮 𝒞 t1s @ ts (t1s @ ts) @ [T-i32]]
          br-table(2,5)
    unfolding const-list-def
    by fastforce
```

obtain *c* **where** *c-def*:*cs2* = [\$*C ConstInt32 c*]
  using *const-of-i32*[*OF cs-def*(*4*,*2*)]
  by *blast*
have $\exists$ *a s′ vs′ es′*. ⦇*s*;*vs*;[\$*C ConstInt32 c*, \$*Br-table is i*]⦈ ⇝-*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* ≤ *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* 𝒞 *ts t1s t2s*)
thus *?case*
  using *Lfilled.intros*(*1*)[*OF return*(*5*), *of* - [] [\$*Return*]]
  by *fastforce*

next
case (*call j* 𝒞)
show *?case*
  using *progress-L0*[*OF reduce.intros*(*2*) *call*(*6*)]
  by *fastforce*

next
case (*call-indirect j* 𝒞 *t1s t2s*)
obtain *cs1 cs2* **where** *cs-def*:𝒮•𝒞 ⊢ *cs1* : ([]-> *t1s*)
𝒮•𝒞 ⊢ *cs2* : ([] -> [*T-i32*])
const-list cs1
const-list cs2
cs = *cs1* @ *cs2*
  using *e-type-const-list-cons*[*OF call-indirect*(*7*), *of* 𝒮 𝒞 *t1s* [*T-i32*]]
    e-type-const-list[*of* - 𝒮 𝒞 *t1s t1s* @ [*T-i32*]]
    call-indirect(*4*)
  by *fastforce*
obtain *c* **where** *c-def*:*cs2* = [\$*C ConstInt32 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* ∧ *stypes s i j* = *tf* ∧ *cl-type cl* = *tf*
| (*2*) $\exists$ *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 $\exists$ *a s′ vs′ es′*. ⦇*s*;*vs*;[$*C ConstInt32 c*, $*Call-indirect j*]⦈ ⇝-*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* $\mathcal{C}$ *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* $\mathcal{C}$ *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* $\mathcal{C}$ *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* $\mathcal{C}$ *t*)
thus *?case*
using *reduce.intros*(*10*)[*of s vs j i*] *progress-L0*
by *fastforce*
next
case (*set-global j* $\mathcal{C}$ *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* $\mathcal{C}$ *n a tp-sx t off*)
obtain *c* **where** *c-def*: *cs* = [$*C ConstInt32 c*]
using *const-of-i32 load*(*3*,*6*) *e-type-const-unwrap*
unfolding *const-list-def*
by *fastforce*
obtain *j* **where** *mem-some*:*smem-ind s i* = *Some j*
using *load*(*1*,*10*)
unfolding *smem-ind-def*
by *fastforce*
have ∃ *a' s' vs' es'*. ⦇*s*;*vs*;[$*C ConstInt32 c*, $*Load t tp-sx a off*]⦈ ⇝-*i* ⦇*s'*;*vs'*;*es'*⦈
proof (*cases tp-sx*)
case *None*
note *tp-none* = *None*
show *?thesis*
proof (*cases load* ((*mem s*)!*j*) (*nat-of-int c*) *off* (*t-length t*))
case *None*
show *?thesis*
using *reduce.intros*(*13*)[*OF mem-some - None*, *of vs*] *tp-none load*(*2*)
by *fastforce*
next

```
      case (Some a)
      show ?thesis
        using reduce.intros(12)[OF mem-some - Some, of vs] tp-none 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 ((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 vs] tp-some load(2)
        by fastforce
    next
      case (Some a)
      show ?thesis
        using reduce.intros(14)[OF mem-some - Some, of vs] tp-some load(2)
        by fastforce
    qed
  qed
  then show ?case
    using c-def progress-L0
    by fastforce
next
  case (store C n a tp t off)
  obtain cs' v where cs-def:S•C ⊢ [cs'] : ([] -> [T-i32])
                            S•C ⊢ [$ C v] : ([] -> [t])
                            cs = [cs',$ C v]
    using const-list-split-2[OF store(6,3)] e-type-const-unwrap
    unfolding const-list-def
    by fastforce
  have t-def:typeof v = t
    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(1,10)
    unfolding smem-ind-def
    by fastforce
  obtain c where c-def:cs' = $C ConstInt32 c
    using const-of-i32[OF - cs-def(1)] cs-def(3) store(6)
    unfolding const-list-def
    by fastforce
  have ∃a' s' vs' es'. ⦇s;vs;[$C ConstInt32 c, $C v, $Store t tp a off]⦈ ⇝-i
⦇s';vs';es'⦈
  proof (cases tp)
```

case *None*
note *tp-none* = *None*
show *?thesis*
proof (*cases store* (*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 vs*] *t-def tp-none store*(*2*)
unfolding *types-agree-def*
by *fastforce*
next
case (*Some a*)
show *?thesis*
using *reduce.intros*(*16*)[*OF - mem-some - Some*, *of vs*] *t-def tp-none store*(*2*)
unfolding *types-agree-def*
by *fastforce*
qed
next
case (*Some a*)
note *tp-some* = *Some*
show *?thesis*
proof (*cases store-packed* (*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 vs*] *t-def tp-some store*(*2*)
unfolding *types-agree-def*
by *fastforce*
next
case (*Some a*)
show *?thesis*
using *reduce.intros*(*18*)[*OF - mem-some - Some*, *of t vs*] *t-def tp-some 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*)
obtain *j* **where** *mem-some*:*smem-ind s i* = *Some j*
using *current-memory*(*1*,*9*)
unfolding *smem-ind-def*
by *fastforce*
thus *?case*
using *progress-L0*[*OF reduce.intros*(*20*)[*OF mem-some*] *current-memory*(*5*), *of - - vs* []]
by *fastforce*

```
next
  case (grow-memory 𝒞 n)
  obtain c where c-def:cs = [$C ConstInt32 c]
    using const-of-i32 grow-memory(2,5)
    by fastforce
  obtain j where mem-some:smem-ind s i = Some j
    using grow-memory(1,9)
    unfolding smem-ind-def
    by fastforce
  show ?case
    using reduce.intros(22)[OF mem-some, of -] c-def
    by fastforce
next
  case (empty 𝒞)
  thus ?case
    unfolding const-list-def
    by simp
next
  case (composition 𝒞 es t1s t2s e t3s)
  consider (1) ¬ const-list ($* es) | (2) const-list ($* es) ¬ const-list ($*[e])
    using composition(9)
    unfolding const-list-def
    by fastforce
  thus ?case
  proof (cases)
    case 1
    have (⋀lholed. ¬ Lfilled 0 lholed [$Return] (cs @ ($* es)))
         (⋀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 composi-
tion(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 𝒮•𝒞 ⊢ (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 (⋀lholed. ¬ Lfilled 0 lholed [$Return] (cs2 @ ($* es)))
         (⋀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*)⦈ ⇝-*i* ⦇*s′*;*vs′*;*es′*⦈
using *weakening*(*2*)[*OF cs-def*(*2*) - - *cs-def*(*5*) *weakening*(*7*)] *weakening*(*8*−)
by *fastforce*
thus *?case*
using *progress-L0*[*OF* - *cs-def*(*4*), *of s vs cs2* @ ($* *es*) *i* - - - []] *cs-def*(*3*)
by *fastforce*
qed

lemma *progress-e*:
assumes 𝒮•*None* ⊩-*i vs*;*cs-es* : *ts′*
⋀ *k lholed*. ¬(*Lfilled k lholed* [$*Return*] *cs-es*)
⋀ *i k lholed*. (*Lfilled k lholed* [$*Br* (*i*)] *cs-es*) ⟹ *i* < *k*
cs-es ≠ [*Trap*]
¬ *const-list* (*cs-es*)
store-typing s 𝒮
shows ∃ *a s′ vs′ es′*. ⦇*s*;*vs*;*cs-es*⦈ ⇝-*i* ⦇*s′*;*vs′*;*es′*⦈
proof −
fix 𝒞 *cs es ts-c*
have *prems1*:
𝒮•𝒞 ⊢ *es* : (*ts-c* -> *ts′*) ⟹
𝒮•𝒞 ⊢ *cs-es* : ([] -> *ts′*) ⟹
cs-es = *cs*@*es* ⟹
const-list cs ⟹
𝒮•𝒞 ⊢ *cs* : ([] -> *ts-c*) ⟹
(⋀ *k lholed*. ¬(*Lfilled k lholed* [$*Return*] *cs-es*)) ⟹
(⋀ *i k lholed*. (*Lfilled k lholed* [$*Br* (*i*)] *cs-es*) ⟹ *i* < *k*) ⟹
cs-es ≠ [*Trap*] ⟹
¬ *const-list* (*cs-es*) ⟹
store-typing s 𝒮 ⟹
i < *length* (*s-inst* 𝒮) ⟹
length (*local* 𝒞) = *length* (*vs*) ⟹

```
      Option.is-none (memory 𝒞) = Option.is-none (inst.mem ((inst s)!i)) ⟹
        ∃ a s′ vs′ cs-es′. ⦇s;vs;cs-es⦈ ⇝-i ⦇s′;vs′;cs-es′⦈
  and prems2:
      𝒮•None ⊩-i vs;cs-es : ts′ ⟹
      (⋀ k lholed. ¬(Lfilled k lholed [$Return] cs-es)) ⟹
      (⋀ i k lholed. (Lfilled k lholed [$Br (i)] cs-es) ⟹ i < k) ⟹
      cs-es ≠ [Trap] ⟹
      ¬ const-list (cs-es) ⟹
      store-typing s 𝒮 ⟹
        ∃ a s′ vs′ cs-es′. ⦇s;vs;cs-es⦈ ⇝-i ⦇s′;vs′;cs-es′⦈
proof (induction arbitrary: vs ts-c ts′ i cs-es cs rule: e-typing-s-typing.inducts)
  case (1 𝒞 b-es tf 𝒮)
  hence 𝒞 ⊢ b-es : (ts-c -> ts′)
    using e-type-comp-conc1[of 𝒮 𝒞 cs ($* b-es) [] ts′] unlift-b-e
    by (metis e-type-const-conv-vs typing-map-typeof)
  then show ?case
    using progress-b-e[OF - 1(5) - - 1(4)] 1(3,4,9) list-all-append 1
    unfolding const-list-def
    by fastforce
next
  case (2 𝒮 𝒞 es t1s t2s e t3s)
  show ?case
  proof (cases const-list es)
    case True
    hence const-list (cs@es)
      using 2(7)
      unfolding const-list-def
      by simp
    moreover
    have ∃ ts″. (𝒮•𝒞 ⊢ (cs @ es) : ([] -> ts″))
      using 2(5,6)
      by (metis append.assoc e-type-comp-conc1)
    ultimately
    show ?thesis
      using 2(4)[OF 2(5) - - - 2(9,10,11,12,13,14,15), of (cs@es)] 2(6,16)
      by fastforce
  next
    case False
    hence ¬const-list (cs@es)
      unfolding const-list-def
      by simp
    moreover
    have ∃ ts″. (𝒮•𝒞 ⊢ (cs @ es) : ([] -> ts″))
      using 2(5,6)
      by (metis append.assoc e-type-comp-conc1)
    moreover
    have ⋀k lholed. ¬ Lfilled k lholed [$Return] (cs @ es)
    proof −
      {
```

assume $\exists$ *k lholed. Lfilled k lholed* [\$*Return*] (*cs* @ *es*)
then obtain *k lholed* **where** *local-assms*:*Lfilled k lholed* [\$*Return*] (*cs* @ *es*)
by *blast*
hence $\exists$ *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*)
qed
hence *False*
using *2*(*6*,*9*)
by *blast*
}
thus $\bigwedge$*k lholed.* $\neg$ *Lfilled k lholed* [\$*Return*] (*cs* @ *es*)
by *blast*
qed
moreover
have $\bigwedge$*i k lholed. Lfilled k lholed* [\$*Br i*] (*cs* @ *es*) $\Longrightarrow$ *i* < *k*
proof −
{
assume $\exists$ *i k lholed. Lfilled k lholed* [\$*Br i*] (*cs* @ *es*) $\wedge$ $\neg$(*i* < *k*)
then obtain *i k lholed* **where** *local-assms*:*Lfilled k lholed* [\$*Br i*] (*cs* @ *es*) $\neg$(*i* < *k*)
by *blast*
hence $\exists$ *lholed'. Lfilled k lholed'* [\$*Br i*] (*cs* @ *es* @ [*e*]) $\wedge$ $\neg$(*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 [$Br i] (cs @ es) ⟹ i < k
        by blast
    qed
    moreover
    note preds = calculation
    show ?thesis
    proof (cases cs @ es = [Trap])
      case True
      thus ?thesis
        using reduce-simple.trap[of - (LBase [] [e])]
              Lfilled.intros(1)[of [] LBase [] [e] [e] cs @ es]
              reduce.intros(1) 2(6,11)
        unfolding const-list-def
        by (metis append.assoc append-Nil list.pred-inject(1))
    next
      case False
      thus ?thesis
        using 2(3)[OF - - 2(7,8) - - - - 2(13,14,15)] preds 2(6,16)
              progress-L0[of s vs (cs @ es) - - - - [] [e]]
        unfolding const-list-def
        by (metis append.assoc append-Nil list.pred-inject(1))
    qed
  qed
next
  case (3 𝒮 𝒞 es t1s t2s ts)
  thus ?case
    by fastforce
next
  case (4 𝒮 𝒞)
  have cs-es-def:Lfilled 0 (LBase cs []) [Trap] cs-es
    using Lfilled.intros(1)[OF 4(3), of - [] [Trap]] 4(2)
    by fastforce
  thus ?case
    using reduce-simple.trap[OF 4(7) cs-es-def] reduce.intros(1)
    by blast
next
case (5 𝒮 ts j vls es n 𝒞)
  consider (1) (⋀k lholed. ¬ Lfilled k lholed [$Return] es)
               (⋀k lholed i. (Lfilled k lholed [$Br i] es) ⟹ i < k)
               es ≠ [Trap]
               ¬ const-list es
         | (2) ∃k lholed. Lfilled k lholed [$Return] es
         | (3) const-list es ∨ (es = [Trap])
```

```
        | (4) ∃k lholed i. (Lfilled k lholed [$Br i] es) ∧ i ≥ k
      using not-le-imp-less
      by blast
    thus ?case
    proof (cases)
      case 1
      obtain s' vs'' a where temp1:(|s;vls;es|) ⇝- j (|s';vs'';a|)
        using 5(3)[OF 1(1) - 1(3,4) 5(12)] 1(2)
        by fastforce
      show ?thesis
        using reduce.intros(24)[OF temp1, of vs] progress-L0[where ?cs = cs, OF
- 5(6)] 5(5)
        by fastforce
    next
      case 2
      then obtain k lholed where local-assms:(Lfilled k lholed [$Return] es)
        by blast
     then obtain lholed' vs' C' where lholed'-def:(Lfilled k lholed' (vs'@[$Return])
es)
                                      S•C' ⊢ vs' : ([] -> ts)
                                      const-list vs'
          using progress-LN-return[OF local-assms, of S - ts ts] s-type-unfold[OF
5(1)]
        by fastforce
      hence temp1:∃a. (|[Local n j vls es]|) ⇝ (|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:∃a. (|s;vs;[Local n j vls es]|) ⇝- 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-inst* $\mathcal{S}$)!*j*)(|*local* := (*local* ((*s-inst* $\mathcal{S}$)!*j*)) @ (*map typeof vls*), *return* := *Some ts*|)
by *blast*
hence *es-def*:$\mathcal{S}$•$\mathcal{C}'$ ⊢ *es* : ([] -> *ts*) *j* < *length* (*s-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* $\mathcal{S}$ *cl tf* $\mathcal{C}$)
obtain *ts″* **where** *ts″-def*:$\mathcal{S}$•$\mathcal{C}$ ⊢ *cs* : ([] -> *ts″*) $\mathcal{S}$•$\mathcal{C}$ ⊢ [*Callcl cl*] : (*ts″* -> *ts′*)
using *6*(*2*,*3*) *e-type-comp-conc1*
by *fastforce*
obtain *ts-c t1s t2s* **where** *cl-def*:(*ts″* = *ts-c* @ *t1s*)
(*ts′* = *ts-c* @ *t2s*)
cl-type cl = (*t1s* -> *t2s*)
using *e-type-callcl*[*OF ts″-def*(*2*)]
by *fastforce*
obtain *vs1 vs2* **where** *vs-def*:$\mathcal{S}$•$\mathcal{C}$ ⊢ *vs1* : ([] -> *ts-c*)
$\mathcal{S}$•$\mathcal{C}$ ⊢ *vs2* : (*ts-c* -> *ts-c* @ *t1s*)
cs = *vs1* @ *vs2*
const-list vs1
const-list vs2
using *e-type-const-list-cons*[*OF 6*(*4*)] *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* (*t1s* -> *t2s*) *x13 x14*
using *cl-def*(*3*)
unfolding *cl-type-def*
by *simp*

```
      have ∃ a a'. ⦇s;vs;vs2 @ [Callcl cl]⦈ ⇝- 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(3)
        by fastforce
    next
      case (Func-host x21 x22)
      hence func-host-def:cl = Func-host (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]⦈ ⇝- i ⦇s';vs;a⦈
      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(3) progress-L0
        by fastforce
    qed
  next
    case (7 𝒮 𝒞 e0s ts t2s es n)
    consider (1) (⋀k lholed. ¬ Lfilled k lholed [$Return] es)
                 (⋀k lholed i. (Lfilled k lholed [$Br i] es) ⟹ i < k)
                 es ≠ [Trap]
                 ¬ const-list es
           | (2) ∃ k lholed. Lfilled k lholed [$Return] es
           | (3) const-list es ∨ (es = [Trap])
```

| (4) ∃ k lholed i. (Lfilled k lholed [$Br i] es) ∧ i = k
| (5) ∃ k lholed i. (Lfilled k lholed [$Br i] es) ∧ 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|) ⊢ [] : ([] -> [])
using *b-e-typing.empty e-typing-s-typing.intros*(*1*)
by *fastforce*
have ∃ *s′ vs′ a*. (|*s*;*vs*;*es*|) ⇝- *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*:(|*s*;*vs*;*es*|) ⇝- *i* (|*s′*;*vs′*;*a*|)
by *blast*
have *temp4*:⋀*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*:∃ *a*. ⦇*s*;*vs*;[*Label n e0s es*]⦈ ⇝- *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*)
$\mathcal{S}$•*𝒞′* ⊢ *vs′* : ([] -> *ts*)
const-list vs′
using *progress-LN*[*OF lholed-def 7*(*2*), *of ts*]
by *fastforce*
have ∃ *es′ a*. ⦇[*Label n e0s es*]⦈ ⇝ ⦇*vs′*@*e0s*⦈
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 ∃ *es′ a*. ⦇*s*;*vs*;[*Label n e0s es*]⦈ ⇝- *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* $\mathcal{S}$ *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.is-none* (*memory 𝒞*) = *Option.is-none* (*inst.mem* ((*inst s*)!*i*))
using *store-mem-exists*[*OF 8*(*1*,*11*)] *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* $\mathcal{C}$]]
unfolding *const-list-def*
by *fastforce*
qed
show *?thesis*
using *prems2*[*OF assms*]
by *fastforce*
qed

lemma *progress-e1*:
assumes $\mathcal{S}\bullet None \Vdash\text{-}i\ vs;es : ts$
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*:$i < length\ (s\text{-}inst\ \mathcal{S})$
$\mathcal{C} = ((s\text{-}inst\ \mathcal{S})!i)(\!|local := (local\ ((s\text{-}inst\ \mathcal{S})!i))\ @\ (map\ typeof\ vs),$
$return := None|\!)$
$(\mathcal{S}\bullet\mathcal{C} \vdash es : ([] \text{ -> } ts))$
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 $\mathcal{S}\bullet None \Vdash\text{-}i\ vs;es : ts$
store-typing s $\mathcal{S}$
shows $(Lfilled\ k\ lholed\ [\$Br\ (j)]\ es) \Longrightarrow j < k$
proof −

{
assume (∃ *i k lholed*. (*Lfilled k lholed* [$*Br* (*i*)] *es*) ∧ *i* ≥ *k*)
then obtain *j k lholed* **where** *local-assms*:(*Lfilled k lholed* [$*Br* (*k*+*j*)] *es*)
by (*metis le-iff-add*)
obtain 𝒞 **where** *c-def*:*i* < *length* (*s-inst* 𝒮)
𝒞 = ((*s-inst* 𝒮)!*i*)⦇*local* := (*local* ((*s-inst* 𝒮)!*i*)) @ (*map typeof vs*), *return* := *None*⦈
(𝒮•𝒞 ⊢ *es* : ([] -> *ts*))
using *assms s-type-unfold*
by *fastforce*
have *j* < *length* (*label* 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 (⋀ *j k lholed*. (*Lfilled k lholed* [$*Br* (*j*)] *es*) ⟹ *j* < *k*)
by *fastforce*
qed

lemma *progress-e3*:
assumes 𝒮•*None* ⊩-*i vs*;*cs-es* : *ts*′
cs-es ≠ [*Trap*]
¬ *const-list* (*cs-es*)
store-typing s 𝒮
shows ∃ *a s*′ *vs*′ *es*′. ⦇*s*;*vs*;*cs-es*⦈ ⇝-*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 ⊢-*i s*;*vs*;*es* : *ts*
⦇*s*;*vs*;*es*⦈ ⇝-*i* ⦇*s*′;*vs*′;*es*′⦈
shows ⊢-*i s*′;*vs*′;*es*′ : *ts*
proof −
obtain 𝒮 **where** *store-typing s* 𝒮 𝒮•*None* ⊩-*i vs*;*es* : *ts*
using *assms*(*1*) *config-typing.simps*
by *blast*
hence *store-typing s*′ 𝒮 𝒮•*None* ⊩-*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 $\vdash$-*i* $s;vs;es : ts$
shows *const-list es* $\vee$ $es = [Trap] \vee (\exists\, a\ s'\ vs'\ es'.\ (\!|s;vs;es|\!) \rightsquigarrow\text{-}i\ (\!|s';vs';es'|\!))$
proof −
obtain $\mathcal{S}$ **where** *store-typing s* $\mathcal{S}$ $\mathcal{S}$•*None* $\Vdash$-*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
| *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 *ct-eq* :: *ct* ⇒ *ct* ⇒ *bool* **where**
ct-eq (*TSome t*) (*TSome t'*) = (*t* = *t'*)
| *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 (*metis ct-eq.elims*(*3*) *ct-eq.simps*(*1*))

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

lemma *ct-eq-common-tsome*: *ct-eq x y* = (∃ *t*. *ct-eq x* (*TSome t*) ∧ *ct-eq* (*TSome t*) *y*)
by (*metis ct-eq.elims*(*3*) *ct-eq.simps*(*1*))

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 ct-eq.elims*(*3*) *ct-eq.simps*(*1*) *list-all2-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*: ∃ *ys*. *ct-list-eq xs* (*to-ct-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(3) ct-eq-commute list.rel-intros(2) list.simps(9))
    apply (metis ct-eq.simps(1) list.rel-intros(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)
```

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 $\exists$ *bs bs'*. *as* = *bs* @ *bs'* $\wedge$ *ct-list-eq* (*to-ct-list bs*) *xs* $\wedge$ *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 $\exists$ *bs*. *as* = *bs* @ *ys* $\wedge$ *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*

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*) $\wedge$ *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 $\exists\, as'$. *to-ct-list* as' = *as*
$\exists\, bs'$. *to-ct-list* bs' = *bs*
using *assms*
proof (*induct as arbitrary*: *ts*)
fix *ts*
assume *to-ct-list ts* = [] @ *bs*
thus $\exists\, as'$. *to-ct-list* as' = []
$\exists\, bs'$. *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 $\exists\, as'$. *to-ct-list* as' = $a \# as$
$\exists\, as'$. *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* $\leq$ *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 $\exists$ *as bs. ts'* = *as* @ *bs* $\wedge$ *ct-list-eq bs ts*
then obtain *ccs* :: *ct list* **and** *ccsa* :: *ct list* **where**
f1: *ts'* = *ccs* @ *ccsa* $\wedge$ *ct-list-eq ccsa ts*
by *moura*
then have *f2*: *length ccsa* = *length ts*
by (*meson ct-list-eq-length*)
have $\bigwedge$*n. ct-list-eq* (*take n ccsa*) (*take n ts*)
using *f1* **by** (*meson ct-list-eq-take*)
then show $\exists$ *cs csa. take* (*length ts'* $-$ *n*) *ts'* = *cs* @ *csa* $\wedge$ *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-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*:
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 (*meson ct-eq.simps*(*2*) *ct-list-eq-commute ct-list-eq-def list.rel-intros*(*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-list-eq-def*)

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*
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 *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 *select-return-top* :: [*ct list*] ⇒ *ct* ⇒ *ct* ⇒ *checker-type* **where**
select-return-top ts ct1 TAny = *TopType* ((*take* (*length ts* − *3*) *ts*) @ [*ct1*])
| *select-return-top ts TAny ct2* = *TopType* ((*take* (*length ts* − *3*) *ts*) @ [*ct2*])
| *select-return-top ts* (*TSome t1*) (*TSome t2*) = (*if* (*t1* = *t2*)
then (*TopType* ((*take* (*length ts* − *3*) *ts*) @ [*TSome t1*]))
else Bot)

fun *type-update-select* :: *checker-type* ⇒ *checker-type* **where**
type-update-select (*Type ts*) = (*if* (*length ts* ≥ *3* ∧ (*ts*!(*length ts*−*2*)) = (*ts*!(*length ts*−*3*)))
then consume (*Type ts*) [*TAny*, *TSome T-i32*]
else Bot)
| *type-update-select* (*TopType ts*) = (*case length ts of*
0 ⇒ *TopType* [*TAny*]
| *Suc 0* ⇒ *type-update* (*TopType ts*) [*TSome T-i32*] (*TopType* [*TAny*])
| *Suc* (*Suc 0*) ⇒ *consume* (*TopType ts*) [*TSome T-i32*]
| - ⇒ *type-update* (*TopType ts*) [*TAny*, *TAny*, *TSome T-i32*]
(*select-return-top 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 *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**
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 $\bigwedge$*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 $\leq$ *length* ts'
c-t $\neq$ *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 $\exists$ *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 $\exists$ *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*) $\wedge$ *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 suf-fixE*
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
shows ∃ *c-t'*. *consume ctn cts* = *c-t'* ∧ *c-types-agree c-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 ∨ 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' ≠ Bot
        shows (ts' = prods ∧ (∃ ts-c. prods = (TopType ts-c)))
                 ∨ (∃ ts-a ts-b. prods = Type ts-a ∧ ts = ts-b@cons ∧ ts' = Type
(ts-b@ts-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'*
$\exists$ *t-ag'*. *t-ag* = *t-ag'@prods* $\wedge$ *c-types-agree* (*TopType ts*) (*t-ag'@cons*)
proof −
consider (*1*) *ct-suffix* (*to-ct-list cons*) *ts*
| (*2*) $\neg$*ct-suffix* (*to-ct-list cons*) *ts ct-suffix ts* (*to-ct-list cons*)
| (*3*) $\neg$*ct-suffix* (*to-ct-list cons*) *ts* $\neg$*ct-suffix ts* (*to-ct-list cons*)
by *blast*
hence *ct-suffix* (*to-ct-list prods*) *ts'* $\wedge$ ($\exists$ *t-ag'*. *t-ag* = *t-ag'@prods* $\wedge$ *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*) *consume.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-ag'*. *t-ag* = *t-ag'@prods* $\wedge$ *c-types-agree* (*TopType ts*) (*t-ag'@cons*)
by *simp-all*
qed

lemma *type-update-select-length0*:

assumes *type-update-select* (*TopType cts*) = *tm*
length cts = *0*
tm ≠ *Bot*
shows *tm* = *TopType* [*TAny*]
using *assms*
by *simp*

lemma *type-update-select-length1*:
assumes *type-update-select* (*TopType cts*) = *tm*
length cts = *1*
tm ≠ *Bot*
shows *ct-list-eq cts* [*TSome T-i32*]
tm = *TopType* [*TAny*]
proof −
have *1*:*type-update* (*TopType cts*) [*TSome T-i32*] (*TopType* [*TAny*]) = *tm*
using *assms*(*1*,*2*)
by *simp*
hence *ct-suffix cts* [*TSome T-i32*] ∨ *ct-suffix* [*TSome T-i32*] *cts*
using *assms*(*3*)
by (*metis consume.simps*(*2*) *produce.simps*(*7*) *type-update.simps*)
thus *ct-list-eq cts* [*TSome T-i32*]
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* [*TAny*]
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* (*TopType cts*) = *tm*
length cts = *2*
tm ≠ *Bot*
shows ∃ *t1 t2*. *cts* = [*t1*, *t2*] ∧ *ct-eq t2* (*TSome T-i32*) ∧ *tm* = *TopType* [*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 *consume* (*TopType* [*x*,*y*]) [*TSome T-i32*] = *tm*
using *assms*(*1*,*2*)
by *simp*
moreover
hence *ct-suffix* [*x*,*y*] [*TSome T-i32*] ∨ *ct-suffix* [*TSome T-i32*] [*x*,*y*]
using *assms*(*3*)
by (*metis consume.simps*(*2*))
hence *ct-suffix* [*TSome T-i32*] ([*x*]@[*y*])

using *assms*(*2*) *ct-suffix-length*
by *fastforce*
moreover
hence *ct-eq y* (*TSome T-i32*)
by (*metis ct-eq-commute ct-list-eq-def ct-suffix-cons2 list.rel-sel list.sel*(*1*) *list.simps*(*3*) *list.size*(*4*))
ultimately
show *?thesis*
by *auto*
qed

lemma *type-update-select-length3*:
assumes *type-update-select* (*TopType cts*) = (*TopType ctm*)
length cts $\geq$ *3*
shows $\exists$ *cts′ ct1 ct2 ct3*. *cts* = *cts′*@[*ct1*, *ct2*, *ct3*] $\wedge$ *ct-eq ct3* (*TSome T-i32*)
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*)
hence (*type-update* (*TopType cts*) [*TAny*, *TAny*, *TSome T-i32*] (*select-return-top cts* (*cts* ! (*length cts* − *2*)) (*cts* ! (*length cts* − *3*)))) = *TopType ctm*
using *assms*
by *simp*
then obtain *c-mid* **where** *consume* (*TopType cts*) [*TAny*, *TAny*, *TSome T-i32*] = *TopType c-mid*
by (*metis consume.simps*(*2*) *produce.simps*(*6*) *type-update.simps*)
hence *ct-suffix* [*TAny*, *TAny*, *TSome T-i32*] (*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*)
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 *simp*
qed

lemma *type-update-select-type-length3*:
  assumes *type-update-select* (*Type tn*) = (*Type tm*)
  shows $\exists$ *t ts'*. *tn* = *ts'*@[*t*, *t*, *T-i32*]
proof −
  have *tn-cond*:(*length tn* $\geq$ *3* $\wedge$ (*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*] = *Type tm*
    using *assms*
    by *simp*
  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*)
                            zero-less-diff)
  have *ct-suffix* [*TAny*, *TSome T-i32*] (*to-ct-list* (*tn'* @ [*t1*, *t2*, *t3*]))
    using *tn-def tm-def*
    by (*simp*, *metis checker-type.distinct*(*5*))
  hence *t3* = *T-i32*
      using *ct-suffix-unfold-one*[*of* [*TAny*] *TSome T-i32 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 cts c1 c2* = *ctm*
          ctm $\neq$ *Bot*
  shows $\exists$*xs*. *ctm* = *TopType xs*

using *assms*
by (*meson select-return-top.elims*)

lemma *type-update-select-top-exists*:
assumes *type-update-select xs* = (*TopType tm*)
shows $\exists$ *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 *type-update-select-conv-select-return-top*:
assumes *ct-suffix* [*TSome T-i32*] *cts*
length cts $\geq$ *3*
shows *type-update-select* (*TopType cts*) = (*select-return-top cts* (*cts*!(*length cts*−*2*)) (*cts*!(*length cts*−*3*)))
proof −
obtain *nat* **where** *nat-def*:*length cts* = *Suc* (*Suc* (*Suc nat*))
using *assms*(*2*)
by (*metis add-eq-if diff-Suc-1 le-Suc-ex numeral-3-eq-3 nat.distinct*(*2*))
have *ct-suffix* [*TAny*, *TAny*, *TSome T-i32*] *cts*
using *assms ct-suffix-extend-any1*
by *simp*
then obtain *cts*′ **where** *consume* (*TopType cts*) [*TAny*, *TAny*, *TSome T-i32*] = *TopType cts*′
by *simp*
thus *?thesis*
using *nat-def select-return-top-exists*
apply (*cases select-return-top cts* (*cts* ! *Suc nat*) (*cts* ! *nat*))
apply *simp-all*
apply (*metis checker-type.distinct*(*1*) *checker-type.distinct*(*5*))
done
qed

lemma *select-return-top-ct-eq*:
assumes *select-return-top cts c1 c2* = *TopType ctm*
length cts $\geq$ *3*
c-types-agree (*TopType ctm*) *cm*
shows $\exists$ *c*′ *cm*′. *cm* = *cm*′@[*c*′]
$\wedge$ *ct-suffix* (*take* (*length cts* − *3*) *cts*) (*to-ct-list cm*′)
$\wedge$ *ct-eq c1* (*TSome c*′)
$\wedge$ *ct-eq c2* (*TSome c*′)
proof (*cases c1*)
case *TAny*
note *outer-TAny* = *TAny*
show *?thesis*

proof (*cases c2*)
case *TAny*
hence *take* (*length cts* − *3*) *cts* @ [*TAny*] = *ctm*
using *outer-TAny assms ct-suffix-cons-ct-list*
by *simp*
then obtain *cm′ c* **where** *cm* = *cm′* @ [*c*]
ct-suffix (*take* (*length cts* − *3*) *cts*) (*to-ct-list cm′*)
using *ct-suffix-cons-ct-list*[*of take* (*length cts* − *3*) *cts* [*TAny*]]
assms(*3*) *c-types-agree.simps*(*2*)[*of ctm cm*]
unfolding *ct-list-eq-def to-ct-list-def*
by (*metis Suc-leI impossible-Cons length-Cons length-map list.exhaust list.size*(*3*)
list-all2-conv-all-nth zero-less-Suc)
thus *?thesis*
using *TAny outer-TAny ct-eq.simps*(*2*)
by *blast*
next
case (*TSome x2*)
hence *take* (*length cts* − *3*) *cts* @ [*TSome x2*] = *ctm*
using *outer-TAny assms ct-suffix-cons-ct-list*
by *simp*
then obtain *cm′* **where** *cm* = *cm′* @ [*x2*]
ct-suffix (*take* (*length cts* − *3*) *cts*) (*to-ct-list cm′*)
using *ct-suffix-cons-ct-list*[*of take* (*length cts* − *3*) *cts* [*TSome x2*]]
assms(*3*) *c-types-agree.simps*(*2*)[*of ctm cm*] *ct-list-eq-ts-conv-eq*
unfolding *ct-list-eq-def to-ct-list-def*
by (*metis* (*no-types*, *opaque-lifting*) *list.simps*(*8*,*9*))
thus *?thesis*
using *TSome outer-TAny*
by *simp*
qed
next
case (*TSome x2*)
note *outer-TSome* = *TSome*
show *?thesis*
proof (*cases c2*)
case *TAny*
hence *take* (*length cts* − *3*) *cts* @ [*TSome x2*] = *ctm*
using *TSome assms ct-suffix-cons-ct-list*
by *simp*
then obtain *cm′* **where** *cm* = *cm′* @ [*x2*]
ct-suffix (*take* (*length cts* − *3*) *cts*) (*to-ct-list cm′*)
using *ct-suffix-cons-ct-list*[*of take* (*length cts* − *3*) *cts* [*TSome x2*]]
assms(*3*) *c-types-agree.simps*(*2*)[*of ctm cm*] *ct-list-eq-ts-conv-eq*
unfolding *ct-list-eq-def to-ct-list-def*
by (*metis* (*no-types*, *opaque-lifting*) *list.simps*(*8*,*9*))
thus *?thesis*
using *TSome TAny*
by *simp*
next

case (*TSome x3*)
hence *x-eq*:*x2* = *x3*
using *outer-TSome assms ct-suffix-cons-ct-list*
by (*metis checker-type.distinct*(*3*) *select-return-top.simps*(*3*))
hence *take* (*length cts* − *3*) *cts* @ [*TSome x2*] = *ctm*
using *TSome outer-TSome assms ct-suffix-cons-ct-list*
by (*metis checker-type.inject*(*1*) *select-return-top.simps*(*3*))
then obtain *cm′* **where** *cm* = *cm′* @ [*x2*]
ct-suffix (*take* (*length cts* − *3*) *cts*) (*to-ct-list cm′*)
using *ct-suffix-cons-ct-list*[*of take* (*length cts* − *3*) *cts* [*TSome x2*]]
assms(*3*) *c-types-agree.simps*(*2*)[*of ctm cm*] *ct-list-eq-ts-conv-eq*
unfolding *ct-list-eq-def to-ct-list-def*
by (*metis* (*no-types*, *opaque-lifting*) *list.simps*(*8*,*9*))
thus *?thesis*
using *TSome outer-TSome x-eq*
by *simp*
qed
qed

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*) ∧ (*sx* = *None*) = ((*is-float-t t1* ∧ *is-float-t t2*)
∨ (*is-int-t t1* ∧ *is-int-t t2* ∧ (*t-length t1* < *t-length t2*))))

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

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

lemma *same-lab-h-conv-list-all*:
assumes *same-lab-h ils ls ts′* = *Some ts*
shows *list-all* (λ*i*. *i* < *length ls* ∧ *ls*!*i* = *ts*) *ils* ∧ *ts′* = *ts*
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 (λi. i < length ls ∧ 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 (λi. i < length ls ∧ 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 (λi. i < length ls ∧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 ⇒  b-e list ⇒ tf ⇒ bool
and check :: t-context ⇒ b-e list ⇒ checker-type ⇒ checker-type
and check-single :: t-context ⇒  b-e ⇒ checker-type ⇒ checker-type where
  b-e-type-checker 𝒞 es (tn -> tm) = c-types-agree (check 𝒞 es (Type tn)) tm
| check 𝒞 es ts = (case es of
                     [] ⇒ ts
                   | (e#es) ⇒ (case ts of
                                 Bot ⇒ Bot
                               | - ⇒ check 𝒞 es (check-single 𝒞 e ts)))


| check-single 𝒞 (C v) ts = type-update ts [] (Type [typeof v])
| check-single 𝒞 (Unop-i t -) ts = (if is-int-t t
```

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

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

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

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

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

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

| *check-single* $\mathcal{C}$ *(If (tn -> tm) es1 es2) ts = (if (b-e-type-checker* ($\mathcal{C}$⦇*label := ([tm]*

```
@ (label 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]))
(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]))
(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]))
(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
                                   (tn -> tm) ⇒ type-update ts (to-ct-list tn) (Type
tm))
                                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
                                         (tn -> tm) ⇒ type-update ts (to-ct-list
(tn@[T-i32])) (Type tm))
                                    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-update ts [TSome ((local 𝒞)!i)] (Type [])
                                  else Bot)

| check-single 𝒞 (Tee-local i) ts = (if i < length (local 𝒞)
                               then type-update ts [TSome ((local 𝒞)!i)] (Type [(local
𝒞)!i])
                              else Bot)
```

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

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

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

| *check-single* $\mathcal{C}$ (*Store t tp a off*) *ts* = (*if* (*memory* $\mathcal{C}$) ≠ *None* ∧ *load-store-t-bounds*
a tp t
then type-update ts [*TSome T-i32*,*TSome t*]
(*Type* [])
else Bot)

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

| *check-single* $\mathcal{C}$ *Grow-memory ts* = (*if* (*memory* $\mathcal{C}$) ≠ *None*
then type-update ts [*TSome T-i32*] (*Type* [*T-i32*])
else Bot)

end

10 Correctness of Type Checker

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

10.1 Soundness

lemma *b-e-check-single-type-sound*:
assumes *type-update* (*Type x1*) (*to-ct-list t-in*) (*Type t-out*) = *Type x2*
c-types-agree (*Type x2*) *tm*
$\mathcal{C}$ ⊢ [*e*] : (*t-in* -> *t-out*)
shows ∃ *tn*. *c-types-agree* (*Type x1*) *tn* ∧ $\mathcal{C}$ ⊢ [*e*] : (*tn* -> *tm*)
using *assms*(*2*) *b-e-typing.intros*(*35*)[*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.intros*(*35*)[*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* $\neq$ *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 $\neq$ *Bot*

$ts' \neq Bot$
c-types-agree ts' *tm*
$\mathcal{C} \vdash [e] : (t\text{-}in \text{ -> } t\text{-}out)$
shows $\exists\, tn.$ *c-types-agree ts tn* $\wedge\ \mathcal{C} \vdash [e] : (tn \text{ -> } tm)$
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*) $\vee$ *e* = (*Testop t uv*)) $\wedge$ *is-int-t t*)
$\vee$ (*e* = (*Unop-f t uw*) $\wedge$ *is-float-t t*)
$\vee$ (*e* = (*Cvtop t1 Convert t sx*) $\wedge$ *convert-cond t1 t sx*)
$\vee$ (*e* = (*Cvtop t1 Reinterpret t sx*) $\wedge$ ((*t1* $\neq$ *t*) $\wedge$ *t-length t1* = *t-length t* $\wedge$ *sx* = *None*))
c-types-agree tm' tm
$tn' \neq Bot$
$tm' \neq Bot$
shows $\exists\, tn.$ *c-types-agree tn' tn* $\wedge\ \mathcal{C} \vdash [e] : (tn \text{ -> } tm)$
proof −
have (*e* = (*Cvtop t1 Convert t sx*) $\Longrightarrow$ *convert-cond t1 t sx*)
using *assms*(*2*)
by *simp*
hence *temp0*:(*e* = (*Cvtop t1 Convert t sx*)) $\Longrightarrow$ (*type-update tn'* [*TSome t*] (*Type* [*arity-1-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-update tn'* [*TSome t*] (*Type* [*arity-1-result e*]) = *tm'*)

using *assms*(*1*,*2*,*5*) *arity-1-result-def*
by *simp*
have *1*:*type-update tn′* (*to-ct-list* [*t*]) (*Type* [*arity-1-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*) *temp0 temp1 b-e.simps*(*978*,*979*,*982*) *check-single.simps*(*2*) *check-single.simps*(*3*) *check-single.simps*(*6*) *type-update.simps*)
done
have $\mathcal{C}$ ⊢ [*e*] : ([*t*] -> [*arity-1-result e*])
using *assms*(*2*) *b-e-typing.intros*(*2*,*3*,*6*,*9*,*10*)
unfolding *arity-1-result-def*
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-i t iop* ∧ *is-int-t t*)
∨ (*e* = *Binop-f t fop* ∧ *is-float-t t*)
∨ (*e* = *Relop-i t irop* ∧ *is-int-t t*)
∨ (*e* = *Relop-f t frop* ∧ *is-float-t t*))
c-types-agree tm′ tm
tn′ ≠ *Bot*
tm′ ≠ *Bot*
shows ∃ *tn*. *c-types-agree tn′ tn* ∧ $\mathcal{C}$ ⊢ [*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 $\mathcal{C}$ ⊢ [*e*] : ([*t*,*t*] -> [*arity-2-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 $\mathcal{C}$ ⊢ *es* : (*tn* -> *tm*)
proof −
fix *e tn′*

```
have b-e-type-checker 𝒞 es (tn -> tm) ⟹
      𝒞 ⊢ es : (tn -> tm)
and ⋀tm' tm.
    check 𝒞 es tn' = tm' ⟹
    c-types-agree tm' tm ⟹
     ∃ tn. c-types-agree tn' tn ∧ 𝒞 ⊢ es : (tn -> tm)
and ⋀tm' tm.
    check-single 𝒞 e tn' = tm' ⟹
    c-types-agree tm' tm ⟹
    tn' ≠ Bot ⟹
    tm' ≠ Bot ⟹
     ∃ tn. c-types-agree tn' tn ∧ 𝒞 ⊢ [e] : (tn -> tm)
proof (induction rule: b-e-type-checker-check-check-single.induct)
  case (1 𝒞 es tn' tm)
  thus ?case
    by simp
next
  case (2 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
    by (simp, meson)
```

thus *?case*
  using *b-e-check-single-sound-binop-relop 6*
  by *fastforce*
next
  case (*7* $\mathcal{C}$ *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* $\mathcal{C}$ *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* $\mathcal{C}$ *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* $\mathcal{C}$ *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* $\mathcal{C}$ *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* $\mathcal{C}$ *t1 t2 sx ts*)
  hence $t1 \neq t2 \wedge$ *t-length t1* $=$ *t-length t2* $\wedge$ *sx* $=$ *None*
    by (*simp*, *presburger*)
  thus *?case*
    using *b-e-check-single-sound-unop-testop-cvtop 12*
    by *fastforce*
next
  case (*13* $\mathcal{C}$ *ts*)
  thus *?case*
    using *b-e-typing.intros*(*11*) *c-types-agree-not-bot-exists*

by *blast*
next
case (*14 𝒞 ts*)
thus *?case*
using *b-e-typing.intros*(*12*,*35*)
by *fastforce*
next
case (*15 𝒞 ts*)
thus *?case*
proof (*cases ts*)
case (*TopType x1*)
thus *?thesis*
proof (*cases x1 rule*: *List.rev-cases*)
case *Nil*
have $\mathcal{C} \vdash$ [*Drop*] : (*tm*@[*T-i32*] -> *tm*)
using *b-e-typing.intros*(*13*,*35*)
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 *15 TopType type-update-empty*
by (*metis check-single.simps*(*13*))
hence *temp2*:*c-types-agree* (*TopType ys*) *tm*
using *consume-top-geq*[*OF temp1*] *15*(*2*,*3*,*4*)
by (*metis Suc-leI add-diff-cancel-right′ append-eq-conv-conj consume.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*(*13*,*35*)
by *fastforce*
qed
next
case (*Type x2*)
thus *?thesis*
proof (*cases x2 rule*: *List.rev-cases*)
case *Nil*
hence (*consume* (*Type* []) [*TAny*]) = *tm′*
using *15 Type type-update-empty*

by *fastforce*
thus *?thesis*
using *15(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 *15 Type type-update-empty*
by (*metis check-single.simps*(*13*))
hence *temp2*:*c-types-agree* (*Type ys*) *tm*
using *15*(*2*,*3*,*4*) *ct-suffix-def*
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*(*13*,*35*)
by *fastforce*
qed
qed *simp*
next
case (*16 $\mathcal{C}$ 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* ≥ *3*
by *linarith*
thus *?thesis*
proof (*cases*)
case *1*
hence *tm′* = *TopType* [*TAny*]
using *TopType 16*
by *simp*
then obtain *t″ tm″* **where** *tm-def*:*tm* = *tm″*@[*t″*]
using *16*(*2*) *ct-suffix-def*
by (*simp*, *metis Nil-is-append-conv append-butlast-last-id checker-type.inject*(*1*)
ct-prefixI ct-prefix-nil(*2*) *produce.simps*(*1*) *produce-nil*)
have $\mathcal{C}$ ⊢ [*Select*] : ([*t″*,*t″*,*T-i32*] -> [*t″*])
using *b-e-typing.intros*(*14*)
by *blast*

```
      thus ?thesis
        using TopType 16 1 tm-def b-e-typing.intros(35) c-types-agree.simps(2)
c-types-agree-top1
        by fastforce
    next
      case 2
      have type-update-select (TopType x1) = tm′
        using 16 TopType
        unfolding check-single.simps
        by simp
      hence x1-def:ct-list-eq x1 [TSome T-i32] tm′ = TopType [TAny]
        using type-update-select-length1[OF - 2 16(4)]
        by simp-all
      then obtain t′′ tm′′ where tm-def:tm = tm′′@[t′′]
        using 16(2) ct-suffix-def
        by (metis Nil-is-append-conv append-butlast-last-id c-types-agree.simps(2)
ct-prefixI
              ct-prefix-nil(2) list.simps(8) to-ct-list-def)
      have c-types-agree (TopType x1) ((tm′′@[t′′,t′′])@[T-i32])
        using x1-def(1)
        by (metis c-types-agree-top2 list.simps(8,9) to-ct-list-def)
      thus ?thesis
        using TopType b-e-typing.intros(14,35) tm-def
        by auto
    next
      case 3
      have type-update-select (TopType x1) = tm′
        using 16 TopType
        unfolding check-single.simps
        by simp
      then obtain ct1 ct2 where x1-def:x1 = [ct1, ct2]
                                ct-eq ct2 (TSome T-i32)
                                tm′ = TopType [ct1]
        using type-update-select-length2[OF - 3 16(4)]
        by blast
      then obtain t′′ tm′′ where tm-def:tm = tm′′@[t′′]
                                 ct-list-eq [ct1] [(TSome t′′)]
        using 16(2) c-types-agree-imp-ct-list-eq[of [ct1] tm]
          by (metis append-Nil2 append-butlast-last-id append-eq-append-conv-if
append-eq-conv-conj
              ct-list-eq-length diff-Suc-1 length-Cons length-butlast length-map
              list.simps(8,9) list.size(3) nat.distinct(2) to-ct-list-def)
      hence ct-list-eq x1 (to-ct-list [ t′′, T-i32])
        using x1-def(1,2)
        unfolding ct-list-eq-def to-ct-list-def
        by fastforce
      hence c-types-agree (TopType x1) ((tm′′@[t′′])@[t′′,T-i32])
        using c-types-agree-top2
        by blast
```

```
    thus ?thesis
      using TopType b-e-typing.intros(14,35) tm-def
      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 (TopType x1) = tm′
      using 16 TopType
      by simp
    then obtain tm-int where (select-return-top x1
                              (x1 ! (length x1 − 2))
                              (x1 ! (length x1 − 3))) = tm-int
                             tm-int ≠ Bot
      using nat-def 16(4)
      unfolding type-update-select.simps
      by fastforce
    then obtain x2 where x2-def:(select-return-top x1
                                (x1 ! (length x1 − 2))
                                (x1 ! (length x1 − 3))) = TopType x2
      using select-return-top-exists
      by fastforce
     have ct-suffix x1 [TAny, TAny, TSome T-i32] ∨ ct-suffix [TAny, TAny,
TSome T-i32] x1
      using tm′-def nat-def 16(4)
      by (simp, metis (full-types) produce.simps(6))
    hence tm′-eq:tm′ = TopType x2
      using tm′-def nat-def 16(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)
      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 16(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])
    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
  thus ?thesis
    using b-e-typing.intros(14,35) tm-def
    by auto
 qed
next

 case (Type x2)
 hence x2-cond:(length x2 ≥ 3 ∧ (x2!(length x2−2)) = (x2!(length x2−3)))
   using 16
   by (simp, meson)
 hence tm'-def:consume (Type x2) [TAny, TSome T-i32] = tm'
   using 16 Type
   by simp
 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] (to-ct-list (ts' @ [t1, t2, t3]))
   using tm'-def 16(4)
   by (simp, metis cts-def2)
 hence tm'-def:tm' = Type (ts'@[t1])
   using tm'-def 16(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]
   using ts'-suffix
   unfolding ct-suffix-def to-ct-list-def
   by fastforce
 hence t3 = T-i32
   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])
        using cts-def2
        by simp
      thus ?thesis
        using b-e-typing.intros(14,35) Type tm'-def 16(2)
        by fastforce
    qed simp
  next
    case (17 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 17
      by (simp, meson)
    hence 𝒞 ⊢ [Block (tn'' -> tm'') es] : (tn'' -> tm'')
      using b-e-typing.intros(15)[OF - 17(1)]
      by blast
    ultimately
    show ?case
      using b-e-check-single-type-not-bot-sound[OF - 17(4,5,3)]
      by blast
  next
    case (18 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 18
      by (simp, meson)
    hence 𝒞 ⊢ [Loop (tn'' -> tm'') es] : (tn'' -> tm'')
      using b-e-typing.intros(16)[OF - 18(1)]
      by blast
    ultimately
    show ?case
      using b-e-check-single-type-not-bot-sound[OF - 18(4,5,3)]
      by blast
  next
    case (19 𝒞 tn'' tm'' es1 es2 ts)
    hence type-update ts (to-ct-list (tn''@[T-i32])) (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 19
      by (simp, meson)+
```

hence $\mathcal{C} \vdash$ [*If* (*tn''* -> *tm''*) *es1 es2*] : (*tn''*@[*T-i32*] -> *tm''*)
using *b-e-typing.intros*(*17*)[*OF* - *19*(*1*,*2*)]
by *blast*
ultimately
show *?case*
using *b-e-check-single-type-not-bot-sound*[*OF* - *19*(*5*,*6*,*4*)]
by *blast*
next
case (*20* $\mathcal{C}$ *i ts*)
hence *type-update ts* (*to-ct-list* ((*label* $\mathcal{C}$)!*i*)) (*TopType* []) = *tm'*
by *auto*
moreover
have *i* < *length* (*label* $\mathcal{C}$)
using *20*
by (*simp*, *meson*)
ultimately
show *?case*
using *b-e-check-single-top-not-bot-sound*[*OF* - *20*(*3*,*4*)]
b-e-typing.intros(*18*)
b-e-typing.intros(*35*)
by (*metis suffix-def*)
next
case (*21* $\mathcal{C}$ *i ts*)
hence *type-update ts* (*to-ct-list* ((*label* $\mathcal{C}$)!*i* @ [*T-i32*])) (*Type* ((*label* $\mathcal{C}$)!*i*)) = *tm'*
by *auto*
moreover
have *i* < *length* (*label* $\mathcal{C}$)
using *21*
by (*simp*, *meson*)
hence $\mathcal{C} \vdash$ [*Br-if i*] : ((*label* $\mathcal{C}$)!*i* @ [*T-i32*] -> (*label* $\mathcal{C}$)!*i*)
using *b-e-typing.intros*(*19*)
by *fastforce*
ultimately
show *?case*
using *b-e-check-single-type-not-bot-sound*[*OF* - *21*(*3*,*4*,*2*)]
by *fastforce*
next
case (*22* $\mathcal{C}$ *is i ts*)
then obtain *tls* **where** *tls-def*:(*same-lab* (*is*@[*i*]) (*label* $\mathcal{C}$)) = *Some tls*
by *fastforce*
hence *type-update ts* (*to-ct-list* (*tls* @ [*T-i32*])) (*TopType* []) = *tm'*
using *22*
by *simp*
thus *?case*
using *b-e-check-single-top-not-bot-sound*[*OF* - *22*(*3*,*4*)]
b-e-typing.intros(*20*)[*OF same-lab-conv-list-all*[*OF tls-def*]]
b-e-typing.intros(*35*)
by (*metis suffix-def*)

next
  case (*23 $\mathcal{C}$ ts*)
  then obtain *ts-r* **where** (*return* $\mathcal{C}$) = *Some ts-r*
    by *fastforce*
  moreover
  hence *type-update ts* (*to-ct-list ts-r*) (*TopType* []) = *tm′*
    using *23*
    by *simp*
  ultimately
  show *?case*
    using *b-e-check-single-top-not-bot-sound*[*OF - 23*(*3*,*4*)]
          b-e-typing.intros(*21*,*35*)
    by (*metis suffix-def*)
next
  case (*24 $\mathcal{C}$ i ts*)
  obtain *tn″ tm″* **where** *func-def*:(*func-t* $\mathcal{C}$)!*i* = (*tn″* -> *tm″*)
    using *tf.exhaust*
    by *blast*
  hence *type-update ts* (*to-ct-list tn″*) (*Type tm″*) = *tm′*
    using *24*
    by *auto*
  moreover
  have $i <$ *length* (*func-t* $\mathcal{C}$)
    using *24*
    by (*simp*, *meson*)
  hence $\mathcal{C} \vdash$ [*Call i*] : (*tn″* -> *tm″*)
    using *b-e-typing.intros*(*22*) *func-def*
    by *fastforce*
  ultimately
  show *?case*
    using *b-e-check-single-type-not-bot-sound*[*OF - 24*(*3*,*4*,*2*)]
    by *fastforce*
next
  case (*25 $\mathcal{C}$ i ts*)
  obtain *tn″ tm″* **where** *type-def*:(*types-t* $\mathcal{C}$)!*i* = (*tn″* -> *tm″*)
    using *tf.exhaust*
    by *blast*
  hence *type-update ts* (*to-ct-list* (*tn″*@[*T-i32*])) (*Type tm″*) = *tm′*
    using *25*
    by *auto*
  moreover
  have (*table* $\mathcal{C}$) $\neq$ *None* $\wedge$ $i <$ *length* (*types-t* $\mathcal{C}$)
    using *25*
    by (*simp*, *meson*)
  hence $\mathcal{C} \vdash$ [*Call-indirect i*] : (*tn″*@[*T-i32*] -> *tm″*)
    using *b-e-typing.intros*(*23*) *type-def*
    by *fastforce*
  ultimately
  show *?case*

using *b-e-check-single-type-not-bot-sound*[*OF* - *25*(*3*,*4*,*2*)]
by *fastforce*
next
case (*26* $\mathcal{C}$ *i ts*)
hence *type-update ts* [] (*Type* [(*local* $\mathcal{C}$)!*i*]) = *tm'*
by *auto*
moreover
have *i* < *length* (*local* $\mathcal{C}$)
using *26*
by (*simp*, *meson*)
hence $\mathcal{C}$ ⊢ [*Get-local i*] : ([] -> [(*local* $\mathcal{C}$)!*i*])
using *b-e-typing.intros*(*24*)
by *fastforce*
ultimately
show *?case*
using *b-e-check-single-type-not-bot-sound*[*OF* - *26*(*3*,*4*,*2*)]
unfolding *to-ct-list-def*
by (*metis list.map-disc-iff*)
next
case (*27* $\mathcal{C}$ *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 *27*
by (*simp*, *meson*)
hence $\mathcal{C}$ ⊢ [*Set-local i*] : ([(*local* $\mathcal{C}$)!*i*] -> [])
using *b-e-typing.intros*(*25*)
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* (*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 *28*
by (*simp*, *meson*)
hence $\mathcal{C}$ ⊢ [*Tee-local i*] : ([(*local* $\mathcal{C}$)!*i*] -> [(*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*)]

by *fastforce*
next
case (*29* $\mathcal{C}$ *i ts*)
hence *type-update ts* [] (*Type* [*tg-t* ((*global* $\mathcal{C}$)!*i*)]) = *tm*′
by *auto*
moreover
have *i* < *length* (*global* $\mathcal{C}$)
using *29*
by (*simp*, *meson*)
hence $\mathcal{C}$ ⊢ [*Get-global i*] : ([] -> [*tg-t* ((*global* $\mathcal{C}$)!*i*)])
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*)]
unfolding *to-ct-list-def*
by (*metis list.map-disc-iff*)
next
case (*30* $\mathcal{C}$ *i ts*)
hence *type-update ts* (*to-ct-list* [*tg-t* ((*global* $\mathcal{C}$)!*i*)]) (*Type* []) = *tm*′
unfolding *to-ct-list-def*
by *auto*
moreover
have *i* < *length* (*global* $\mathcal{C}$) ∧ *is-mut* (*global* $\mathcal{C}$! *i*)
using *30*
by (*simp*, *meson*)
then obtain *t* **where** (*global* $\mathcal{C}$! *i*) = (|*tg-mut* = *T-mut*, *tg-t* = *t*|) *i* < *length* (*global* $\mathcal{C}$)
unfolding *is-mut-def*
by (*cases global* $\mathcal{C}$! *i*, *auto*)
hence $\mathcal{C}$ ⊢ [*Set-global i*] : ([*tg-t* (*global* $\mathcal{C}$! *i*)] -> [])
using *b-e-typing.intros*(*28*)[*of i* $\mathcal{C}$ *tg-t* (*global* $\mathcal{C}$! *i*)]
unfolding *is-mut-def tg-t-def*
by *fastforce*
ultimately
show *?case*
using *b-e-check-single-type-not-bot-sound*[*OF - 30*(*3*,*4*,*2*)]
by *fastforce*
next
case (*31* $\mathcal{C}$ *t tp-sx a off ts*)
hence *type-update ts* (*to-ct-list* [*T-i32*]) (*Type* [*t*]) = *tm*′
unfolding *to-ct-list-def*
by *auto*
moreover
have (*memory* $\mathcal{C}$) ≠ *None* ∧ *load-store-t-bounds a* (*option-projl tp-sx*) *t*
using *31*
by (*simp*, *meson*)
hence $\mathcal{C}$ ⊢ [*Load t tp-sx a off*] : ([*T-i32*] -> [*t*])
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*)]
by *fastforce*
next
case (*32* $\mathcal{C}$ *t tp a off ts*)
hence *type-update ts* (*to-ct-list* [*T-i32*,*t*]) (*Type* []) = *tm'*
unfolding *to-ct-list-def*
by *auto*
moreover
have (*memory* $\mathcal{C}$) ≠ *None* ∧ *load-store-t-bounds a tp t*
using *32*
by (*simp*, *meson*)
hence $\mathcal{C}$ ⊢ [*Store t tp a off*] : ([*T-i32*,*t*] -> [])
using *b-e-typing.intros*(*30*)
by *fastforce*
ultimately
show *?case*
using *b-e-check-single-type-not-bot-sound*[*OF - 32*(*3*,*4*,*2*)]
by *fastforce*
next
case (*33* $\mathcal{C}$ *ts*)
hence *type-update ts* [] (*Type* [*T-i32*]) = *tm'*
by *auto*
moreover
have *memory* $\mathcal{C}$ ≠ *None*
using *33*
by (*simp*, *meson*)
hence $\mathcal{C}$ ⊢ [*Current-memory*] : ([] -> [*T-i32*])
using *b-e-typing.intros*(*31*)
by *fastforce*
ultimately
show *?case*
using *b-e-check-single-type-not-bot-sound*[*OF - 33*(*3*,*4*,*2*)]
unfolding *to-ct-list-def*
by (*metis list.map-disc-iff*)
next
case (*34* $\mathcal{C}$ *ts*)
hence *type-update ts* (*to-ct-list* [*T-i32*]) (*Type* [*T-i32*]) = *tm'*
unfolding *to-ct-list-def*
by *auto*
moreover
have *memory* $\mathcal{C}$ ≠ *None*
using *34*
by (*simp*, *meson*)
hence $\mathcal{C}$ ⊢ [*Grow-memory*] : ([*T-i32*] -> [*T-i32*])
using *b-e-typing.intros*(*32*)
by *fastforce*

ultimately
show *?case*
using *b-e-check-single-type-not-bot-sound*[*OF - 34*(*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*
$\vee$ *check-single* $\mathcal{C}$ *e* = (λ*ctn*. *type-update-select ctn*)
$\vee$ ($\exists$ *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
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*)
thus *?case*
by *simp*
next
case (*3* $\mathcal{C}$ *v ts*)
thus *?case*
by *fastforce*
next
case (*4* $\mathcal{C}$ *t uu ts*)
thus *?case*
by (*simp*, *meson assms*(*2*) *type-update.simps*)
next
case (*5* $\mathcal{C}$ *t uv ts*)
thus *?case*
by (*simp*, *meson assms*(*2*) *type-update.simps*)
next
case (*6* $\mathcal{C}$ *t uw ts*)
thus *?case*
by (*simp*, *meson assms*(*2*) *type-update.simps*)
next

```
  case (7 𝒞 t ux ts)
  thus ?case
    by (simp, meson assms(2) type-update.simps)
next
  case (8 𝒞 t uy ts)
  thus ?case
    by (simp, meson assms(2) type-update.simps)
next
  case (9 𝒞 t uz ts)
  thus ?case
    by (simp, meson assms(2) type-update.simps)
next
  case (10 𝒞 t va ts)
  thus ?case
    by (simp, meson assms(2) type-update.simps)
next
  case (11 𝒞 t1 t2 sx ts)
  thus ?case
    by (simp del: convert-cond.simps, meson assms(2) type-update.simps)
next
  case (12 𝒞 t1 t2 sx ts)
  thus ?case
    by (simp, meson assms(2) type-update.simps)
next
  case (13 𝒞 ts)
  thus ?case
    by fastforce
next
  case (14 𝒞 ts)
  thus ?case
    by fastforce
next
  case (15 𝒞 ts)
  thus ?case
    by fastforce
next
  case (16 𝒞 ts)
  thus ?case
    by fastforce
next
  case (17 𝒞 tn tm es ts)
  thus ?case
    by (simp, meson assms(2) type-update.simps)
next
  case (18 𝒞 tn tm es ts)
  thus ?case
    by (simp, meson assms(2) type-update.simps)
next
  case (19 𝒞 tn tm es1 es2 ts)
```

```
  thus ?case
    by (simp, meson assms(2) type-update.simps)
next
  case (20 𝒞 i ts)
  thus ?case
    by (simp, meson assms(2) type-update.simps)
next
  case (21 𝒞 i ts)
  thus ?case
    by (simp, meson assms(2) type-update.simps)
next
  case (22 𝒞 is ts)
  thus ?case
    by (simp, metis assms(2) option.case-eq-if type-update.simps)
next
  case (23 𝒞 ts)
  thus ?case
    by (simp, metis assms(2) option.case-eq-if type-update.simps)
next
  case (24 𝒞 i ts)
  then show ?case
   by (simp, metis (no-types, lifting) assms(2) tf.case tf.exhaust type-update.simps)
next
  case (25 𝒞 i ts)
  thus ?case
   by (simp, metis (no-types, lifting) assms(2) tf.case tf.exhaust type-update.simps)
next
  case (26 𝒞 i ts)
  thus ?case
    by (simp, meson assms(2) type-update.simps)
next
  case (27 𝒞 i ts)
  thus ?case
    by (simp, meson assms(2) type-update.simps)
next
  case (28 𝒞 i ts)
  thus ?case
    by (simp, meson assms(2) type-update.simps)
next
  case (29 𝒞 i ts)
  thus ?case
    by (simp, meson assms(2) type-update.simps)
next
  case (30 𝒞 i ts)
  thus ?case
    by (simp, meson assms(2) type-update.simps)
next
  case (31 𝒞 t tp-sx a off ts)
  thus ?case
```

by (*simp*, *meson assms*(*2*) *type-update.simps*)
next
case (*32* $\mathcal{C}$ *t tp a off ts*)
thus *?case*
by (*simp*, *meson assms*(*2*) *type-update.simps*)
next
case (*33* $\mathcal{C}$ *ts*)
thus *?case*
by (*simp*, *meson assms*(*2*) *type-update.simps*)
next
case (*34* $\mathcal{C}$ *ts*)
thus *?case*
by (*simp*, *meson assms*(*2*) *type-update.simps*)
qed
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* | - $\Rightarrow$ *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* $\mathcal{C}$ (*es*@[*e*]) *ts* $\neq$ *Bot*
shows *check* $\mathcal{C}$ *es ts* $\neq$ *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 $check\ \mathcal{C}\ es\ ts \neq Bot$
shows $check\ \mathcal{C}\ (es@[e])\ ts = check\text{-}single\ \mathcal{C}\ e\ (check\ \mathcal{C}\ es\ ts)$
proof −
obtain *f* **where** *f-def*:$f = (\lambda\ e\ ts.\ (case\ ts\ of\ Bot \Rightarrow Bot\ |\ \text{-} \Rightarrow check\text{-}single\ \mathcal{C}\ e\ ts))$
by *blast*
have *f-simp*:$\bigwedge ts.\ ts \neq Bot \Longrightarrow (f\ e\ ts = check\text{-}single\ \mathcal{C}\ e\ ts)$
proof −
fix *ts*
show $ts \neq Bot \Longrightarrow (f\ e\ ts = check\text{-}single\ \mathcal{C}\ e\ ts)$
using *f-def*
by (*cases ts*, *simp-all*)
qed
have $check\ \mathcal{C}\ (es@[e])\ ts = foldl\ (\lambda\ ts\ e.\ (case\ ts\ of\ Bot \Rightarrow Bot\ |\ \text{-} \Rightarrow check\text{-}single\ \mathcal{C}\ e\ ts))\ ts\ (es@[e])$
using *check-equiv-fold*
by *simp*
also
have $... = foldr\ (\lambda\ e\ ts.\ (case\ ts\ of\ Bot \Rightarrow Bot\ |\ \text{-} \Rightarrow check\text{-}single\ \mathcal{C}\ e\ ts))\ (rev\ (es@[e]))\ ts$
using *foldl-conv-foldr*
by *fastforce*
also
have $... = f\ e\ (foldr\ (\lambda\ e\ ts.\ (case\ ts\ of\ Bot \Rightarrow Bot\ |\ \text{-} \Rightarrow check\text{-}single\ \mathcal{C}\ e\ ts))\ (rev\ es)\ ts)$
using *f-def*
by *simp*
also
have $... = f\ e\ (check\ \mathcal{C}\ es\ ts)$
using *foldr-conv-foldl*[*of* - (*rev es*) *ts*] *rev-rev-ident*[*of es*] *check-equiv-fold*
by *simp*
also
have $... = check\text{-}single\ \mathcal{C}\ e\ (check\ \mathcal{C}\ es\ ts)$
using *assms f-simp*
by *simp*
finally
show *?thesis* **.**
qed

lemma *check-single-imp-weakening*:
  assumes *check-single* $\mathcal{C}$ *e* (*Type t1s*) = *ctm*
    ctm ≠ *Bot*
    c-types-agree ctn t1s
    c-types-agree ctm t2s
  shows ∃ *ctm'*. *check-single* $\mathcal{C}$ *e ctn* = *ctm'* ∧ *c-types-agree ctm' t2s*
proof −
  consider (*1*) *check-single* $\mathcal{C}$ *e* = *id*
    | (*2*) *check-single* $\mathcal{C}$ *e* = (λ*ctn. type-update-select ctn*)
    | (*3*) (∃ *cons prods*. (*check-single* $\mathcal{C}$ *e* = (λ*ctn. 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*
    note *outer-2* = *2*
    hence *t1s-cond*:(*length t1s* ≥ *3* ∧ (*t1s*!(*length t1s*−*2*)) = (*t1s*!(*length t1s*−*3*)))
      using *assms*(*1*,*2*)
      by (*simp*, *meson*)
    hence *ctm-def*:*ctm* = *consume* (*Type t1s*) [*TAny*, *TSome T-i32*]
      using *assms*(*1*,*2*) *2*
      by *simp*
    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*] (*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*] (*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*]
      using *type-update-select-type-length3 assms*(*1*) *c-t-def outer-2*
      by *fastforce*
    hence *t2s-def*:*t2s* = *t1s'*@[*t*]

using *ctm-def c-t-def t2s-eq t1s-suffix assms*(*2*) *t1s-suffix-full*
by *simp*
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* ≥ *3*
by *linarith*
thus *?thesis*
proof (*cases*)
case *1*
hence *check-single* $\mathcal{C}$ *e ctn* = *TopType* [*TAny*]
using *2 TopType*
by *simp*
thus *?thesis*
using *ct-suffix-singleton to-ct-list-def t2s-len*
by *auto*
next
case *2*
hence *ct-suffix* [*TSome T-i32*] *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* $\mathcal{C}$ *e ctn* = *TopType* [*TAny*]
using *outer-2 TopType 2*
by *simp*
thus *?thesis*
using *t2s-len ct-suffix-singleton*
by (*simp add*: *to-ct-list-def*)
next
case *3*
have *ct-list-eq* (*to-ct-list t1s*) (*to-ct-list* (*t1s′* @ [*t*, *t*, *T-i32*]))
using *t1s-suffix2*
by (*simp add*: *ct-list-eq-ts-conv-eq*)
hence *temp1*:*to-ct-list t1s* = (*to-ct-list* (*t1s′* @ [*t*])) @ (*to-ct-list* [*t*, *T-i32*])
using *t1s-suffix2 to-ct-list-def*
by *simp*
hence *ct-suffix* (*to-ct-list* [*t*, *T-i32*]) (*to-ct-list t1s*)
using *ct-suffix-def*[*of* (*to-ct-list* [*t*, *T-i32*]) (*to-ct-list t1s*)]
by (*simp add*: *ct-suffix-cons-it*)
hence *ct-suffix* (*to-ct-list* [*t*, *T-i32*]) *x1*
using *assms*(*3*) *TopType 3*
by (*simp*, *metis temp1 append-Nil ct-suffix-cons2 ct-suffix-def length-Cons length-map*

list.size(*3*) *numeral-2-eq-2 to-ct-list-def*)
hence *temp3*:*ct-list-eq* (*to-ct-list* [*t*, *T-i32*]) *x1*
using *3 ct-suffix-imp-ct-list-eq*
unfolding *to-ct-list-def*
by (*metis Suc-leI ct-list-eq-commute diff-is-0-eq drop-0 length-Cons length-map lessI*
list.size(*3*) *numeral-2-eq-2*)
hence *temp4*:*ct-suffix* [*TSome T-i32*] *x1*
using *ct-suffix-less*[*of* [*TSome t*] [*TSome T-i32*] *x1*]
ct-suffix-extend-ct-list-eq[*of* [] []] *ct-suffix-nil*
unfolding *to-ct-list-def*
by *fastforce*
hence *ct-suffix* (*take 1 x1*) (*to-ct-list* [*t*])
using *temp3 ct-suffix-nil ct-list-eq-commute ct-suffix-extend-ct-list-eq*[*of* [] [] (*take 1 x1*) (*to-ct-list* [*t*])]
unfolding *to-ct-list-def*
by (*simp*, *metis butlast.simps*(*2*) *butlast-conv-take ct-list-eq-take diff-Suc-1 length-Cons*
list.distinct(*1*) *list.size*(*3*))
thus *?thesis*
using *TopType 2 3 ct-suffix-nil temp3 temp4 t2s-def to-ct-list-def*
apply (*simp*, *safe*)
apply (*metis append.assoc ct-suffix-def*)
done
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*))
obtain *x1' x x' x''* **where** *x1-split*:*x1* = *x1'*@[*x*,*x'*,*x''*]
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 *a*:*ct-suffix* (*x1'*@[*x*,*x'*,*x''*]) (*to-ct-list* (*t1s'* @ [*t*, *t*, *T-i32*]))
using *t1s-suffix2 assms*(*3*) *TopType*
by *simp*
hence *b*:*ct-suffix* (*x1'*@[*x*,*x'*]) (*to-ct-list* (*t1s'* @ [*t*, *t*])) ∧ (*ct-eq x''* (*TSome T-i32*))

using *to-ct-list-def ct-suffix-unfold-one*[*of* (*x1'*@[*x*,*x'*]) *x''* *to-ct-list* (*t1s'* @ [*t*, *t*])]
by *fastforce*
hence *c*:*ct-suffix* (*x1'*@[*x*]) (*to-ct-list* (*t1s'* @ [*t*])) ∧ (*ct-eq* *x'* (*TSome* *t*))
using *to-ct-list-def ct-suffix-unfold-one*[*of* (*x1'*@[*x*]) *x'* *to-ct-list* (*t1s'* @ [*t*])]
by *fastforce*
hence *d*:*ct-suffix* *x1'* (*to-ct-list* *t1s'*) ∧ (*ct-eq* *x* (*TSome* *t*))
using *to-ct-list-def ct-suffix-unfold-one*[*of* (*x1'*) *x* *to-ct-list* (*t1s'*)]
by *fastforce*
have (*take* (*length* *x1* − *3*) *x1*) = *x1'*
using *x1-split*
by *simp*
have *x'-ind*:(*x1*!(*length* *x1*−*2*)) = *x'*
using *x1-split List.nth-append-length*[*of* *x1'* @ [*x*]]
by *simp*
have *x-ind*:(*x1*!(*length* *x1*−*3*)) = *x*
using *x1-split*
by *simp*
have *ct-suffix* [*TSome* *T-i32*] *x1*
using *b x1-split ct-suffix-def ct-list-eq-def ct-suffixI*[*of* *x1* *x1'* @ [*x*, *x'*]]
by *simp*
hence *check-single* 𝒞 *e* (*TopType* *x1*) = (*select-return-top* *x1* (*x1*!(*length* *x1*−*2*)) (*x1*!(*length* *x1*−*3*)))
using *type-update-select-conv-select-return-top*[*OF* - *4*]
unfolding *2*
by *blast*
moreover
have ... = (*TopType* (*x1'*@[*x*])) ∨ ... = (*TopType* (*x1'*@[*x'*]))
apply (*cases* *x*; *cases* *x'*)
using *x1-split 4 nat-def 2 x-ind x'-ind c d*
by *simp-all*
moreover
have *ct-suffix* (*x1'*@[*x*]) (*to-ct-list* *t2s*)
by (*simp add*: *c t2s-def*)
moreover
have *ct-suffix* (*x1'*@[*x'*]) (*to-ct-list* *t2s*)
using *ct-suffix-unfold-one*[*symmetric*, *of* *x'* (*TSome* *t*) *x1'* (*to-ct-list* *t1s'*)] *c d*
t2s-def
unfolding *to-ct-list-def*
by *fastforce*
ultimately
show *?thesis*
using *TopType*
by *auto*
qed
qed *simp-all*
next

case *3*
then obtain *cons prods* **where** *c-s-def*:*check-single* $\mathcal{C}$ *e* = (λ*ctn. 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* $\mathcal{C}$ *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 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 𝒞 es (t1s -> t2s)
          b-e-type-checker 𝒞 [e] (t2s -> t3s)
  shows b-e-type-checker 𝒞 (es @ [e]) (t1s -> t3s)
proof −
  have c-types-agree (check-single 𝒞 e (Type t2s)) t3s
    using assms(2)
    by simp
```

```
  then obtain ctm where ctm-def:check-single 𝒞 e (Type t2s) = ctm
                             c-types-agree ctm t3s
                             ctm ≠ Bot
    by fastforce
  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
  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) check-single 𝒞 e = (λctn. type-update-select 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 simp
      next
        case 2
        thus ?thesis
          using assms outer-2 TopType produce-type-type
          by fastforce
      next
        case 3
        thus ?thesis
          using assms 2 TopType
          by (simp, metis checker-type.distinct(1) checker-type.distinct(5))
      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
        proof –
          {
             assume a1: produce (if ct-suffix [TAny, TAny, TSome T-i32] x1 then
TopType (take (length x1 − length [TAny, TAny, TSome T-i32]) x1) else if ct-suffix
x1 [TAny, TAny, TSome T-i32] then TopType [] else Bot) (select-return-top x1 (x1
! Suc nat) (x1 ! nat)) = Type tm
            obtain tts :: checker-type ⇒ t list where
              f2: ∀ c. (∀ ca ts. produce c ca ≠ Type ts) ∨ c = Type (tts c)
              using produce-type-type by moura
            then have f3: ⋀ts. ¬ ct-suffix [TAny, TAny, TSome T-i32] x1 ∨ Type
tm ≠ Type ts
              using a1 by fastforce
            then have ⋀ts. ¬ ct-suffix x1 [TAny, TAny, TSome T-i32] ∨ Type tm
≠ Type ts
              using f2 a1 by fastforce
            then have False
              using f3 a1 by fastforce
          }
          thus ?thesis
            using assms 2 TopType nat-def
            by simp
        qed
      qed
    qed simp-all
  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) check-single 𝒞 e = (λctn. type-update-select 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
    hence cond:(length tn ≥ 3 ∧ (tn!(length tn−2)) = (tn!(length tn−3)))
      using assms
      by (simp, metis checker-type.distinct(5))
    hence consume (Type tn) [TAny, TSome T-i32] = (Type tm)
      using assms 2
      by simp
    hence consume (Type (ts@tn)) [TAny, TSome T-i32] = (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
  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* = (*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* $\mathcal{C}$ *e* (*Type tn*) = *TopType tm*
shows *check-single* $\mathcal{C}$ *e* (*Type* (*ts*@*tn*)) = *TopType tm*
proof −
consider (*1*) *check-single* $\mathcal{C}$ *e* = *id*
| (*2*) *check-single* $\mathcal{C}$ *e* = (λ*ctn. type-update-select ctn*)
| (*3*) ($\exists$ *cons prods.* (*check-single* $\mathcal{C}$ *e* = (λ*ctn. 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*
thus *?thesis*
using *assms*
by (*simp*, *metis checker-type.distinct*(*1*) *checker-type.distinct*(*3*) *consume.simps*(*1*))
next
case *3*
then obtain *cons prods* **where** *check-def*:*check-single* $\mathcal{C}$ *e* = (λ*ctn. 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* $\mathcal{C}$ *es* (*Type tn*) = (*Type tm*)
  shows *check* $\mathcal{C}$ *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* $\mathcal{C}$ *e* (*check* $\mathcal{C}$ *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* $\mathcal{C}$ *es Bot* = *Bot*
  by (*simp add*: *list.case-eq-if*)

lemma *b-e-check-weaken-top*:
  assumes *check* $\mathcal{C}$ *es* (*Type tn*) = (*TopType tm*)
  shows *check* $\mathcal{C}$ *es* (*Type* (*ts*@*tn*)) = (*TopType tm*)
  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* $\mathcal{C}$ *e* (*Type tn*)))
    case (*TopType x1*)
    hence *check-single* $\mathcal{C}$ *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* $\mathcal{C}$ *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 ⊢ 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 𝒞 t)
  have ct-list-eq [TAny, TSome T-i32] [TSome t, TSome T-i32]
    by (simp add: to-ct-list-def ct-list-eq-def)
```

thus *?case*
using *ct-suffix-extend-ct-list-eq*[*OF ct-suffix-nil*[*of* [*TSome t*]]] *to-ct-list-def*
by *auto*
next
case (*br-table* $\mathcal{C}$ *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* $\mathcal{C}$ *t*)
thus *?case*
using *to-ct-list-def ct-suffix-refl is-mut-def tg-t-def*
by *auto*
next
case (*composition* $\mathcal{C}$ *es t1s t2s e t3s*)
thus *?case*
using *b-e-type-checker-compose*
by *simp*
next
case (*weakening* $\mathcal{C}$ *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 ($\mathcal{C}$ ⊢ *es* : (*tn* -> *tm*)) = (*b-e-type-checker* $\mathcal{C}$ *es* (*tn* -> *tm*))
using *b-e-type-checker-sound b-e-type-checker-complete*
by *blast*

end

11 WebAssembly Interpreter

theory *Wasm-Interpreter* **imports** *Wasm* **begin**

datatype *res-crash* =
CError
| *CExhaustion*

datatype *res* =
RCrash res-crash
| *RTrap*
| *RValue v list*

datatype *res-step* =
RSCrash res-crash
| *RSBreak nat v list*

| *RSReturn v list*
| *RSNormal e list*

abbreviation *crash-error* **where** *crash-error* ≡ *RSCrash CError*

type-synonym *depth* = *nat*
type-synonym *fuel* = *nat*

type-synonym *config-tuple* = *s* × *v list* × *e list*

type-synonym *config-one-tuple* = *s* × *v list* × *v list* × *e*

type-synonym *res-tuple* = *s* × *v list* × *res-step*

fun *split-vals* :: *b-e list* ⇒ *v list* × *b-e list* **where**
  split-vals ((*C v*)#*es*) = (*let* (*vs′*, *es′*) = *split-vals es in* (*v*#*vs′*, *es′*))
| *split-vals es* = ([], *es*)

fun *split-vals-e* :: *e list* ⇒ *v list* × *e list* **where**
  split-vals-e (($ *C v*)#*es*) = (*let* (*vs′*, *es′*) = *split-vals-e es in* (*v*#*vs′*, *es′*))
| *split-vals-e es* = ([], *es*)

fun *split-n* :: *v list* ⇒ *nat* ⇒ *v list* × *v list* **where**
  split-n [] *n* = ([], [])
| *split-n es 0* = ([], *es*)
| *split-n* (*e*#*es*) (*Suc n*) = (*let* (*es′*, *es″*) = *split-n es n in* (*e*#*es′*, *es″*))

lemma *split-n-conv-take-drop*: *split-n es n* = (*take n es*, *drop n es*)
  by (*induction es n rule*: *split-n.induct*, *simp-all*)

lemma *split-n-length*:
  assumes *split-n es n* = (*es1*, *es2*) *length es* ≥ *n*
  shows *length es1* = *n*
  using *assms*
  unfolding *split-n-conv-take-drop*
  by *fastforce*

lemma *split-n-conv-app*:
  assumes *split-n es n* = (*es1*, *es2*)
  shows *es* = *es1*@*es2*
  using *assms*
  unfolding *split-n-conv-take-drop*
  by *auto*

lemma *app-conv-split-n*:
  assumes *es* = *es1*@*es2*
  shows *split-n es* (*length es1*) = (*es1*, *es2*)
  using *assms*
  unfolding *split-n-conv-take-drop*

by *auto*

lemma *split-vals-const-list*: *split-vals* (*map EConst vs*) = (*vs*, [])
by (*induction vs*, *simp-all*)

lemma *split-vals-e-const-list*: *split-vals-e* ($$* *vs*) = (*vs*, [])
by (*induction vs*, *simp-all*)

lemma *split-vals-e-conv-app*:
assumes *split-vals-e xs* = (*as*, *bs*)
shows *xs* = ($$* *as*)@*bs*
using *assms*
proof (*induction xs arbitrary*: *as rule*: *split-vals-e.induct*)
case (*1 v es*)
obtain *as′ bs′* **where** *split-vals-e es* = (*as′*, *bs′*)
by (*meson surj-pair*)
thus *?case*
using *1*
by *fastforce*
qed *simp-all*

abbreviation *expect* :: *′a option* ⇒ (*′a* ⇒ *′b*) ⇒ *′b* ⇒ *′b* **where**
expect a f b ≡ (*case a of*
Some a′ ⇒ *f a′*
| *None* ⇒ *b*)

abbreviation *vs-to-es* :: *v list* ⇒ *e list*
where *vs-to-es v* ≡ $$* (*rev v*)

definition *e-is-trap* :: *e* ⇒ *bool* **where**
e-is-trap e = (*case e of Trap* ⇒ *True* | - ⇒ *False*)

definition *es-is-trap* :: *e list* ⇒ *bool* **where**
es-is-trap es = (*case es of* [*e*] ⇒ *e-is-trap e* | - ⇒ *False*)

lemma[*simp*]: *e-is-trap e* = (*e* = *Trap*)
using *e-is-trap-def*
by (*cases e*) *auto*

lemma[*simp*]: *es-is-trap es* = (*es* = [*Trap*])
proof (*cases es*)
case *Nil*
thus *?thesis*
using *es-is-trap-def*
by *auto*
next
case *outer-Cons*:(*Cons a list*)
thus *?thesis*
proof (*cases list*)

case *Nil*
thus *?thesis*
using *outer-Cons es-is-trap-def*
by *auto*
next
case (*Cons a′ list′*)
thus *?thesis*
using *es-is-trap-def outer-Cons*
by *auto*
qed
qed

axiomatization
mem-grow-impl:: *mem* ⇒ *nat* ⇒ *mem option* **where**
mem-grow-impl-correct:(*mem-grow-impl m n* = *Some m′*) ⟹ (*mem-grow m n* = *m′*)

axiomatization
host-apply-impl:: *s* ⇒ *tf* ⇒ *host* ⇒ *v list* ⇒ (*s* × *v list*) *option* **where**
host-apply-impl-correct:(*host-apply-impl s tf h vs* = *Some m′*) ⟹ (∃ *hs. host-apply s tf h vs hs* = *Some m′*)

```
function (sequential)
    run-step :: depth ⇒ nat ⇒ config-tuple ⇒ res-tuple
and run-one-step :: depth ⇒ nat ⇒ config-one-tuple ⇒ res-tuple where
  run-step d i (s,vs,es) = (let (ves, es′) = split-vals-e es in
                            case es′ of
                              [] ⇒ (s,vs, crash-error)
                            | e#es′′ ⇒
                              if e-is-trap e
                                then
                                  if (es′′ ≠ [] ∨ ves ≠ [])
                                    then
                                      (s, vs, RSNormal [Trap])
                                    else
                                      (s, vs, crash-error)
                                else
                                  (let (s′,vs′,r) = run-one-step d i (s,vs,(rev ves),e) in
                                   case r of
                                     RSNormal res ⇒ (s′, vs′, RSNormal (res@es′′))
                                   | - ⇒ (s′, vs′, r)))
| run-one-step d i (s, vs, ves, e) =
     (case e of
    — B-E
      — UNOPS
        $(Unop-i T-i32 iop) ⇒
         (case ves of
            (ConstInt32 c)#ves′ ⇒
```

```
        (s, vs, RSNormal (vs-to-es ((ConstInt32 (app-unop-i iop c))#ves')))
      | - ⇒ (s, vs, crash-error))
    | $(Unop-i T-i64 iop) ⇒
      (case ves of
        (ConstInt64 c)#ves' ⇒
         (s, vs, RSNormal (vs-to-es ((ConstInt64 (app-unop-i iop c))#ves')))
       | - ⇒ (s, vs, crash-error))
    | $(Unop-i - iop) ⇒ (s, vs, crash-error)
    | $(Unop-f T-f32 fop) ⇒
      (case ves of
        (ConstFloat32 c)#ves' ⇒
         (s, vs, RSNormal (vs-to-es ((ConstFloat32 (app-unop-f fop c))#ves')))
       | - ⇒ (s, vs, crash-error))
    | $(Unop-f T-f64 fop) ⇒
      (case ves of
        (ConstFloat64 c)#ves' ⇒
         (s, vs, RSNormal (vs-to-es ((ConstFloat64 (app-unop-f fop c))#ves')))
       | - ⇒ (s, vs, crash-error))
    | $(Unop-f - fop) ⇒ (s, vs, crash-error)
    — BINOPS
    | $(Binop-i T-i32 iop) ⇒
      (case ves of
        (ConstInt32 c2)#(ConstInt32 c1)#ves' ⇒
              expect (app-binop-i iop c1 c2) (λc. (s, vs, RSNormal (vs-to-es
((ConstInt32 c)#ves')))) (s, vs, RSNormal ((vs-to-es ves')@[Trap]))
       | - ⇒ (s, vs, crash-error))
    | $(Binop-i T-i64 iop) ⇒
      (case ves of
        (ConstInt64 c2)#(ConstInt64 c1)#ves' ⇒
              expect (app-binop-i iop c1 c2) (λc. (s, vs, RSNormal (vs-to-es
((ConstInt64 c)#ves')))) (s, vs, RSNormal ((vs-to-es ves')@[Trap]))
       | - ⇒ (s, vs, crash-error))
    | $(Binop-i - iop) ⇒ (s, vs, crash-error)
    | $(Binop-f T-f32 fop) ⇒
      (case ves of
        (ConstFloat32 c2)#(ConstFloat32 c1)#ves' ⇒
              expect (app-binop-f fop c1 c2) (λc. (s, vs, RSNormal (vs-to-es
((ConstFloat32 c)#ves')))) (s, vs, RSNormal ((vs-to-es ves')@[Trap]))
       | - ⇒ (s, vs, crash-error))
    | $(Binop-f T-f64 fop) ⇒
     (case ves of
       (ConstFloat64 c2)#(ConstFloat64 c1)#ves' ⇒
             expect (app-binop-f fop c1 c2) (λc. (s, vs, RSNormal (vs-to-es
((ConstFloat64 c)#ves')))) (s, vs, RSNormal ((vs-to-es ves')@[Trap]))
      | - ⇒ (s, vs, crash-error))
    | $(Binop-f - fop) ⇒ (s, vs, crash-error)
    — TESTOPS
    | $(Testop T-i32 testop) ⇒
      (case ves of
```

$(ConstInt32\ c)\#ves' \Rightarrow$
$(s,\ vs,\ RSNormal\ (vs\text{-}to\text{-}es\ ((ConstInt32\ (wasm\text{-}bool\ (app\text{-}testop\text{-}i\ testop\ c)))\#ves')))$
$|\ \text{-} \Rightarrow (s,\ vs,\ crash\text{-}error))$
$|\ \$(Testop\ T\text{-}i64\ testop) \Rightarrow$
$(case\ ves\ of$
$(ConstInt64\ c)\#ves' \Rightarrow$
$(s,\ vs,\ RSNormal\ (vs\text{-}to\text{-}es\ ((ConstInt32\ (wasm\text{-}bool\ (app\text{-}testop\text{-}i\ testop\ c)))\#ves')))$
$|\ \text{-} \Rightarrow (s,\ vs,\ crash\text{-}error))$
$|\ \$(Testop\ \text{-}\ testop) \Rightarrow (s,\ vs,\ crash\text{-}error)$
— *RELOPS*
$|\ \$(Relop\text{-}i\ T\text{-}i32\ iop) \Rightarrow$
$(case\ ves\ of$
$(ConstInt32\ c2)\#(ConstInt32\ c1)\#ves' \Rightarrow$
$(s,\ vs,\ RSNormal\ (vs\text{-}to\text{-}es\ ((ConstInt32\ (wasm\text{-}bool\ (app\text{-}relop\text{-}i\ iop\ c1\ c2)))\#ves')))$
$|\ \text{-} \Rightarrow (s,\ vs,\ crash\text{-}error))$
$|\ \$(Relop\text{-}i\ T\text{-}i64\ iop) \Rightarrow$
$(case\ ves\ of$
$(ConstInt64\ c2)\#(ConstInt64\ c1)\#ves' \Rightarrow$
$(s,\ vs,\ RSNormal\ (vs\text{-}to\text{-}es\ ((ConstInt32\ (wasm\text{-}bool\ (app\text{-}relop\text{-}i\ iop\ c1\ c2)))\#ves')))$
$|\ \text{-} \Rightarrow (s,\ vs,\ crash\text{-}error))$
$|\ \$(Relop\text{-}i\ \text{-}\ iop) \Rightarrow (s,\ vs,\ crash\text{-}error)$
$|\ \$(Relop\text{-}f\ T\text{-}f32\ fop) \Rightarrow$
$(case\ ves\ of$
$(ConstFloat32\ c2)\#(ConstFloat32\ c1)\#ves' \Rightarrow$
$(s,\ vs,\ RSNormal\ (vs\text{-}to\text{-}es\ ((ConstInt32\ (wasm\text{-}bool\ (app\text{-}relop\text{-}f\ fop\ c1\ c2)))\#ves')))$
$|\ \text{-} \Rightarrow (s,\ vs,\ crash\text{-}error))$
$|\ \$(Relop\text{-}f\ T\text{-}f64\ fop) \Rightarrow$
$(case\ ves\ of$
$(ConstFloat64\ c2)\#(ConstFloat64\ c1)\#ves' \Rightarrow$
$(s,\ vs,\ RSNormal\ (vs\text{-}to\text{-}es\ ((ConstInt32\ (wasm\text{-}bool\ (app\text{-}relop\text{-}f\ fop\ c1\ c2)))\#ves')))$
$|\ \text{-} \Rightarrow (s,\ vs,\ crash\text{-}error))$
$|\ \$(Relop\text{-}f\ \text{-}\ fop) \Rightarrow (s,\ vs,\ crash\text{-}error)$
— *CONVERT*
$|\ \$(Cvtop\ t2\ Convert\ t1\ sx) \Rightarrow$
$(case\ ves\ of$
$v\#ves' \Rightarrow$
$(if\ (types\text{-}agree\ t1\ v)$
then
$expect\ (cvt\ t2\ sx\ v)\ (\lambda v'.\ (s,\ vs,\ RSNormal\ (vs\text{-}to\text{-}es\ (v'\#ves'))))\ (s,\ vs,\ RSNormal\ ((vs\text{-}to\text{-}es\ ves')@[Trap]))$
else
$(s,\ vs,\ crash\text{-}error))$
$|\ \text{-} \Rightarrow (s,\ vs,\ crash\text{-}error))$

```
  | $(Cvtop t2 Reinterpret t1 sx) ⇒
     (case ves of
        v#ves′ ⇒
          (if (types-agree t1 v ∧ sx = None)
             then
               (s, vs, RSNormal (vs-to-es ((wasm-deserialise (bits v) t2)#ves′)))
             else
               (s, vs, crash-error))
       | - ⇒ (s, vs, crash-error))
  — UNREACHABLE
  | $Unreachable ⇒
     (s, vs, RSNormal ((vs-to-es ves)@[Trap]))
  — NOP
  | $Nop ⇒
     (s, vs, RSNormal (vs-to-es ves))
  — DROP
  | $Drop ⇒
     (case ves of
        v#ves′ ⇒
          (s, vs, RSNormal (vs-to-es ves′))
       | - ⇒ (s, vs, crash-error))
  — SELECT
  | $Select ⇒
     (case ves of
        (ConstInt32 c)#v2#v1#ves′ ⇒
          (if int-eq c 0 then (s, vs, RSNormal (vs-to-es (v2#ves′))) else (s, vs,
RSNormal (vs-to-es (v1#ves′))))
       | - ⇒ (s, vs, crash-error))
  — BLOCK
  | $(Block (t1s -> t2s) es) ⇒
     (if length ves ≥ length t1s
        then
          let (ves′, ves′′) = split-n ves (length t1s) in
            (s, vs, RSNormal ((vs-to-es ves′′) @ [Label (length t2s) [] ((vs-to-es
ves′)@($* es))]))
        else
          (s, vs, crash-error))
  — LOOP
  | $(Loop (t1s -> t2s) es) ⇒
     (if length ves ≥ length t1s
        then
          let (ves′, ves′′) = split-n ves (length t1s) in
         (s, vs, RSNormal ((vs-to-es ves′′) @ [Label (length t1s) [$(Loop (t1s ->
t2s) es)] ((vs-to-es ves′)@($* es))]))
        else
          (s, vs, crash-error))
  — IF
  | $(If tf es1 es2) ⇒
     (case ves of
```

```
      (ConstInt32 c)#ves' ⇒
        if int-eq c 0
          then
            (s, vs, RSNormal ((vs-to-es ves')@[$(Block tf es2)]))
          else
            (s, vs, RSNormal ((vs-to-es ves')@[$(Block tf es1)]))
    | - ⇒ (s, vs, crash-error))
— BR
| $Br j ⇒
    (s, vs, RSBreak j ves)
— BR-IF
| $Br-if j ⇒
    (case ves of
      (ConstInt32 c)#ves' ⇒
        if int-eq c 0
          then
            (s, vs, RSNormal (vs-to-es ves'))
          else
            (s, vs, RSNormal ((vs-to-es ves') @ [$Br j]))
    | - ⇒ (s, vs, crash-error))
— BR-TABLE
| $Br-table js j ⇒
    (case ves of
      (ConstInt32 c)#ves' ⇒
      let k = nat-of-int c in
        if k < length js
          then
            (s, vs, RSNormal ((vs-to-es ves') @ [$Br (js!k)]))
          else
            (s, vs, RSNormal ((vs-to-es ves') @ [$Br j]))
    | - ⇒ (s, vs, crash-error))
— CALL
| $Call j ⇒
    (s, vs, RSNormal ((vs-to-es ves) @ [Callcl (sfunc s i j)]))
— CALL-INDIRECT
| $Call-indirect j ⇒
    (case ves of
      (ConstInt32 c)#ves' ⇒
        (case (stab s i (nat-of-int c)) of
          Some cl ⇒
            if (stypes s i j = cl-type cl)
              then
                (s, vs, RSNormal ((vs-to-es ves') @ [Callcl cl]))
              else
                (s, vs, RSNormal ((vs-to-es ves')@[Trap]))
        | - ⇒ (s, vs, RSNormal ((vs-to-es ves')@[Trap])))
    | - ⇒ (s, vs, crash-error))
— RETURN
| $Return ⇒
```

```
      (s, vs, RSReturn ves)
    — GET-LOCAL
    | $Get-local j ⇒
      (if j < length vs
        then (s, vs, RSNormal (vs-to-es ((vs!j)#ves)))
        else (s, vs, crash-error))
    — SET-LOCAL
    | $Set-local j ⇒
      (case ves of
        v#ves′ ⇒
          if j < length vs
            then (s, vs[j := v], RSNormal (vs-to-es ves′))
            else (s, vs, crash-error)
       | - ⇒ (s, vs, crash-error))
    — TEE-LOCAL
    | $Tee-local j ⇒
      (case ves of
        v#ves′ ⇒
          (s, vs, RSNormal ((vs-to-es (v#ves)) @ [$(Set-local j)]))
       | - ⇒ (s, vs, crash-error))
    — GET-GLOBAL
    | $Get-global j ⇒
      (s, vs, RSNormal (vs-to-es ((sglob-val s i j)#ves)))
    — SET-GLOBAL
    | $Set-global j ⇒
      (case ves of
        v#ves′ ⇒ ((supdate-glob s i j v), vs, RSNormal (vs-to-es ves′))
       | - ⇒ (s, vs, crash-error))
    — LOAD
    | $(Load t None a off) ⇒
      (case ves of
        (ConstInt32 k)#ves′ ⇒
          expect (smem-ind s i)
            (λj.
              expect (load ((mem s)!j) (nat-of-int k) off (t-length t))
              (λbs. (s, vs, RSNormal (vs-to-es ((wasm-deserialise bs t)#ves′))))
                (s, vs, RSNormal ((vs-to-es ves′)@[Trap])))
            (s, vs, crash-error)
       | - ⇒ (s, vs, crash-error))
    — LOAD PACKED
    | $(Load t (Some (tp, sx)) a off) ⇒
      (case ves of
        (ConstInt32 k)#ves′ ⇒
          expect (smem-ind s i)
            (λj.
              expect (load-packed sx ((mem s)!j) (nat-of-int k) off (tp-length tp)
(t-length t))
              (λbs. (s, vs, RSNormal (vs-to-es ((wasm-deserialise bs t)#ves′))))
                (s, vs, RSNormal ((vs-to-es ves′)@[Trap])))
```

```
            (s, vs, crash-error)
         | - ⇒ (s, vs, crash-error))
   — STORE
   | $(Store t None a off) ⇒
      (case ves of
         v#(ConstInt32 k)#ves' ⇒
          (if (types-agree t v)
           then
             expect (smem-ind s i)
               (λj.
                 expect (store ((mem s)!j) (nat-of-int k) off (bits v) (t-length t))
                     (λmem'. (s(|mem:= ((mem s)[j := mem'])|), vs, RSNormal
(vs-to-es ves')))
                   (s, vs, RSNormal ((vs-to-es ves')@[Trap])))
               (s, vs, crash-error)
           else
             (s, vs, crash-error))
         | - ⇒ (s, vs, crash-error))
   — STORE-PACKED
   | $(Store t (Some tp) a off) ⇒
      (case ves of
            v#(ConstInt32 k)#ves' ⇒
             (if (types-agree t v)
              then
                expect (smem-ind s i)
                  (λj.
                    expect (store-packed ((mem s)!j) (nat-of-int k) off (bits v)
(tp-length tp))
                      (λmem'. (s(|mem:= ((mem s)[j := mem'])|), vs, RSNormal
(vs-to-es ves')))
                       (s, vs, RSNormal ((vs-to-es ves')@[Trap])))
                  (s, vs, crash-error)
              else
                (s, vs, crash-error))
            | - ⇒ (s, vs, crash-error))
   — CURRENT-MEMORY
   | $Current-memory ⇒
      expect (smem-ind s i)
          (λj. (s, vs, RSNormal (vs-to-es ((ConstInt32 (int-of-nat (mem-size
((s.mem s)!j))))#ves))))
        (s, vs, crash-error)
   — GROW-MEMORY
   | $Grow-memory ⇒
      (case ves of
         (ConstInt32 c)#ves' ⇒
           expect (smem-ind s i)
            (λj.
              let l = (mem-size ((s.mem s)!j)) in
              (expect (mem-grow-impl ((mem s)!j) (nat-of-int c))
```

$(\lambda mem'.\ (s(\!|mem{:}=\ ((mem\ s)[j := mem'])|\!),\ vs,\ RSNormal$
$(vs\text{-}to\text{-}es\ ((ConstInt32\ \ (int\text{-}of\text{-}nat\ l))\#ves'))))$
$(s,\ vs,\ RSNormal\ (vs\text{-}to\text{-}es\ ((ConstInt32\ int32\text{-}minus\text{-}one)\#ves')))))$
$(s,\ vs,\ crash\text{-}error)$
$|\ \text{-} \Rightarrow (s,\ vs,\ crash\text{-}error))$
— VAL - should not be executed
$|\ \$C\ v \Rightarrow (s,\ vs,\ crash\text{-}error)$
— E
— $CALLCL$
$|\ Callcl\ cl \Rightarrow$
$(case\ cl\ of$
$Func\text{-}native\ i'\ (t1s\ \text{->}\ t2s)\ ts\ es \Rightarrow$
$let\ n = length\ t1s\ in$
$let\ m = length\ t2s\ in$
$if\ length\ ves \geq n$
$then$
$let\ (ves',\ ves'') = split\text{-}n\ ves\ n\ in$
$let\ zs = n\text{-}zeros\ ts\ in$
$(s,\ vs,\ RSNormal\ ((vs\text{-}to\text{-}es\ ves'')\ @\ ([Local\ m\ i'\ ((rev\ ves')@zs)$
$[\$(Block\ ([]\ \text{->}\ t2s)\ es)]])))$
$else$
$(s,\ vs,\ crash\text{-}error)$
$|\ Func\text{-}host\ (t1s\ \text{->}\ t2s)\ f \Rightarrow$
$let\ n = length\ t1s\ in$
$let\ m = length\ t2s\ in$
$if\ length\ ves \geq n$
$then$
$let\ (ves',\ ves'') = split\text{-}n\ ves\ n\ in$
$case\ host\text{-}apply\text{-}impl\ s\ (t1s\ \text{->}\ t2s)\ f\ (rev\ ves')\ of$
$Some\ (s',rves) \Rightarrow$
$if\ list\text{-}all2\ types\text{-}agree\ t2s\ rves$
$then$
$(s',\ vs,\ RSNormal\ ((vs\text{-}to\text{-}es\ ves'')\ @\ (\$\$*\ rves)))$
$else$
$(s',\ vs,\ crash\text{-}error)$
$|\ None \Rightarrow (s,\ vs,\ RSNormal\ ((vs\text{-}to\text{-}es\ ves'')@[Trap]))$
$else$
$(s,\ vs,\ crash\text{-}error))$
— $LABEL$
$|\ Label\ ln\ les\ es \Rightarrow$
$if\ es\text{-}is\text{-}trap\ es$
$then$
$(s,\ vs,\ RSNormal\ ((vs\text{-}to\text{-}es\ ves)@[Trap]))$
$else$
$(if\ (const\text{-}list\ es)$
$then$
$(s,\ vs,\ RSNormal\ ((vs\text{-}to\text{-}es\ ves)@es))$
$else$
$let\ (s',\ vs',\ res) = run\text{-}step\ d\ i\ (s,\ vs,\ es)\ in$

```
                (case res of
                   RSBreak 0 bvs ⇒
                     if (length bvs ≥ ln)
                      then (s', vs', RSNormal ((vs-to-es ((take ln bvs)@ves))@les))
                       else (s', vs', crash-error)
                 | RSBreak (Suc n) bvs ⇒
                     (s', vs', RSBreak n bvs)
                 | RSReturn rvs ⇒
                     (s', vs', RSReturn rvs)
                 | RSNormal es' ⇒
                     (s', vs', RSNormal ((vs-to-es ves)@[Label ln les es']))
                 | - ⇒ (s', vs', crash-error)))
  — LOCAL
  | Local ln j vls es ⇒
       if es-is-trap es
         then
          (s, vs, RSNormal ((vs-to-es ves)@[Trap]))
         else
           (if (const-list es)
              then
                if (length es = ln)
                  then (s, vs, RSNormal ((vs-to-es ves)@es))
                  else (s, vs, crash-error)
              else
                case d of
                  0 ⇒ (s, vs, crash-error)
                | Suc d' ⇒
                    let (s', vls', res) = run-step d' j (s, vls, es) in
                    (case res of
                        RSReturn rvs ⇒
                          if (length rvs ≥ ln)
                            then (s', vs, RSNormal (vs-to-es ((take ln rvs)@ves)))
                            else (s', vs, crash-error)
                      | RSNormal es' ⇒
                          (s', vs, RSNormal ((vs-to-es ves)@[Local ln j vls' es']))
                      | - ⇒ (s', vs, RSCrash CExhaustion)))
  — TRAP - should not be executed
  | Trap ⇒ (s, vs, crash-error))
 by pat-completeness auto
termination
proof −
 {
   fix xs::e list and as b bs
   assume local-assms:(as, b#bs) = split-vals-e xs
   have 2∗(size b) < 2∗(size-list size xs) + 1
     using local-assms[symmetric] split-vals-e-conv-app
           size-list-estimation'[of b xs size b size]
     unfolding size-list-def
     by fastforce
```

```
  }
  thus ?thesis
    by (relation measure (case-sum
                              (λp. 2 ∗ (size-list size (snd (snd (snd (snd p))))) + 1)
                              (λp. 2 ∗ size (snd (snd (snd (snd (snd p)))))))) auto
qed

fun run-v :: fuel ⇒ depth ⇒ nat ⇒ config-tuple ⇒ (s × res) where
  run-v (Suc n) d i (s,vs,es) = (if (es-is-trap es)
                                   then (s, RTrap)
                                   else if (const-list es)
                                          then (s, RValue (fst (split-vals-e es)))
                                          else (let (s′,vs′,res) = (run-step d i (s,vs,es)) in
                                                  case res of
                                                    RSNormal es′ ⇒ run-v n d i (s′,vs′,es′)
                                                  | RSCrash error ⇒ (s, RCrash error)
                                                  | - ⇒ (s, RCrash CError)))
| run-v 0 d i (s,vs,es) = (s, RCrash CExhaustion)

end
```

12 Soundness of Interpreter

```
theory Wasm-Interpreter-Properties imports Wasm-Interpreter Wasm-Properties
begin

lemma is-const-list-vs-to-es-list: const-list ($$∗ vs)
  using is-const-list
  by auto

lemma not-const-vs-to-es-list:
  assumes ~(is-const e)
  shows vs1 @ [e] @ vs2 ≠ $$∗ vs
proof −
  fix vs
  {
    assume vs1 @ [e] @ vs2 = $$∗ vs
    hence (∀ y∈set (vs1 @ [e] @ vs2). ∃ x. y = $C x)
      by simp
    hence False
      using assms
      unfolding is-const-def
      by fastforce
  }
  thus vs1 @ [e] @ vs2 ≠ $$∗ vs
    by fastforce
qed

lemma neq-label-nested:[Label n les es] ≠ es
```

proof −
  have *size-list size* [*Label n les es*] > *size-list size es*
    by *simp*
  thus *?thesis*
    by *fastforce*
qed

lemma *neq-local-nested*:[*Local n i lvs es*] ≠ *es*
proof −
  have *size-list size* [*Local n i lvs es*] > *size-list size es*
    by *simp*
  thus *?thesis*
    by *fastforce*
qed

lemma *trap-not-value*:[*Trap*] ≠ $$*es*
  by *fastforce*

thm *Lfilled.simps*[*of* - - - [*e*], *simplified*]

lemma *lfilled-single*:
  assumes *Lfilled k lholed es* [*e*]
      ⋀ *a b c. e* ≠ *Label a b c*
  shows (*es* = [*e*] ∧ *lholed* = *LBase* [] []) ∨ *es* = []
  using *assms*
proof (*cases rule*: *Lfilled.cases*)
  case (*L0 vs es′*)
  thus *?thesis*
    by (*metis Nil-is-append-conv append-self-conv2 butlast-append butlast-snoc*)
next
  case (*LN vs n es′ l es″ k lfilledk*)
  assume (⋀*a b c. e* ≠ *Label a b c*)
  thus *?thesis*
    using *LN*(*2*)
    unfolding *Cons-eq-append-conv*
    by *fastforce*
qed

lemma *lfilled-eq*:
  assumes *Lfilled j lholed es LI*
      Lfilled j lholed es′ LI
  shows *es* = *es′*
  using *assms*
proof (*induction arbitrary*: *es′ rule*: *Lfilled.induct*)
  case (*L0 vs lholed es′ es*)
  thus *?case*
    using *Lfilled.simps*[*of 0*, *simplified*]
    by *auto*
next

case (*LN vs lholed n les′ l les″ k les lfilledk*)
thus *?case*
using *Lfilled.simps*[*of* (*k+1*) *LRec vs n les′ l les″ es′* (*vs* @ [*Label n les′ lfilledk*] @ *les″*), *simplified*]
by *auto*
qed

lemma *lfilled-size*:
assumes *Lfilled j lholed es LI*
shows *size-list size LI* $\geq$ *size-list size es*
using *assms*
by (*induction rule*: *Lfilled.induct*) *auto*

thm *Lfilled.simps*[*of - - es es*, *simplified*]

lemma *reduce-simple-not-eq*:
assumes ⦇*es*⦈ ⇝ ⦇*es′*⦈
shows *es* $\neq$ *es′*
using *assms*
proof (*induction es′ rule*: *reduce-simple.induct*)
case (*label-const vs n es*)
thus *?case*
using *neq-label-nested*
by *auto*
next
case (*br vs n i lholed LI es*)
have *size-list size* [*Label n es LI*] > *size-list size* (*vs* @ *es*)
using *lfilled-size*[*OF br*(*3*)]
by *simp*
thus *?case*
by *fastforce*
next
case (*local-const es n i vs*)
thus *?case*
using *neq-local-nested*
by *auto*
next
case (*return vs n j lholed es i vls*)
hence *size-list size* [*Local n i vls es*] > *size-list size vs*
using *lfilled-size*[*OF return*(*3*)]
by *simp*
thus *?case*
by *auto*
qed *auto*

lemma *reduce-not-eq*:
assumes ⦇*s*;*vs*;*es*⦈ ⇝-*i* ⦇*s′*;*vs′*;*es′*⦈
shows *es* $\neq$ *es′*
using *assms*

proof (*induction es′ rule*: *reduce.induct*)
  case (*basic e e′ s vs i*)
  thus *?case*
    using *reduce-simple-not-eq*
    by *simp*
next
  case (*callcl-host-Some cl t1s t2s f ves vcs n m s hs s′ vcs′ vs i*)
  thus *?case*
    by (*cases vcs′ rule:rev-cases*) *auto*
next
  case (*label s vs es i s′ vs′ es′ k lholed les les′*)
  thus *?case*
    using *lfilled-eq*
    by *fastforce*
qed *auto*

lemma *reduce-simple-not-value*:
  assumes ⦇*es*⦈ ⇝ ⦇*es′*⦈
  shows *es* ≠ $$∗ *vs*
  using *assms*
proof (*induction rule*: *reduce-simple.induct*)
  case (*block vs n t1s t2s m es*)
  have ¬(*is-const* ($*Block* (*t1s* -> *t2s*) *es*))
    unfolding *is-const-def*
    by *simp*
  thus *?case*
    using *not-const-vs-to-es-list*
    by (*metis append.right-neutral*)
next
  case (*loop vs n t1s t2s m es*)
  have ¬(*is-const* ($*Loop* (*t1s* -> *t2s*) *es*))
    unfolding *is-const-def*
    by *simp*
  thus *?case*
    using *not-const-vs-to-es-list*
    by (*metis append.right-neutral*)
next
  case (*trap lholed es*)
  show *?case*
    using *trap*(*2*)
  proof (*cases rule*: *Lfilled.cases*)
    case *L0*
    have ¬(*is-const Trap*)
      unfolding *is-const-def*
      by *simp*
    thus *?thesis*
      using *L0 not-const-vs-to-es-list*
      by *fastforce*
  qed *auto*

qed *auto*

lemma *reduce-not-value*:
  assumes $\llparenthesis s;vs;es \rrparenthesis \leadsto\text{-}i \llparenthesis s';vs';es' \rrparenthesis$
  shows $es \neq \$\$* \ ves$
  using *assms*
proof (*induction es′ arbitrary*: *ves rule*: *reduce.induct*)
  case (*basic e e′ s vs i*)
  thus *?case*
    using *reduce-simple-not-value*
    by *fastforce*
next
  case (*callcl-native cl i′ j ts es s t1s t2s ves vcs n k m zs vs i*)
  have ¬(*is-const* (*Callcl cl*))
    unfolding *is-const-def*
    by *simp*
  thus *?case*
    using *not-const-vs-to-es-list*
    by (*metis append.right-neutral*)
next
  case (*callcl-host-Some cl t1s t2s f ves vcs n m s i s′ vcs′ vs*)
  have ¬(*is-const* (*Callcl cl*))
    unfolding *is-const-def*
    by *simp*
  thus *?case*
    using *not-const-vs-to-es-list*
    by (*metis append.right-neutral*)
next
  case (*callcl-host-None cl t1s t2s f ves vcs n m s vs i*)
  have ¬(*is-const* (*Callcl cl*))
    unfolding *is-const-def*
    by *simp*
  thus *?case*
    using *not-const-vs-to-es-list*
    by (*metis append.right-neutral*)
next
  case (*label s vs es i s′ vs′ es′ k lholed les les′*)
  show *?case*
    using *label*(*2*,*4*)
  proof (*induction rule*: *Lfilled.induct*)
    case (*L0 lvs lholed les′ les*)
    {
      assume $lvs \ @ \ les \ @ \ les' = \$\$* \ ves$
      hence $(\forall y \in set \ (lvs \ @ \ les \ @ \ les'). \ \exists x. \ y = \$C \ x)$
        by *simp*
      hence $(\forall y \in set \ les. \ \exists x. \ y = \$C \ x)$
        by *simp*
      hence $\exists vs1. \ les = \$\$* \ vs1$
        unfolding *ex-map-conv*.

}
thus *?case*
using *L0*(*3*)
by *fastforce*
next
case (*LN lvs lholed ln les′ l les′′ k les lfilledk*)
have ¬(*is-const* (*Label ln les′ lfilledk*))
unfolding *is-const-def*
by *simp*
thus *?case*
using *not-const-vs-to-es-list*
by *fastforce*
qed
qed *auto*

lemma *reduce-simple-not-nil*:
assumes (|*es*|) ⇝ (|*es′*|)
shows *es* ≠ []
using *assms*
proof (*induction rule*: *reduce-simple.induct*)
case (*trap es lholed*)
thus *?case*
using *Lfilled.simps*[*of 0 lholed* [*Trap*]]
by *auto*
qed *auto*

lemma *reduce-not-nil*:
assumes (|*s*;*vs*;*es*|) ⇝-*i* (|*s′*;*vs′*;*es′*|)
shows *es* ≠ []
using *assms*
proof (*induction rule*: *reduce.induct*)
case (*basic e e′ s vs i*)
thus *?case*
using *reduce-simple-not-nil*
by *simp*
next
case (*label s vs es i s′ vs′ es′ k lholed les les′*)
show *?case*
using *lfilled-size*[*OF label*(*2*)] *label*(*4*)
by (*metis One-nat-def add-is-0 le-0-eq list.exhaust list.size*(*2*) *list.size-gen*(*1*) *zero-neq-one*)
qed *auto*

lemma *reduce-simple-not-trap*:
assumes (|*es*|) ⇝ (|*es′*|)
shows *es* ≠ [*Trap*]
using *assms*
by (*induction rule*: *reduce-simple.induct*) *auto*

lemma *reduce-not-trap*:
  assumes $(\!|s;vs;es|\!) \rightsquigarrow\text{-}i\ (\!|s';vs';es'|\!)$
  shows $es \neq [Trap]$
  using *assms*
proof (*induction rule*: *reduce.induct*)
  case (*basic e e′ s vs i*)
  thus *?case*
    using *reduce-simple-not-trap*
    by *simp*
next
  case (*label s vs es i s′ vs′ es′ k lholed les les′*)
  {
    assume $les = [Trap]$
    hence *Lfilled k lholed es* [*Trap*]
      using *label*(*2*)
      by *simp*
    hence *False*
      using *lfilled-single reduce-not-nil*[*OF label*(*1*)] *label*(*4*)
      by *fastforce*
  }
  thus *?case*
    by *auto*
qed *auto*

lemma *reduce-simple-call*: $\neg(\!|[\$Call\ j]|\!) \rightsquigarrow (\!|es'|\!)$
  using *reduce-simple.simps*[*of* [$*Call j*], *simplified*] *lfilled-single*
  by *fastforce*

lemma *reduce-call*:
  assumes $(\!|s;vs;[\$Call\ j]|\!) \rightsquigarrow\text{-}\ i\ (\!|s';vs';es'|\!)$
  shows $s = s'$
    $vs = vs'$
    $es' = [Callcl\ (sfunc\ s\ i\ j)]$
  using *assms*
proof (*induction* [$*Call j*]:: *e list i s′ vs′ es′ rule*: *reduce.induct*)
  case (*label s vs es i s′ vs′ es′ k lholed les′*)
  have $es = [\$Call\ j]$
    $lholed = LBase\ []\ []$
    using *reduce-not-nil*[*OF label*(*1*)] *lfilled-single*[*OF label*(*5*)]
    by *auto*
  thus $s = s'$
    $vs = vs'$
    $les' = [Callcl\ (sfunc\ s\ i\ j)]$
    using *label*(*2*,*3*,*4*,*6*) *Lfilled.simps*[*of k LBase* [] [] [*Callcl* (*sfunc s i j*)] *les′*]
    by *auto*
qed (*auto simp add*: *reduce-simple-call*)

lemma *run-one-step-basic-unreachable-result*:
  assumes *run-one-step d i* (*s*,*vs*,*ves*,$*Unreachable*) = (*s′*, *vs′*, *res*)

shows $\exists\, r.\ res = RSNormal\ r$
using *assms*
by *auto*

lemma *run-one-step-basic-nop-result*:
assumes $run\text{-}one\text{-}step\ d\ i\ (s,vs,ves,\$Nop) = (s',\ vs',\ res)$
shows $\exists\, r.\ res = RSNormal\ r$
using *assms*
by *auto*

lemma *run-one-step-basic-drop-result*:
assumes $run\text{-}one\text{-}step\ d\ i\ (s,vs,ves,\$Drop) = (s',\ vs',\ res)$
shows $(\exists\, r.\ res = RSNormal\ r) \vee (\exists\, e.\ res = RSCrash\ e)$
using *assms*
by (*cases ves*) *auto*

lemma *run-one-step-basic-select-result*:
assumes $run\text{-}one\text{-}step\ d\ i\ (s,vs,ves,\$Select) = (s',\ vs',\ res)$
shows $(\exists\, r.\ res = RSNormal\ r) \vee (\exists\, e.\ res = RSCrash\ e)$
using *assms*
proof (*cases ves*)
case (*Cons a list*)
thus *?thesis*
using *assms*
proof (*cases a*; *cases list*)
fix *x1a aa listaa*
assume $a = ConstInt32\ x1a$ **and** $list = aa\#listaa$
thus *?thesis*
using *assms Cons*
by (*cases listaa*; *cases int-eq x1a 0*) *auto*
qed *auto*
qed *auto*

lemma *run-one-step-basic-block-result*:
assumes $run\text{-}one\text{-}step\ d\ i\ (s,vs,ves,\$(Block\ x51\ x52)) = (s',\ vs',\ res)$
shows $(\exists\, r.\ res = RSNormal\ r) \vee (\exists\, e.\ res = RSCrash\ e)$
using *assms*
proof −
obtain *t1s t2s* **where** $x51 = (t1s \text{ -> } t2s)$
using *tf.exhaust*
by *blast*
moreover obtain $ves'\ ves''$ **where** $split\text{-}n\ ves\ (length\ t1s) = (ves',\ ves'')$
by (*metis surj-pair*)
ultimately
show *?thesis*
using *assms*
by (*cases* $length\ t1s \leq length\ ves$) *auto*
qed

lemma *run-one-step-basic-loop-result*:
  assumes *run-one-step d i* (*s*,*vs*,*ves*,$(*Loop x61 x62*)) = (*s'*, *vs'*, *res*)
  shows (∃ *r*. *res* = *RSNormal r*) ∨ (∃ *e*. *res* = *RSCrash e*)
  using *assms*
proof −
  obtain *t1s t2s* **where** *x61* = (*t1s* -> *t2s*)
    using *tf.exhaust*
    by *blast*
  moreover obtain *ves' ves''* **where** *split-n ves* (*length t1s*) = (*ves'*, *ves''*)
    by (*metis surj-pair*)
  ultimately
    show *?thesis*
    using *assms*
    by (*cases length t1s* ≤ *length ves*) *auto*
qed

lemma *run-one-step-basic-if-result*:
  assumes *run-one-step d i* (*s*,*vs*,*ves*,$(*If x71 x72 x73*)) = (*s'*, *vs'*, *res*)
  shows (∃ *r*. *res* = *RSNormal r*) ∨ (∃ *e*. *res* = *RSCrash e*)
  using *assms*
proof (*cases ves*)
  case (*Cons a list*)
  thus *?thesis*
    using *assms*
  proof (*cases a*)
    fix *x1a*
    assume *a* = *ConstInt32 x1a*
    thus *?thesis*
      using *assms Cons*
      by (*cases int-eq x1a 0*) *auto*
  qed *auto*
qed *auto*

lemma *run-one-step-basic-br-result*:
  assumes *run-one-step d i* (*s*,*vs*,*ves*,$*Br x8*) = (*s'*, *vs'*, *res*)
  shows ∃ *r vrs*. *res* = *RSBreak r vrs*
  using *assms*
  by (*cases ves*) *auto*

lemma *run-one-step-basic-br-if-result*:
  assumes *run-one-step d i* (*s*,*vs*,*ves*,$*Br-if x9*) = (*s'*, *vs'*, *res*)
  shows (∃ *r*. *res* = *RSNormal r*) ∨ (∃ *e*. *res* = *RSCrash e*)
  using *assms*
proof (*cases ves*)
  case (*Cons a list*)
  thus *?thesis*
    using *assms*
  proof (*cases a*)
    case (*ConstInt32 x1*)

thus *?thesis*
using *assms Cons*
by (*cases int-eq x1 0*) *auto*
qed *auto*
qed *auto*

lemma *run-one-step-basic-br-table-result*:
assumes *run-one-step d i* (*s*,*vs*,*ves*,$*Br-table js j*) = (*s'*, *vs'*, *res*)
shows (∃ *r*. *res* = *RSNormal r*) ∨ (∃ *e*. *res* = *RSCrash e*)
using *assms*
proof (*cases ves*)
case (*Cons a list*)
thus *?thesis*
using *assms*
proof (*cases a*)
case (*ConstInt32 x1*)
thus *?thesis*
using *assms Cons*
by (*cases nat-of-int x1* < *length js*) *auto*
qed *auto*
qed *auto*

lemma *run-one-step-basic-return-result*:
assumes *run-one-step d i* (*s*,*vs*,*ves*,$*Return*) = (*s'*, *vs'*, *res*)
shows ∃ *vrs*. *res* = *RSReturn vrs*
using *assms*
by (*cases ves*) *auto*

lemma *run-one-step-basic-call-result*:
assumes *run-one-step d i* (*s*,*vs*,*ves*,$*Call x12*) = (*s'*, *vs'*, *res*)
shows ∃ *r*. *res* = *RSNormal r*
using *assms*
by (*cases ves*) *auto*

lemma *run-one-step-basic-call-indirect-result*:
assumes *run-one-step d i* (*s*,*vs*,*ves*,$*Call-indirect x13*) = (*s'*, *vs'*, *res*)
shows (∃ *r*. *res* = *RSNormal r*) ∨ (∃ *e*. *res* = *RSCrash e*)
using *assms*
proof (*cases ves*)
case (*Cons a list*)
thus *?thesis*
using *assms*
proof (*cases a*)
case (*ConstInt32 x1*)
thus *?thesis*
using *Cons assms*
proof (*cases stab s i* (*nat-of-int x1*))
case (*Some cl*)
thus *?thesis*

using *Cons assms ConstInt32*
by (*cases cl*; *cases stypes s i x13* = *cl-type cl*) *auto*
qed *auto*
qed *auto*
qed *auto*

lemma *run-one-step-basic-get-local-result*:
assumes *run-one-step d i* (*s*,*vs*,*ves*,$*Get-local x14*) = (*s*′, *vs*′, *res*)
shows (∃ *r*. *res* = *RSNormal r*) ∨ (∃ *e*. *res* = *RSCrash e*)
using *assms*
by (*cases x14* < *length vs*) *auto*

lemma *run-one-step-basic-set-local-result*:
assumes *run-one-step d i* (*s*,*vs*,*ves*,$*Set-local x15*) = (*s*′, *vs*′, *res*)
shows (∃ *r*. *res* = *RSNormal r*) ∨ (∃ *e*. *res* = *RSCrash e*)
using *assms*
by (*cases ves*; *cases x15* < *length vs*) *auto*

lemma *run-one-step-basic-tee-local-result*:
assumes *run-one-step d i* (*s*,*vs*,*ves*,$*Tee-local x16*) = (*s*′, *vs*′, *res*)
shows (∃ *r*. *res* = *RSNormal r*) ∨ (∃ *e*. *res* = *RSCrash e*)
using *assms*
by (*cases ves*) *auto*

lemma *run-one-step-basic-get-global-result*:
assumes *run-one-step d i* (*s*,*vs*,*ves*,$*Get-global x17*) = (*s*′, *vs*′, *res*)
shows (∃ *r*. *res* = *RSNormal r*) ∨ (∃ *e*. *res* = *RSCrash e*)
using *assms*
by *auto*

lemma *run-one-step-basic-set-global-result*:
assumes *run-one-step d i* (*s*,*vs*,*ves*,$*Set-global x18*) = (*s*′, *vs*′, *res*)
shows (∃ *r*. *res* = *RSNormal r*) ∨ (∃ *e*. *res* = *RSCrash e*)
using *assms*
by (*cases ves*) *auto*

lemma *run-one-step-basic-load-result*:
assumes *run-one-step d i* (*s*,*vs*,*ves*,$*Load x191 x192 x193 x194*) = (*s*′, *vs*′, *res*)
shows (∃ *r*. *res* = *RSNormal r*) ∨ (∃ *e*. *res* = *RSCrash e*)
proof (*cases x192*)
case *None*
thus *?thesis*
using *assms*
proof (*cases ves*)
case (*Cons a list*)
thus *?thesis*
using *assms None*
proof (*cases smem-ind s i*; *cases a*)
fix *aa x1*

assume *smem-ind s i* = *Some aa* **and** *a* = *ConstInt32 x1*
thus *?thesis*
using *assms None Cons*
by (*cases load* (*s.mem s* ! *aa*) (*nat-of-int x1*) *x194* (*t-length x191*)) *auto*
qed *auto*
qed *auto*
next
case (*Some a*)
thus *?thesis*
using *assms*
proof (*cases ves*)
case (*Cons a′ list*)
thus *?thesis*
using *assms Some*
proof (*cases smem-ind s i*; *cases a*; *cases a′*)
fix *aa x y x1*
assume *smem-ind s i* = *Some aa* **and** *a* = (*x*, *y*) **and** *a′* = *ConstInt32 x1*
thus *?thesis*
using *assms Some Cons*
by (*cases load-packed y* (*s.mem s* ! *aa*) (*nat-of-int x1*) *x194* (*tp-length x*) (*t-length x191*)) *auto*
qed *auto*
qed *auto*
qed

lemma *run-one-step-basic-store-result*:
assumes *run-one-step d i* (*s*,*vs*,*ves*,$*Store x201 x202 x203 x204*) = (*s′*, *vs′*, *res*)
shows (∃ *r*. *res* = *RSNormal r*) ∨ (∃ *e*. *res* = *RSCrash e*)
proof (*cases x202*)
case *None*
thus *?thesis*
using *assms*
proof (*cases ves*)
case (*Cons a list*)
note *outer-Cons* = *Cons*
thus *?thesis*
using *assms None*
proof (*cases list*)
case (*Cons a′ list′*)
thus *?thesis*
using *assms None outer-Cons*
proof (*cases a′*; *cases types-agree x201 a*; *cases smem-ind s i*)
fix *k aa*
assume *a′* = *ConstInt32 k* **and** *types-agree x201 a* **and** *smem-ind s i* = *Some aa*
thus *?thesis*
using *assms None outer-Cons Cons*
by (*cases store* (*s.mem s* ! *aa*) (*nat-of-int k*) *x204* (*bits a*) (*t-length x201*)) *auto*

qed *auto*
qed *auto*
qed *auto*
next
case (*Some* a'')
thus *?thesis*
using *assms*
proof (*cases ves*)
case (*Cons a list*)
note *outer-Cons* = *Cons*
thus *?thesis*
using *assms Some*
proof (*cases list*)
case (*Cons* a' *list*$'$)
thus *?thesis*
using *assms Some outer-Cons*
proof (*cases* a'; *cases types-agree x201 a*; *cases smem-ind s i*)
fix *k aa*
assume a' = *ConstInt32 k* **and** *types-agree x201 a* **and** *smem-ind s i* = *Some aa*
thus *?thesis*
using *assms Some outer-Cons Cons*
by (*cases store-packed* (*s.mem s* ! *aa*) (*nat-of-int k*) *x204* (*bits a*) (*tp-length* a'')) *auto*
qed *auto*
qed *auto*
qed *auto*
qed

lemma *run-one-step-basic-current-memory-result*:
assumes *run-one-step d i* (*s*,*vs*,*ves*,$*Current-memory*) = ($s'$, vs', *res*)
shows ($\exists$ *r*. *res* = *RSNormal r*) $\vee$ ($\exists$ *e*. *res* = *RSCrash e*)
using *assms*
by (*cases smem-ind s i*) *auto*

lemma *run-one-step-basic-grow-memory-result*:
assumes *run-one-step d i* (*s*,*vs*,*ves*,$*Grow-memory*) = ($s'$, vs', *res*)
shows ($\exists$ *r*. *res* = *RSNormal r*) $\vee$ ($\exists$ *e*. *res* = *RSCrash e*)
using *assms*
proof (*cases ves*)
case (*Cons a list*)
thus *?thesis*
using *assms*
proof (*cases a*; *cases smem-ind s i*)
fix *c* a'
assume *a* = *ConstInt32 c* **and** *smem-ind s i* = *Some* a'
thus *?thesis*
using *assms Cons*
by (*cases mem-grow-impl* (*s.mem s* ! a') (*nat-of-int c*)) *auto*

qed *auto*
qed *auto*

lemma *run-one-step-basic-const-result*:
assumes *run-one-step d i* (*s*,*vs*,*ves*,$*EConst x23*) = (*s'*, *vs'*, *res*)
shows ∃ *e*. *res* = *RSCrash e*
using *assms*
by *auto*

lemma *run-one-step-basic-unop-i-result*:
assumes *run-one-step d i* (*s*,*vs*,*ves*,$*Unop-i x241 x242*) = (*s'*, *vs'*, *res*)
shows (∃ *r*. *res* = *RSNormal r*) ∨ (∃ *e*. *res* = *RSCrash e*)
using *assms*
proof (*cases ves*)
case (*Cons a list*)
thus *?thesis*
using *assms*
by (*cases x241*; *cases a*) *auto*
qed (*cases x241*; *auto*)

lemma *run-one-step-basic-unop-f-result*:
assumes *run-one-step d i* (*s*,*vs*,*ves*,$*Unop-f x251 x252*) = (*s'*, *vs'*, *res*)
shows (∃ *r*. *res* = *RSNormal r*) ∨ (∃ *e*. *res* = *RSCrash e*)
using *assms*
proof (*cases ves*)
case (*Cons a list*)
thus *?thesis*
using *assms*
by (*cases x251*; *cases a*) *auto*
qed (*cases x251*; *auto*)

lemma *run-one-step-basic-binop-i-result*:
assumes *run-one-step d i* (*s*,*vs*,*ves*,$*Binop-i x261 x262*) = (*s'*, *vs'*, *res*)
shows (∃ *r*. *res* = *RSNormal r*) ∨ (∃ *e*. *res* = *RSCrash e*)
using *assms*
proof (*cases ves*)
case (*Cons a list*)
note *outer-Cons* = *Cons*
thus *?thesis*
using *assms*
proof (*cases list*)
case (*Cons a' list'*)
thus *?thesis*
using *assms outer-Cons*
proof (*cases x261*; *cases a*; *cases a'*)
fix *x1 x2*
assume *x261* = *T-i32 a* = *ConstInt32 x1* **and** *a'* = *ConstInt32 x2*
thus *?thesis*
using *assms outer-Cons Cons*

by (*cases app-binop-i x262 x2 x1*) *auto*
next
fix *x1 x2*
assume *x261* = *T-i64 a* = *ConstInt64 x1* **and** *a′* = *ConstInt64 x2*
thus *?thesis*
using *assms outer-Cons Cons*
by (*cases app-binop-i x262 x2 x1*) *auto*
qed *auto*
qed (*cases x261*; *cases a*; *auto*)
qed (*cases x261*; *auto*)

lemma *run-one-step-basic-binop-f-result*:
assumes *run-one-step d i* (*s*,*vs*,*ves*,$*Binop-f x271 x272*) = (*s′*, *vs′*, *res*)
shows (∃ *r*. *res* = *RSNormal r*) ∨ (∃ *e*. *res* = *RSCrash e*)
using *assms*
proof (*cases ves*)
case (*Cons a list*)
note *outer-Cons* = *Cons*
thus *?thesis*
using *assms*
proof (*cases list*)
case (*Cons a′ list′*)
thus *?thesis*
using *assms outer-Cons*
proof (*cases x271*; *cases a*; *cases a′*)
fix *x1 x2*
assume *x271* = *T-f32 a* = *ConstFloat32 x1* **and** *a′* = *ConstFloat32 x2*
thus *?thesis*
using *assms outer-Cons Cons*
by (*cases app-binop-f x272 x2 x1*) *auto*
next
fix *x1 x2*
assume *x271* = *T-f64 a* = *ConstFloat64 x1* **and** *a′* = *ConstFloat64 x2*
thus *?thesis*
using *assms outer-Cons Cons*
by (*cases app-binop-f x272 x2 x1*) *auto*
qed *auto*
qed (*cases x271*; *cases a*; *auto*)
qed (*cases x271*; *auto*)

lemma *run-one-step-basic-testop-result*:
assumes *run-one-step d i* (*s*,*vs*,*ves*,$*Testop x281 x282*) = (*s′*, *vs′*, *res*)
shows (∃ *r*. *res* = *RSNormal r*) ∨ (∃ *e*. *res* = *RSCrash e*)
using *assms*
proof (*cases ves*)
case (*Cons a list*)
thus *?thesis*
using *assms*
by (*cases x281*; *cases a*) *auto*

qed (*cases x281*; *auto*)

lemma *run-one-step-basic-relop-i-result*:
  assumes *run-one-step d i* (*s*,*vs*,*ves*,$*Relop-i x291 x292*) = (*s*′, *vs*′, *res*)
  shows (∃ *r*. *res* = *RSNormal r*) ∨ (∃ *e*. *res* = *RSCrash e*)
  using *assms*
proof (*cases ves*)
  case (*Cons a list*)
  note *outer-Cons* = *Cons*
  thus *?thesis*
    using *assms*
  proof (*cases list*)
    case (*Cons a*′ *list*′)
    thus *?thesis*
      using *assms outer-Cons*
    proof (*cases x291*; *cases a*; *cases a*′)
      fix *x1 x2*
      assume *x291* = *T-i32 a* = *ConstInt32 x1* **and** *a*′ = *ConstInt32 x2*
      thus *?thesis*
        using *assms outer-Cons Cons*
        by (*cases app-relop-i x292 x2 x1*) *auto*
    next
      fix *x1 x2*
      assume *x291* = *T-i64 a* = *ConstInt64 x1* **and** *a*′ = *ConstInt64 x2*
      thus *?thesis*
        using *assms outer-Cons Cons*
        by (*cases app-relop-i x292 x2 x1*) *auto*
    qed *auto*
  qed (*cases x291*; *cases a*; *auto*)
qed (*cases x291*; *auto*)

lemma *run-one-step-basic-relop-f-result*:
  assumes *run-one-step d i* (*s*,*vs*,*ves*,$*Relop-f x301 x302*) = (*s*′, *vs*′, *res*)
  shows (∃ *r*. *res* = *RSNormal r*) ∨ (∃ *e*. *res* = *RSCrash e*)
  using *assms*
proof (*cases ves*)
  case (*Cons a list*)
  note *outer-Cons* = *Cons*
  thus *?thesis*
    using *assms*
  proof (*cases list*)
    case (*Cons a*′ *list*′)
    thus *?thesis*
      using *assms outer-Cons*
    proof (*cases x301*; *cases a*; *cases a*′)
      fix *x1 x2*
      assume *x301* = *T-f32 a* = *ConstFloat32 x1* **and** *a*′ = *ConstFloat32 x2*
      thus *?thesis*
        using *assms outer-Cons Cons*

by (*cases app-relop-f x302 x2 x1*) *auto*
next
fix *x1 x2*
assume *x301* = *T-f64 a* = *ConstFloat64 x1* **and** *a'* = *ConstFloat64 x2*
thus *?thesis*
using *assms outer-Cons Cons*
by (*cases app-relop-f x302 x2 x1*) *auto*
qed *auto*
qed (*cases x301*; *cases a*; *auto*)
qed (*cases x301*; *auto*)

lemma *run-one-step-basic-cvtop-result*:
assumes *run-one-step d i* (*s*,*vs*,*ves*,$*Cvtop t2 x312 t1 sx*) = (*s'*, *vs'*, *res*)
shows (∃ *r*. *res* = *RSNormal r*) ∨ (∃ *e*. *res* = *RSCrash e*)
using *assms*
proof (*cases ves*; *cases x312*)
fix *a ves'*
assume *ves* = *a*#*ves'* **and** *x312* = *Convert*
thus *?thesis*
using *assms*
by (*cases cvt t2 sx a*; *cases types-agree t1 a*) *auto*
next
fix *a ves'*
assume *ves* = *a*#*ves'* **and** *x312* = *Reinterpret*
thus *?thesis*
using *assms*
by (*cases sx*; *cases types-agree t1 a*) *auto*
qed *auto*

lemma *run-one-step-trap-result*:
assumes *run-one-step d i* (*s*,*vs*,*ves*,*Trap*) = (*s'*, *vs'*, *res*)
shows ∃ *e*. *res* = *RSCrash e*
using *assms*
by *auto*

lemma *run-one-step-callcl-result*:
assumes *run-one-step d i* (*s*,*vs*,*ves*,*Callcl cl*) = (*s'*, *vs'*, *res*)
shows (∃ *r*. *res* = *RSNormal r*) ∨ (∃ *e*. *res* = *RSCrash e*)
proof −
obtain *t1s t2s* **where** *cl-type-is*:*cl-type cl* = (*t1s* -> *t2s*)
using *tf.exhaust*
by *blast*
obtain *ves' ves''* **where** *split-n-is*:*split-n ves* (*length t1s*) = (*ves'*, *ves''*)
by *fastforce*
show *?thesis*
proof (*cases cl*)
case (*Func-native x11 x12 x13 x14*)
thus *?thesis*
using *assms cl-type-is split-n-is*

unfolding *cl-type-def*
by (*cases length t1s* ≤ *length ves*) *auto*
next
case (*Func-host x21 x22*)
show *?thesis*
proof (*cases host-apply-impl s* (*t1s* -> *t2s*) *x22* (*rev ves′*))
case *None*
thus *?thesis*
using *assms cl-type-is split-n-is Func-host*
unfolding *cl-type-def*
by (*cases length t1s* ≤ *length ves*) *auto*
next
case (*Some a*)
thus *?thesis*
proof (*cases a*)
case (*Pair s′ vcs′*)
thus *?thesis*
using *assms cl-type-is split-n-is Func-host Some*
unfolding *cl-type-def*
by (*cases length t1s* ≤ *length ves*; *cases list-all2 types-agree t2s vcs′*) *auto*
qed
qed
qed
qed

lemma *run-one-step-label-result*:
assumes *run-one-step d i* (*s*,*vs*,*ves*,*Label x41 x42 x43*) = (*s′*, *vs′*, *res*)
shows (∃ *r*. *res* = *RSNormal r*) ∨ (∃ *r rvs*. *res* = *RSBreak r rvs*) ∨ (∃ *rvs*. *res* = *RSReturn rvs*) ∨ (∃ *e*. *res* = *RSCrash e*)
using *assms*
by (*cases res*) *auto*

lemma *run-one-step-local-result*:
assumes *run-one-step d i* (*s*,*vs*,*ves*,*Local x51 x52 x53 x54*) = (*s′*, *vs′*, *res*)
shows (∃ *r*. *res* = *RSNormal r*) ∨ (∃ *e*. *res* = *RSCrash e*)
using *assms*
proof (*cases x54* = [*Trap*])
case *False*
note *outer-False* = *False*
thus *?thesis*
proof (*cases const-list x54*)
case *True*
thus *?thesis*
using *assms outer-False*
by (*cases length x54* = *x51*) *auto*
next
case *False*
thus *?thesis*
using *assms outer-False*

```
    proof (cases d)
      case (Suc d')
      obtain s' vs' res where rs-def:run-step d' x52 (s, x53, x54) = (s', vs', res)
      by (metis surj-pair)
      thus ?thesis
        using assms outer-False False Suc
      proof (cases res)
        case (RSReturn x3)
        thus ?thesis
          using assms outer-False False rs-def Suc
          by (cases x51 ≤ length x3) auto
      qed auto
    qed auto
  qed
qed auto

lemma run-one-step-break:
  assumes run-one-step d i (s,vs,ves,e) = (s', vs', RSBreak n res)
  shows (e = $Br n) ∨ (∃ n les es. e = Label n les es)
proof (cases e)
  case (Basic x1)
  thus ?thesis
  proof (cases x1)
    case Unreachable
    thus ?thesis
      using run-one-step-basic-unreachable-result assms Basic
      by fastforce
  next
    case Nop
    thus ?thesis
      using assms Basic
      by fastforce
  next
    case Drop
    thus ?thesis
      using run-one-step-basic-drop-result assms Basic
      by fastforce
  next
    case Select
    thus ?thesis
      using run-one-step-basic-select-result assms Basic
      by fastforce
  next
    case (Block x51 x52)
    thus ?thesis
      using run-one-step-basic-block-result assms Basic
      by fastforce
  next
    case (Loop x61 x62)
```

thus *?thesis*
using *run-one-step-basic-loop-result assms Basic*
by *fastforce*
next
case (*If x71 x72 x73*)
thus *?thesis*
using *run-one-step-basic-if-result assms Basic*
by *fastforce*
next
case (*Br x8*)
thus *?thesis*
using *run-one-step-basic-br-result assms Basic*
by *fastforce*
next
case (*Br-if x9*)
thus *?thesis*
using *run-one-step-basic-br-if-result assms Basic*
by *fastforce*
next
case (*Br-table x10*)
thus *?thesis*
using *run-one-step-basic-br-table-result assms Basic*
by *fastforce*
next
case *Return*
thus *?thesis*
using *run-one-step-basic-return-result assms Basic*
by *fastforce*
next
case (*Call x12*)
thus *?thesis*
using *run-one-step-basic-call-result assms Basic*
by *fastforce*
next
case (*Call-indirect x13*)
thus *?thesis*
using *run-one-step-basic-call-indirect-result assms Basic*
by *fastforce*
next
case (*Get-local x14*)
thus *?thesis*
using *run-one-step-basic-get-local-result assms Basic*
by *fastforce*
next
case (*Set-local x15*)
thus *?thesis*
using *run-one-step-basic-set-local-result assms Basic*
by *fastforce*
next

```
  case (Tee-local x16)
  thus ?thesis
    using run-one-step-basic-tee-local-result assms Basic
    by fastforce
next
  case (Get-global x17)
  thus ?thesis
    using assms Basic
    by fastforce
next
  case (Set-global x18)
  thus ?thesis
    using run-one-step-basic-set-global-result assms Basic
    by fastforce
next
  case (Load x191 x192 x193 x194)
  thus ?thesis
    using run-one-step-basic-load-result assms Basic
    by fastforce
next
  case (Store x201 x202 x203 x204)
  thus ?thesis
    using run-one-step-basic-store-result assms Basic
    by fastforce
next
  case Current-memory
  thus ?thesis
    using run-one-step-basic-current-memory-result assms Basic
    by fastforce
next
  case Grow-memory
  thus ?thesis
    using run-one-step-basic-grow-memory-result assms Basic
    by fastforce
next
  case (EConst x23)
  thus ?thesis
    using assms Basic
    by fastforce
next
  case (Unop-i x241 x242)
  thus ?thesis
    using run-one-step-basic-unop-i-result assms Basic
    by fastforce
next
  case (Unop-f x251 x252)
  thus ?thesis
    using run-one-step-basic-unop-f-result assms Basic
    by fastforce
```

```
  next
    case (Binop-i x261 x262)
    thus ?thesis
      using run-one-step-basic-binop-i-result assms Basic
      by fastforce
  next
    case (Binop-f x271 x272)
    thus ?thesis
      using run-one-step-basic-binop-f-result assms Basic
      by fastforce
  next
    case (Testop x281 x282)
    thus ?thesis
      using run-one-step-basic-testop-result assms Basic
      by fastforce
  next
    case (Relop-i x291 x292)
    thus ?thesis
      using run-one-step-basic-relop-i-result assms Basic
      by fastforce
  next
    case (Relop-f x301 x302)
    thus ?thesis
      using run-one-step-basic-relop-f-result assms Basic
      by fastforce
  next
    case (Cvtop x311 x312 x313 x314)
    thus ?thesis
      using run-one-step-basic-cvtop-result assms Basic
      by fastforce
  qed
next
  case Trap
  thus ?thesis
    using assms
    by auto
next
  case (Callcl x3)
  thus ?thesis
    using assms run-one-step-callcl-result
    by fastforce
next
  case (Label x41 x42 x43)
  thus ?thesis
    by auto
next
  case (Local x51 x52 x53 x54)
  thus ?thesis
    using assms run-one-step-local-result
```

by *fastforce*
qed

lemma *run-one-step-return*:
assumes *run-one-step d i* $(s,vs,ves,e) = (s', vs', RSReturn\ res)$
shows $(e = \$Return) \vee (\exists\, n\ les\ es.\ e = Label\ n\ les\ es)$
proof (*cases e*)
case (*Basic x1*)
thus *?thesis*
proof (*cases x1*)
case *Unreachable*
thus *?thesis*
using *run-one-step-basic-unreachable-result assms Basic*
by *fastforce*
next
case *Nop*
thus *?thesis*
using *assms Basic*
by *fastforce*
next
case *Drop*
thus *?thesis*
using *run-one-step-basic-drop-result assms Basic*
by *fastforce*
next
case *Select*
thus *?thesis*
using *run-one-step-basic-select-result assms Basic*
by *fastforce*
next
case (*Block x51 x52*)
thus *?thesis*
using *run-one-step-basic-block-result assms Basic*
by *fastforce*
next
case (*Loop x61 x62*)
thus *?thesis*
using *run-one-step-basic-loop-result assms Basic*
by *fastforce*
next
case (*If x71 x72 x73*)
thus *?thesis*
using *run-one-step-basic-if-result assms Basic*
by *fastforce*
next
case (*Br x8*)
thus *?thesis*
using *run-one-step-basic-br-result assms Basic*
by *fastforce*

next
  case (*Br-if x9*)
  thus *?thesis*
    using *run-one-step-basic-br-if-result assms Basic*
    by *fastforce*
next
  case (*Br-table x10*)
  thus *?thesis*
    using *run-one-step-basic-br-table-result assms Basic*
    by *fastforce*
next
  case *Return*
  thus *?thesis*
    using *run-one-step-basic-return-result assms Basic*
    by *fastforce*
next
  case (*Call x12*)
  thus *?thesis*
    using *run-one-step-basic-call-result assms Basic*
    by *fastforce*
next
  case (*Call-indirect x13*)
  thus *?thesis*
    using *run-one-step-basic-call-indirect-result assms Basic*
    by *fastforce*
next
  case (*Get-local x14*)
  thus *?thesis*
    using *run-one-step-basic-get-local-result assms Basic*
    by *fastforce*
next
  case (*Set-local x15*)
  thus *?thesis*
    using *run-one-step-basic-set-local-result assms Basic*
    by *fastforce*
next
  case (*Tee-local x16*)
  thus *?thesis*
    using *run-one-step-basic-tee-local-result assms Basic*
    by *fastforce*
next
  case (*Get-global x17*)
  thus *?thesis*
    using *assms Basic*
    by *fastforce*
next
  case (*Set-global x18*)
  thus *?thesis*
    using *run-one-step-basic-set-global-result assms Basic*

by *fastforce*
next
case (*Load x191 x192 x193 x194*)
thus *?thesis*
using *run-one-step-basic-load-result assms Basic*
by *fastforce*
next
case (*Store x201 x202 x203 x204*)
thus *?thesis*
using *run-one-step-basic-store-result assms Basic*
by *fastforce*
next
case *Current-memory*
thus *?thesis*
using *run-one-step-basic-current-memory-result assms Basic*
by *fastforce*
next
case *Grow-memory*
thus *?thesis*
using *run-one-step-basic-grow-memory-result assms Basic*
by *fastforce*
next
case (*EConst x23*)
thus *?thesis*
using *assms Basic*
by *fastforce*
next
case (*Unop-i x241 x242*)
thus *?thesis*
using *run-one-step-basic-unop-i-result assms Basic*
by *fastforce*
next
case (*Unop-f x251 x252*)
thus *?thesis*
using *run-one-step-basic-unop-f-result assms Basic*
by *fastforce*
next
case (*Binop-i x261 x262*)
thus *?thesis*
using *run-one-step-basic-binop-i-result assms Basic*
by *fastforce*
next
case (*Binop-f x271 x272*)
thus *?thesis*
using *run-one-step-basic-binop-f-result assms Basic*
by *fastforce*
next
case (*Testop x281 x282*)
thus *?thesis*

using *run-one-step-basic-testop-result assms Basic*
by *fastforce*
next
case (*Relop-i x291 x292*)
thus *?thesis*
using *run-one-step-basic-relop-i-result assms Basic*
by *fastforce*
next
case (*Relop-f x301 x302*)
thus *?thesis*
using *run-one-step-basic-relop-f-result assms Basic*
by *fastforce*
next
case (*Cvtop x311 x312 x313 x314*)
thus *?thesis*
using *run-one-step-basic-cvtop-result assms Basic*
by *fastforce*
qed
next
case *Trap*
thus *?thesis*
using *assms*
by *auto*
next
case (*Callcl x3*)
thus *?thesis*
using *assms run-one-step-callcl-result*
by *fastforce*
next
case (*Label x41 x42 x43*)
thus *?thesis*
by *auto*
next
case (*Local x51 x52 x53 x54*)
thus *?thesis*
using *assms run-one-step-local-result*
by *fastforce*
qed

lemma *run-step-break-imp-not-trap-const-list*:
assumes *run-step d i* $(s, vs, es) = (s', vs', RSBreak\ n\ res)$
shows $es \neq [Trap]$ $\neg$*const-list es*
proof −
{
assume $es = [Trap]$
hence *False*
using *assms*
by *simp*
}

thus *es* $\neq$ [*Trap*]
by *blast*
{
assume *const-list es*
then obtain *vs* **where** *split-vals-e es* = (*vs*, [])
using *split-vals-e-const-list e-type-const-conv-vs*
by *fastforce*
hence *False*
using *assms*
by *simp*
}
thus ¬*const-list es*
by *blast*
qed

lemma *run-step-return-imp-not-trap-const-list*:
assumes *run-step d i* (*s*, *vs*, *es*) = (*s*′, *vs*′, *RSReturn res*)
shows *es* $\neq$ [*Trap*] ¬*const-list es*
proof −
{
assume *es* = [*Trap*]
hence *False*
using *assms*
by *simp*
}
thus *es* $\neq$ [*Trap*]
by *blast*
{
assume *const-list es*
then obtain *vs* **where** *split-vals-e es* = (*vs*, [])
using *split-vals-e-const-list e-type-const-conv-vs*
by *fastforce*
hence *False*
using *assms*
by *simp*
}
thus ¬*const-list es*
by *blast*
qed

lemma *run-one-step-label-break-imp-break*:
assumes *run-one-step d i* (*s*, *vs*, *ves*, *Label ln les es*) = (*s*′, *vs*′, *RSBreak n res*)
shows *run-step d i* (*s*, *vs*, *es*) = (*s*′, *vs*′, *RSBreak* (*Suc n*) *res*)
using *assms*
proof (*cases es* = [*Trap*]; *cases const-list es*)
assume *local-assms*:*es* $\neq$ [*Trap*] ¬*const-list es*
obtain *s*″ *vs*″ *res*″ **where** *rs-def*:*run-step d i* (*s*, *vs*, *es*) = (*s*″, *vs*″, *res*″)
by (*metis surj-pair*)
thus *?thesis*

using *assms local-assms*
proof (*cases res''*)
case (*RSBreak x21 x22*)
thus *?thesis*
using *assms local-assms rs-def*
by (*cases x21*; *cases ln* $\leq$ *length x22*) *auto*
qed *auto*
qed *auto*

lemma *run-one-step-label-return-imp-return*:
assumes *run-one-step d i* (*s*, *vs*, *ves*, *Label n les es*) = (*s'*, *vs'*, *RSReturn res*)
shows *run-step d i* (*s*, *vs*, *es*) = (*s'*, *vs'*, *RSReturn res*)
using *assms*
proof (*cases es* = [*Trap*]; *cases const-list es*)
assume *local-assms*:*es* $\neq$ [*Trap*] $\neg$*const-list es*
obtain *s'' vs'' res''* **where** *rs-def*:*run-step d i* (*s*, *vs*, *es*) = (*s''*, *vs''*, *res''*)
by (*metis surj-pair*)
thus *?thesis*
using *assms local-assms*
proof (*cases res''*)
case (*RSBreak x21 x22*)
thus *?thesis*
using *assms local-assms rs-def*
by (*cases x21*; *cases n* $\leq$ *length x22*) *auto*
qed *auto*
qed *auto*
thm *run-step-run-one-step.induct*

definition *run-step-break-imp-lfilled-prop* **where**
run-step-break-imp-lfilled-prop s' vs' n res =
(λ*d i* (*s*,*vs*,*es*). (*run-step d i* (*s*,*vs*,*es*) = (*s'*, *vs'*, *RSBreak n res*)) $\longrightarrow$
s = *s'* $\wedge$ *vs* = *vs'* $\wedge$
($\exists$ *n' lfilled es-c*. *n'* $\geq$ *n* $\wedge$ *Lfilled-exact* (*n'*−*n*) *lfilled* ((*vs-to-es res*) @ [$*Br n'*] @ *es-c*) *es*))

definition *run-one-step-break-imp-lfilled-prop* **where**
run-one-step-break-imp-lfilled-prop s' vs' n res =
(λ*d i* (*s*,*vs*,*ves*,*e*). *run-one-step d i* (*s*,*vs*,*ves*,*e*) = (*s'*, *vs'*, *RSBreak n res*) $\longrightarrow$
s = *s'* $\wedge$ *vs* = *vs'* $\wedge$ ((*res* = *ves* $\wedge$ *e* = $*Br n*) $\vee$ ($\exists$ *n' lfilled es-c es les' ln*. *n'* > *n* $\wedge$ *Lfilled-exact* (*n'*−(*n*+*1*)) *lfilled* ((*vs-to-es res*) @ [$*Br n'*] @ *es-c*) *es* $\wedge$ *e* = *Label ln les' es*)))

lemma *run-step-break-imp-lfilled*:
assumes *run-step d i* (*s*,*vs*,*es*) = (*s'*, *vs'*, *RSBreak n res*)
shows *s* = *s'* $\wedge$
vs = *vs'* $\wedge$
($\exists$ *n' lfilled es-c*. *n'* $\geq$ *n* $\wedge$
Lfilled-exact (*n'*−*n*) *lfilled* ((*vs-to-es res*) @ [$*Br n'*] @ *es-c*)

es)
proof −
  fix *ves e*
  have (*run-step-break-imp-lfilled-prop s′ vs′ n res*) *d i* (*s*,*vs*,*es*)
  and (*run-one-step-break-imp-lfilled-prop s′ vs′ n res*) *d i* (*s*,*vs*,*ves*,*e*)
  proof (*induction d i* (*s*,*vs*,*es*) **and** *d i* (*s*,*vs*,*ves*,*e*) *arbitrary*: *n es* **and** *n ves e* *rule*: *run-step-run-one-step.induct*)
    case (*1 d i es*)
    {
      assume *local-assms*:*run-step d i* (*s*,*vs*,*es*) = (*s′*, *vs′*, *RSBreak n res*)
      obtain *ves es′* **where** *split-vals-es*:*split-vals-e es* = (*ves*, *es′*)
        by (*metis surj-pair*)
      then obtain *a as* **where** *es′-def*:*es′* = *a#as*
        using *local-assms*
        by (*cases es′*) *auto*
      hence *a-def*:*a* ≠ *Trap*
        using *local-assms split-vals-es*
        by (*cases a* = *Trap*; *cases* (*as* ≠ [] ∨ *ves* ≠ [])) *simp-all*
     obtain *s′′ vs′′ res′′* **where** *run-one-step d i* (*s*,*vs*,(*rev ves*),*a*) = (*s′′*, *vs′′*, *res′′*)
        by (*metis surj-pair*)
      hence *ros-def*:*run-one-step d i* (*s*,*vs*,(*rev ves*),*a*) = (*s′*, *vs′*, *RSBreak n res*)
        using *local-assms split-vals-es es′-def a-def*
        by (*cases res′′*) (*auto simp del*: *run-one-step.simps*)
      hence *run-one-step-break-imp-lfilled-prop s′ vs′ n res d i* (*s*, *vs*, *rev ves*, *a*)
        using *1 split-vals-es a-def es′-def*
        by *fastforce*
      then obtain *n′ lfilled es-c les les′ ln* **where**
        s = *s′ vs* = *vs′*
        ((*res* = (*rev ves*) ∧ *a* = $*Br n*) ∨
            n′ > *n* ∧ (*Lfilled-exact* (*n′*−(*n*+*1*)) *lfilled* ((*vs-to-es res*) @ [$*Br n′*] @ *es-c*) *les* ∧ *a* = *Label ln les′ les*))
        using *ros-def*
        unfolding *run-one-step-break-imp-lfilled-prop-def*
        by *fastforce*
      then consider
        (*1*) *s* = *s′ vs* = *vs′ res* = (*rev ves*) *a* = $*Br n*
      | (*2*) *s* = *s′ vs* = *vs′ n′* > *n Lfilled-exact* (*n′*−(*n*+*1*)) *lfilled* ((*vs-to-es res*) @ [$*Br n′*] @ *es-c*) *les a* = *Label ln les′ les*
        by *blast*
      hence *s* = *s′* ∧ *vs* = *vs′* ∧
            (∃ *n′ lfilled es-c*. *n′* ≥ *n* ∧ *Lfilled-exact* (*n′*−*n*) *lfilled* ((*vs-to-es res*) @ [$*Br n′*] @ *es-c*) *es*)
      proof *cases*
        case *1*
      thus *?thesis*
         using *es′-def split-vals-e-conv-app*[*OF split-vals-es*] *Lfilled-exact.intros*(*1*) *is-const-list*[*of - ves*]
        by *fastforce*
      next

case *2*
have *test*:*const-list* ($$∗ *ves*)
using *is-const-list*
by *auto*
have $(Suc\ (n' - Suc\ n)) = n' - n$
using *2*(*3*)
by *simp*
thus *?thesis*
using *2*(*1*,*2*,*3*,*5*) *Lfilled-exact.intros*(*2*)[*OF test - 2*(*4*), *of - ln les′ as*] *es′-def split-vals-e-conv-app*[*OF split-vals-es*]
by (*metis Suc-eq-plus1 append-Cons append-Nil less-imp-le-nat*)
qed
}
thus *?case*
unfolding *run-step-break-imp-lfilled-prop-def*
by *fastforce*
next
case (*2 d i ves e*)
{
assume *local-assms*:*run-one-step d i* (*s*,*vs*,*ves*,*e*) = (*s′*, *vs′*, *RSBreak n res*)
consider (*a*) *e* = $*Br n* | (*b*) (∃ *n les es*. *e* = *Label n les es*)
using *run-one-step-break*[*OF local-assms*]
by *blast*
hence *s* = *s′* ∧ *vs* = *vs′* ∧ ((*res* = *ves* ∧ *e* = $*Br n*) ∨ (∃ *n′ lfilled es-c es les′ ln*. *n′* > *n* ∧ *Lfilled-exact* (*n′*−(*n*+*1*)) *lfilled* ((*vs-to-es res*) @ [$*Br n′*] @ *es-c*) *es* ∧ *e* = *Label ln les′ es*))
proof *cases*
case *a*
thus *?thesis*
using *local-assms*
by *simp*
next
case *b*
then obtain *ln les es* **where** *e-def*:*e* = *Label ln les es*
by *blast*
hence *run-one-step d i* (*s*, *vs*, *ves*, *Label ln les es*) = (*s′*, *vs′*, *RSBreak n res*)
using *local-assms* **by** *simp*
hence *rs-def*:*run-step d i* (*s*, *vs*, *es*) = (*s′*, *vs′*, *RSBreak* (*Suc n*) *res*)
using *run-one-step-label-break-imp-break*
by *fastforce*
hence *run-step-break-imp-lfilled-prop s′ vs′* (*Suc n*) *res d i* (*s*, *vs*, *es*)
using *2*(*1*)[*OF e-def - run-step-break-imp-not-trap-const-list*(*2*)]
by *fastforce*
thus *?thesis*
using *e-def rs-def*
unfolding *run-step-break-imp-lfilled-prop-def*
by *fastforce*
qed
}

thus *?case*
unfolding *run-one-step-break-imp-lfilled-prop-def*
by *fastforce*
qed
thus *?thesis*
using *assms*
unfolding *run-step-break-imp-lfilled-prop-def*
by *fastforce*
qed

lemma *run-step-return-imp-lfilled*:
assumes *run-step d i* (*s*,*vs*,*es*) = (*s'*, *vs'*, *RSReturn res*)
shows *s* = *s'* ∧ *vs* = *vs'* ∧ (∃ *n lfilled es-c*. *Lfilled-exact n lfilled* ((*vs-to-es res*) @ [$*Return*] @ *es-c*) *es*)
proof −
fix *ves e*
have (*run-step d i* (*s*,*vs*,*es*) = (*s'*, *vs'*, *RSReturn res*)) ⟹
s = *s'* ∧ *vs* = *vs'* ∧ (∃ *n lfilled es-c*. *Lfilled-exact n lfilled* ((*vs-to-es res*) @ [$*Return*] @ *es-c*) *es*)
and (*run-one-step d i* (*s*,*vs*,*ves*,*e*) = (*s'*, *vs'*, *RSReturn res*)) ⟹
s = *s'* ∧ *vs* = *vs'* ∧
((*res* = *ves* ∧ *e* = $*Return*) ∨
(∃ *n lfilled ves es-c es n' les'*. *Lfilled-exact n lfilled* ((*vs-to-es res*) @ [$*Return*] @ *es-c*) *es* ∧
e = *Label n' les' es*))
proof (*induction d i* (*s*,*vs*,*es*) **and** *d i* (*s*,*vs*,*ves*,*e*) *arbitrary*: *s vs es s' vs' res* **and** *s vs ves e s' vs' res rule*: *run-step-run-one-step.induct*)
case (*1 d i s vs es*)
obtain *ves es'* **where** *split-vals-es*:*split-vals-e es* = (*ves*, *es'*)
by (*metis surj-pair*)
then obtain *a as* **where** *es'-def*:*es'* = *a*#*as*
using *1*(*2*)
by (*cases es'*) *auto*
hence *a-def*:¬ *e-is-trap a*
using *1*(*2*) *split-vals-es*
by (*cases a* = *Trap*; *cases* (*as* ≠ [] ∨ *ves* ≠ [])) *simp-all*
obtain *s'' vs'' res''* **where** *run-one-step d i* (*s*,*vs*,(*rev ves*),*a*) = (*s''*, *vs''*, *res''*)
by (*metis surj-pair*)
hence *ros-def*:*run-one-step d i* (*s*,*vs*,(*rev ves*),*a*) = (*s'*, *vs'*, *RSReturn res*)
using *1*(*2*) *split-vals-es es'-def a-def*
by (*cases res''*) (*auto simp del*: *run-one-step.simps*)
obtain *n lfilled les-c les n' les'* **where**
s = *s' vs* = *vs'*
(*res* = *rev ves* ∧ *a* = $*Return*) ∨ (*Lfilled-exact n lfilled* ((*vs-to-es res*) @ [$*Return*] @ *les-c*) *les* ∧ *a* = *Label n' les' les*)
using *1*(*1*)[*OF split-vals-es*[*symmetric*] - *es'-def a-def ros-def*]
by *fastforce*
then consider
(*1*) *s* = *s' vs* = *vs' res* = *rev ves a* = $*Return*

| (2) $s = s'$ $vs = vs'$ (*Lfilled-exact n lfilled* ((*vs-to-es res*) @ [$*Return*] @ *les-c*) *les*) (a = *Label* n' les' *les*)
  by *blast*
thus *?case*
proof *cases*
  case *1*
  thus *?thesis*
    using *es'-def split-vals-e-conv-app*[*OF split-vals-es*] *Lfilled-exact.intros*(*1*) *is-const-list*[*of - ves*]
    by *fastforce*
next
  case *2*
  have *const-list* ($$* *ves*)
    using *is-const-list*
    by *fastforce*
  thus *?thesis*
    using *2 Lfilled-exact.intros*(*2*) *es'-def split-vals-e-conv-app*[*OF split-vals-es*]
      by *fastforce*
qed
next
  case (*2 d i s vs ves e* s' vs')
  consider (*a*) e = $*Return* | (*b*) ($\exists$ *n les es*. e = *Label n les es*)
    using *run-one-step-return*[*OF 2*(*3*)]
    by *blast*
  thus *?case*
  proof *cases*
    case *a*
    thus *?thesis*
      using *2*(*3*)
      by *simp*
  next
    case *b*
    then obtain *n les es* **where** *e-def*:e = *Label n les es*
      by *blast*
    hence *run-one-step d i* (*s*, *vs*, *ves*, *Label n les es*) = (s', vs', *RSReturn res*)
      using *2*(*3*) **by** *simp*
    hence *run-step d i* (*s*, *vs*, *es*) = (s', vs', *RSReturn res*)
      using *run-one-step-label-return-imp-return*
      by *fastforce*
    thus *?thesis*
      using *2*(*3*) *2*(*1*)[*OF e-def - run-step-return-imp-not-trap-const-list*(*2*)] *e-def*
      by *fastforce*
  qed
qed
thus *?thesis*
  using *assms*
  by *blast*
qed

```
lemma run-step-basic-unop-testop-sound:
  assumes (run-one-step d i (s,vs,ves,$b-e) = (s', vs', RSNormal es'))
          b-e = Unop-i t iop ∨ b-e = Unop-f t fop ∨ b-e = Testop t testop
  shows ⦇s;vs;(vs-to-es ves)@[$b-e]⦈ ⇝- i ⦇s';vs';es'⦈
proof -
  consider (1) b-e = Unop-i t iop | (2) b-e = Unop-f t fop | (3) b-e = Testop t
testop
    using assms(2)
    by blast
  note b-e-cases = this
  show ?thesis
    using assms(1)
  proof (cases ves)
    case (Cons a list)
    thus ?thesis
      using assms(1)
    proof (cases a; cases t)
      case (ConstInt32 x1)
      case T-i32
      thus ?thesis
        using assms(1) Cons ConstInt32
           is-const-list-vs-to-es-list[of rev list]
            progress-L0-left[OF reduce.intros(1)[OF reduce-simple.intros(1)]]
            progress-L0-left[OF reduce.intros(1)[OF reduce-simple.intros(13)]]
        by (cases rule: b-e-cases) auto
    next
      case (ConstInt64 x2)
      case T-i64
      thus ?thesis
        using assms(1) Cons ConstInt64
           is-const-list-vs-to-es-list[of rev list]
            progress-L0-left[OF reduce.intros(1)[OF reduce-simple.intros(2)]]
            progress-L0-left[OF reduce.intros(1)[OF reduce-simple.intros(14)]]
        by (cases rule: b-e-cases) auto
    next
      case (ConstFloat32 x3)
      case T-f32
      thus ?thesis
        using assms(1) Cons ConstFloat32
           is-const-list-vs-to-es-list[of rev list]
            progress-L0-left[OF reduce.intros(1)[OF reduce-simple.intros(3)]]
        by (cases rule: b-e-cases) auto
    next
      case (ConstFloat64 x4)
      case T-f64
      thus ?thesis
        using assms(1) Cons ConstFloat64
           is-const-list-vs-to-es-list[of rev list]
            progress-L0-left[OF reduce.intros(1)[OF reduce-simple.intros(4)]]
```

by (*cases rule*: *b-e-cases*) *auto*
qed (*cases rule*: *b-e-cases*; *auto*)+
qed (*cases rule*: *b-e-cases*; *cases t*; *auto*)
qed

lemma *run-step-basic-binop-relop-sound*:
assumes (*run-one-step d i* (*s*,*vs*,*ves*,$*b-e*) = (*s′*, *vs′*, *RSNormal es′*))
b-e = *Binop-i t iop* ∨ *b-e* = *Binop-f t fop* ∨ *b-e* = *Relop-i t irop* ∨ *b-e* = *Relop-f t frop*
shows ⦇*s*;*vs*;(*vs-to-es ves*)@[$*b-e*]⦈ ⇝- *i* ⦇*s′*;*vs′*;*es′*⦈
proof −
consider
(*1*) *b-e* = *Binop-i t iop*
| (*2*) *b-e* = *Binop-f t fop*
| (*3*) *b-e* = *Relop-i t irop*
| (*4*) *b-e* = *Relop-f t frop*
using *assms*(*2*)
by *blast*
note *b-e-cases* = *this*
show *?thesis*
using *assms*(*1*)
proof (*cases ves*)
case *outer-Cons*:(*Cons v1 list*)
thus *?thesis*
using *assms*(*1*)
proof (*cases list*)
case (*Cons v2 list′*)
thus *?thesis*
using *assms*(*1*) *outer-Cons*
proof (*cases v1*; *cases v2*; *cases t*)
fix *c1 c2*
assume *v1* = *ConstInt32 c1* **and** *v2* = *ConstInt32 c2* **and** *t* = *T-i32*
thus *?thesis*
using *assms*(*1*) *Cons outer-Cons*
is-const-list-vs-to-es-list[*of rev list′*]
progress-L0-left[*OF reduce.intros*(*1*)[*OF reduce-simple.intros*(*5*)]]
progress-L0-left[*OF reduce.intros*(*1*)[*OF reduce-simple.intros*(*6*)]]
progress-L0-left[*OF reduce.intros*(*1*)[*OF reduce-simple.intros*(*15*)]]
by (*cases rule*: *b-e-cases*; *cases app-binop-i iop c2 c1*) *auto*
next
fix *c1 c2*
assume *v1* = *ConstInt64 c1* **and** *v2* = *ConstInt64 c2* **and** *t* = *T-i64*
thus *?thesis*
using *assms*(*1*) *Cons outer-Cons*
is-const-list-vs-to-es-list[*of rev list′*]
progress-L0-left[*OF reduce.intros*(*1*)[*OF reduce-simple.intros*(*7*)]]
progress-L0-left[*OF reduce.intros*(*1*)[*OF reduce-simple.intros*(*8*)]]
progress-L0-left[*OF reduce.intros*(*1*)[*OF reduce-simple.intros*(*16*)]]
by (*cases rule*: *b-e-cases*; *cases app-binop-i iop c2 c1*) *auto*

```
      next
        fix c1 c2
        assume v1 = ConstFloat32 c1 and v2 = ConstFloat32 c2 and t = T-f32
        thus ?thesis
          using assms(1) Cons outer-Cons
             is-const-list-vs-to-es-list[of rev list']
              progress-L0-left[OF reduce.intros(1)[OF reduce-simple.intros(9)]]
              progress-L0-left[OF reduce.intros(1)[OF reduce-simple.intros(10)]]
              progress-L0-left[OF reduce.intros(1)[OF reduce-simple.intros(17)]]
          by (cases rule: b-e-cases; cases app-binop-f fop c2 c1) auto
      next
        fix c1 c2
        assume v1 = ConstFloat64 c1 and v2 = ConstFloat64 c2 and t = T-f64
        thus ?thesis
          using assms(1) Cons outer-Cons
             is-const-list-vs-to-es-list[of rev list']
              progress-L0-left[OF reduce.intros(1)[OF reduce-simple.intros(11)]]
              progress-L0-left[OF reduce.intros(1)[OF reduce-simple.intros(12)]]
              progress-L0-left[OF reduce.intros(1)[OF reduce-simple.intros(18)]]
          by (cases rule: b-e-cases; cases app-binop-f fop c2 c1) auto
      qed (cases rule: b-e-cases; auto)+
    qed (cases rule: b-e-cases; cases t; cases v1; auto)
  qed (cases rule: b-e-cases; cases t; auto)
qed

lemma run-step-basic-sound:
  assumes (run-one-step d i (s,vs,ves,$b-e) = (s', vs', RSNormal es'))
  shows (|s;vs;(vs-to-es ves)@[$b-e]|) ⇝- i (|s';vs';es'|)
proof -
  show ?thesis
  proof (cases b-e)
    case Unreachable
    thus ?thesis
      using is-const-list-vs-to-es-list[of rev ves]
            progress-L0-left[OF reduce.intros(1)[OF reduce-simple.intros(22)]]
            assms
      by fastforce
  next
    case Nop
    thus ?thesis
      using is-const-list-vs-to-es-list[of rev ves]
            progress-L0-left[OF reduce.intros(1)[OF reduce-simple.intros(23)]]
            assms
      by fastforce
  next
    case Drop
    thus ?thesis
      using assms
    proof (cases ves)
```

```
    case (Cons a list)
    hence vs-to-es ves = vs-to-es list @ [$C a]
      by fastforce
    thus ?thesis
      using is-const-list-vs-to-es-list[of rev list]
            progress-L0-left[OF reduce.intros(1)[OF reduce-simple.intros(24)]]
            Drop assms Cons
      by auto
  qed auto
next
  case Select
  thus ?thesis
    using assms
  proof (cases ves)
    case outer-outer-cons:(Cons a list)
    thus ?thesis
      using Select assms
    proof (cases a; cases list)
      case (ConstInt32 x1a)
      case outer-cons:(Cons a' list')
      thus ?thesis
        using assms outer-outer-cons ConstInt32 Select
      proof (cases list')
        case (Cons a'' list'')
        hence vs-to-es ves = vs-to-es list'' @ [$C a'', $C a', $C ConstInt32 x1a]
          using outer-outer-cons outer-cons ConstInt32
          by fastforce
        thus ?thesis
          using is-const-list-vs-to-es-list[of rev list'']
                progress-L0-left[OF reduce.intros(1)]
                reduce-simple.intros(25,26)
                assms outer-outer-cons outer-cons Cons ConstInt32 Select
          by (cases int-eq x1a 0) auto
      qed auto
    qed auto
  qed auto
next
  case (Block x51 x52)
  thus ?thesis
  proof (cases x51)
    case (Tf t1s t2s)
    thus ?thesis
      using Block assms
    proof (cases length t1s ≤ length ves; cases split-n ves (length t1s))
      case True
      case (Pair ves' ves'')
      hence vs-to-es ves = vs-to-es ves'' @ vs-to-es ves'
        using split-n-conv-app
        by fastforce
```

moreover
have *s = s′ vs = vs′ es′ = vs-to-es ves′′ @ [Label (length t2s) [] (vs-to-es ves′ @ ($* x52))]*
using *Block assms Tf True Pair*
by *auto*
moreover
have *⦇s;vs;(vs-to-es ves′′)@(vs-to-es ves′)@[$Block x51 x52]⦈ ⇝-i ⦇s;vs;(vs-to-es ves′′)@[Label (length t2s) [] (vs-to-es ves′ @ ($* x52))]⦈*
using *Tf reduce-simple.intros(27) split-n-length[OF Pair True] progress-L0-left[OF reduce.intros(1)]*
is-const-list-vs-to-es-list[of rev ves′] is-const-list-vs-to-es-list[of rev ves′′]
by *fastforce*
ultimately
show *?thesis*
using *Block*
by *auto*
qed *auto*
qed
next
case *(Loop x61 x62)*
thus *?thesis*
proof *(cases x61)*
case *(Tf t1s t2s)*
thus *?thesis*
using *Loop assms*
proof *(cases length t1s ≤ length ves; cases split-n ves (length t1s))*
case *True*
case *(Pair ves′ ves′′)*
hence *vs-to-es ves = vs-to-es ves′′ @ vs-to-es ves′*
using *split-n-conv-app*
by *fastforce*
moreover
have *s = s′ vs = vs′ es′ = vs-to-es ves′′ @ [Label (length t1s) [$Loop x61 x62] (vs-to-es ves′ @ ($* x62))]*
using *Loop assms Tf True Pair*
by *auto*
moreover
have *⦇s;vs;(vs-to-es ves′′)@(vs-to-es ves′)@[$Loop x61 x62]⦈ ⇝-i ⦇s;vs;(vs-to-es ves′′)@[Label (length t1s) [$Loop x61 x62] (vs-to-es ves′ @ ($* x62))]⦈*
using *Tf reduce-simple.intros(28) split-n-length[OF Pair True] progress-L0-left[OF reduce.intros(1)]*
is-const-list-vs-to-es-list[of rev ves′] is-const-list-vs-to-es-list[of rev ves′′]
by *fastforce*
ultimately
show *?thesis*
using *Loop*
by *auto*
qed *auto*
qed

```
next
  case (If x71 x72 x73)
  thus ?thesis
    using assms
  proof (cases ves)
    case (Cons a list)
    thus ?thesis
      using assms If
    proof (cases a)
      case (ConstInt32 x1)
    hence vs-to-es ves = vs-to-es list @ [$C ConstInt32 x1]
      unfolding Cons
      by simp
    thus ?thesis
      using progress-L0-left[OF reduce.intros(1)]
            is-const-list-vs-to-es-list[of rev list]
            reduce-simple.intros(29,30)
            assms Cons If ConstInt32
      by (cases int-eq x1 0) auto
    qed auto
  qed auto
next
  case (Br x8)
  thus ?thesis
    using assms
    by auto
next
  case (Br-if x9)
  thus ?thesis
    using assms
  proof (cases ves)
    case (Cons a list)
    thus ?thesis
      using assms Br-if
    proof (cases a)
      case (ConstInt32 x1)
    hence vs-to-es ves = vs-to-es list @ [$C ConstInt32 x1]
      unfolding Cons
      by simp
    thus ?thesis
      using progress-L0-left[OF reduce.intros(1)[OF reduce-simple.intros(34)]]
            progress-L0-left[OF reduce.intros(1)[OF reduce-simple.intros(35)]]
            is-const-list-vs-to-es-list[of rev list]
            assms Cons Br-if ConstInt32
      by (cases int-eq x1 0) auto
    qed auto
  qed auto
next
  case (Br-table x10)
```

```
  thus ?thesis
    using assms
  proof (cases ves)
    case (Cons a list)
    thus ?thesis
      using assms Br-table
    proof (cases a)
      case (ConstInt32 x1)
      thus ?thesis
        using progress-L0-left[OF reduce.intros(1)[OF reduce-simple.intros(36)]]
              progress-L0-left[OF reduce.intros(1)[OF reduce-simple.intros(37)]]
              is-const-list-vs-to-es-list[of rev list]
              assms Br-table Cons
        by (cases (nat-of-int x1) < length x10) auto
    qed auto
  qed auto
next
  case Return
  thus ?thesis
    using assms
    by simp
next
  case (Call x12)
  thus ?thesis
    using assms progress-L0-left[OF reduce.intros(2)]
          is-const-list-vs-to-es-list[of rev ves]
    by auto
next
  case (Call-indirect x13)
  thus ?thesis
    using assms
  proof (cases ves)
    case (Cons a list)
    thus ?thesis
      using assms Call-indirect
    proof (cases a)
      case (ConstInt32 c)
      thus ?thesis
      proof (cases stab s i (nat-of-int c))
        case None
        thus ?thesis
          using assms Call-indirect Cons ConstInt32
                progress-L0-left[OF reduce.intros(4)]
                is-const-list-vs-to-es-list[of rev list]
          by auto
      next
        case (Some cl)
        thus ?thesis
        proof (cases stypes s i x13 = cl-type cl)
```

case *True*
hence (|*s*;*vs*;(*vs-to-es list*) @ [$*C ConstInt32 c*, $*Call-indirect x13*]|) ⇝- *i* (|*s*;*vs*;(*vs-to-es list*) @ [*Callcl cl*]|)
using *progress-L0-left*[*OF reduce.intros*(*3*)] *True Some is-const-list-vs-to-es-list*[*of rev list*]
by *fastforce*
thus *?thesis*
using *assms Call-indirect Cons ConstInt32 Some True*
by *auto*
next
case *False*
hence (|*s*;*vs*;(*vs-to-es list*)@[$*C ConstInt32 c*, $*Call-indirect x13*]|) ⇝- *i* (|*s*;*vs*;(*vs-to-es list*)@[*Trap*]|)
using *progress-L0-left*[*OF reduce.intros*(*4*)] *False Some is-const-list-vs-to-es-list*[*of rev list*]
by *fastforce*
thus *?thesis*
using *assms Call-indirect Cons ConstInt32 Some False*
by *auto*
qed
qed
qed *auto*
qed *auto*
next
case (*Get-local j*)
thus *?thesis*
using *assms*
proof (*cases j* < *length vs*)
case *True*
then obtain *vs1 v vs2* **where** *vs* = *vs1*@[*v*]@*vs2 length vs1* = *j*
using *id-take-nth-drop*
by *fastforce*
thus *?thesis*
using *assms Get-local True*
progress-L0-left[*OF reduce.intros*(*8*)]
is-const-list-vs-to-es-list[*of rev ves*]
by *auto*
qed *auto*
next
case (*Set-local j*)
thus *?thesis*
using *assms*
proof (*cases ves*)
case (*Cons a list*)
thus *?thesis*
using *assms Set-local*
proof (*cases j* < *length vs*)
case *True*
obtain *vs1 v vs2* **where** *vs-def*:*vs* = *vs1*@[*v*]@*vs2 length vs1* = *j*

```
        using id-take-nth-drop True
        by fastforce
      thus ?thesis
        using assms Set-local True Cons
              progress-L0-left[OF reduce.intros(9)]
              is-const-list-vs-to-es-list[of rev list]
        by auto
    qed auto
  qed auto
next
  case (Tee-local x16)
  thus ?thesis
    using assms
  proof (cases ves)
    case (Cons a list)
    thus ?thesis
      using assms Tee-local
            progress-L0-left[OF reduce.intros(1)[OF reduce-simple.intros(41)]]
                            is-const-list-vs-to-es-list[of rev list]
      by (auto simp add: is-const-def)
  qed auto
next
  case (Get-global x17)
  thus ?thesis
    using assms
          progress-L0-left[OF reduce.intros(10)]
          is-const-list-vs-to-es-list[of rev ves]
    by (auto simp add: is-const-def)
next
  case (Set-global x18)
  thus ?thesis
    using assms
  proof (cases ves)
    case (Cons a list)
    thus ?thesis
      using assms Set-global
            progress-L0-left[OF reduce.intros(11)]
            is-const-list-vs-to-es-list[of rev list]
      by (auto simp add: is-const-def)
  qed auto
next
  case (Load x191 x192 x193 x194)
  thus ?thesis
    using assms
  proof (cases x192; cases ves)
    case None
    case (Cons a list)
    thus ?thesis
      using Load assms None
```

```
      proof (cases a; cases smem-ind s i)
        case (ConstInt32 x1)
        case (Some a)
        thus ?thesis
          using Load assms None Cons ConstInt32
              progress-L0-left[OF reduce.intros(12)]
              progress-L0-left[OF reduce.intros(13)]
              is-const-list-vs-to-es-list[of rev list]
          by (cases load (s.mem s ! a) (nat-of-int x1) x194 (t-length x191) )
             (auto simp add: is-const-def)
      qed auto
    next
      case outer-some:(Some tp-sx)
      case (Cons a list)
      thus ?thesis
        using Load assms outer-some
        proof (cases a; cases smem-ind s i; cases tp-sx)
          case (ConstInt32 x1)
          case (Pair tp sx)
          case (Some a)
          thus ?thesis
            using Load assms outer-some Cons ConstInt32 Pair
                progress-L0-left[OF reduce.intros(14)]
                progress-L0-left[OF reduce.intros(15)]
                is-const-list-vs-to-es-list[of rev list]
           by (cases load-packed sx (s.mem s ! a) (nat-of-int x1) x194 (tp-length tp)
(t-length x191))
               (auto simp add: is-const-def)
        qed auto
    qed auto
  next
    case (Store t tp a off)
    thus ?thesis
      using assms
    proof (cases ves)
      case outer-Cons:(Cons a list)
      thus ?thesis
        using Store assms
      proof (cases list)
        case (Cons a' list')
        thus ?thesis
          using Store outer-Cons assms
        proof (cases a')
          case (ConstInt32 x1)
          thus ?thesis
            using Store outer-Cons Cons assms
          proof (cases (types-agree t a); cases smem-ind s i)
            case True
            case outer-Some:(Some j)
```

```
          show ?thesis
          proof (cases tp)
            case None
            thus ?thesis
              using Store outer-Cons Cons assms True outer-Some ConstInt32
                    progress-L0-left[OF reduce.intros(16)]
                    progress-L0-left[OF reduce.intros(17)]
                    is-const-list-vs-to-es-list[of rev list']
              by (cases store (s.mem s ! j) (nat-of-int x1) off (bits a) (t-length t))
                 auto
          next
            case (Some the-tp)
            thus ?thesis
              using Store outer-Cons Cons assms True outer-Some ConstInt32
                    progress-L0-left[OF reduce.intros(18)]
                    progress-L0-left[OF reduce.intros(19)]
                    is-const-list-vs-to-es-list[of rev list']
                by (cases store-packed (s.mem s ! j) (nat-of-int x1) off (bits a)
(tp-length the-tp))
                 auto
          qed
        qed (cases tp; auto)+
      qed (cases tp; auto)+
    qed (cases tp; auto)
  qed (cases tp; auto)
next
  case Current-memory
  thus ?thesis
    using assms
  proof (cases smem-ind s i)
    case (Some a)
    thus ?thesis
      using assms Current-memory
            progress-L0-left[OF reduce.intros(20)]
            is-const-list-vs-to-es-list[of rev ves]
      by (auto simp add: is-const-def)
  qed auto
next
  case Grow-memory
  thus ?thesis
    using assms
  proof (cases ves)
    case (Cons a list)
    thus ?thesis
      using assms Grow-memory
    proof (cases a; cases smem-ind s i)
      case (ConstInt32 x1)
      case (Some j)
      thus ?thesis
```

using *assms Grow-memory Cons ConstInt32*
progress-L0-left[*OF reduce.intros*(*21*)]
progress-L0-left[*OF reduce.intros*(*22*)]
is-const-list-vs-to-es-list[*of rev list*]
by (*cases mem-grow-impl* (*s.mem s* ! *j*) (*nat-of-int x1*)) (*auto simp add*: *mem-grow-impl-correct is-const-def*)
qed *auto*
qed *auto*
next
case (*EConst x23*)
thus *?thesis*
using *assms*
by *auto*
next
case (*Unop-i x241 x242*)
thus *?thesis*
using *run-step-basic-unop-testop-sound*[*OF assms*]
by *fastforce*
next
case (*Unop-f x251 x252*)
thus *?thesis*
using *run-step-basic-unop-testop-sound*[*OF assms*]
by *fastforce*
next
case (*Binop-i x261 x262*)
thus *?thesis*
using *run-step-basic-binop-relop-sound*[*OF assms*]
by *fastforce*
next
case (*Binop-f x271 x272*)
thus *?thesis*
using *run-step-basic-binop-relop-sound*[*OF assms*]
by *fastforce*
next
case (*Testop x281 x282*)
thus *?thesis*
using *run-step-basic-unop-testop-sound*[*OF assms*]
by *fastforce*
next
case (*Relop-i x291 x292*)
thus *?thesis*
using *run-step-basic-binop-relop-sound*[*OF assms*]
by *fastforce*
next
case (*Relop-f x301 x302*)
thus *?thesis*
using *run-step-basic-binop-relop-sound*[*OF assms*]
by *fastforce*
next

case (*Cvtop t2 cvtop t1 sx*)
thus *?thesis*
using *assms*
proof (*cases ves*)
case (*Cons a list*)
thus *?thesis*
using *assms Cvtop*
proof (*cases cvtop*; *cases types-agree t1 a*)
case *Convert*
case *True*
thus *?thesis*
using *Convert assms Cvtop Cons*
progress-L0-left[*OF reduce.intros*(*1*)[*OF reduce-simple.intros*(*19*)]]
progress-L0-left[*OF reduce.intros*(*1*)[*OF reduce-simple.intros*(*20*)]]
is-const-list-vs-to-es-list[*of rev list*]
by (*cases* (*cvt t2 sx a*)) *auto*
next
case *Reinterpret*
case *True*
thus *?thesis*
using *Reinterpret assms Cvtop Cons*
progress-L0-left[*OF reduce.intros*(*1*)[*OF reduce-simple.intros*(*21*)]]
is-const-list-vs-to-es-list[*of rev list*]
by (*cases sx*) *auto*
qed *auto*
qed (*cases cvtop*; *auto*)
qed
qed

theorem *run-step-sound*:
assumes *run-step d i* $(s,vs,es) = (s', vs', RSNormal\ es')$
shows $(\!|s;vs;es|\!) \rightsquigarrow\text{-}\ i\ (\!|s';vs';es'|\!)$
using *assms*
proof −
fix *ves e*
have (*run-step d i* $(s,vs,es) = (s', vs', RSNormal\ es')) \Longrightarrow$
$(\lambda i\ (s, vs, es).\ (\!|s;vs;es|\!) \rightsquigarrow\text{-}\ i\ (\!|s';vs';es'|\!))\ i\ (s, vs, es)$
and (*run-one-step d i* $(s,vs,ves,e) = (s', vs', RSNormal\ es')) \Longrightarrow$
$(\lambda i\ (s, vs, ves, e).\ (\!|s;vs;(vs\text{-}to\text{-}es\ ves)@[e]|\!) \rightsquigarrow\text{-}\ i\ (\!|s';vs';es'|\!))\ i\ (s, vs, ves,$
$e)$
proof (*induction d i* - **and** *d i* - *arbitrary*: *s′ vs′ es′* **and** *s′ vs′ es′ rule*:
run-step-run-one-step.induct)
case (*1 d i s vs es*)
obtain *ves ses* **where** *ves-def*:*split-vals-e es* = (*ves*, *ses*)
by (*metis surj-pair*)
thus *?case*
proof (*cases ses*)
case *Nil*
thus *?thesis*

using *1(2) ves-def*
by *simp*
next
case (*Cons a list*)
thus *?thesis*
proof (*cases a = Trap*)
case *True*
have *c-ves*:*const-list* ($$* *ves*)
using *is-const-list*[*of - ves*]
by *simp*
have $es' = [Trap] \wedge (list \neq [] \vee ves \neq [])$
using *Cons 1(2) ves-def True*
by (*cases* ($list \neq [] \vee ves \neq []$)) *auto*
thus *?thesis*
using *Cons 1(2) ves-def split-vals-e-conv-app*[*OF ves-def*] *True progress-L0-trap*[*OF c-ves*]
by *auto*
next
case *False*
obtain *os ovs oes* **where** *ros-def*:*run-one-step d i* (*s*, *vs*, (*rev ves*), *a*) = (*os*, *ovs*, *oes*)
by (*metis surj-pair*)
moreover
then obtain *roes* **where** *oes* = *RSNormal roes*
using *1(2) ves-def Cons False*
by (*cases oes*) *auto*
moreover
hence *os* = s' *ovs* = vs' **and** es'-*def*:es' = *roes* @ *list*
using *1(2) ves-def Cons ros-def False*
by (*cases roes* = [*Trap*], *auto simp del*: *run-one-step.simps*)+
ultimately
have *ros-red*:⦇*s*;*vs*;($$* *ves*) @ [*a*]⦈ ⇝- *i* ⦇s';vs';*roes*⦈
using *1(1)*[*OF ves-def*[*symmetric*] *- Cons*] *ros-def False*
by (*simp del*: *run-one-step.simps*)
have ⦇*s*;*vs*;($$* *ves*)@[*a*]@*list*⦈ ⇝- *i* ⦇s';vs';*roes*@*list*⦈
using *progress-L0*[*OF ros-red*, *of* [] *list*]
unfolding *const-list-def*
by *simp*
thus *?thesis*
using es'-*def Cons split-vals-e-conv-app*[*OF ves-def*]
by *simp*
qed
qed
next
case (*2 d i s vs ves e*)
show *?case*
proof (*cases e*)
case (*Basic x1*)
thus *?thesis*

using *run-step-basic-sound 2(3)*
by *simp*
next
case *Trap*
thus *?thesis*
using *2(3)*
by *simp*
next
case (*Callcl cl*)
obtain *t1s t2s* **where** *cl-type cl* = (*t1s* -> *t2s*)
using *tf.exhaust*[*of - thesis*]
by *fastforce*
moreover
obtain *n* **where** *length t1s* = *n*
by *blast*
moreover
obtain *m* **where** *length t2s* = *m*
by *blast*
moreover
note *local-defs* = *calculation*
show *?thesis*
proof (*cases length ves* ≥ *n*)
case *outer-True*:*True*
obtain *ves′ ves′′* **where** *true-defs*:*split-n ves n* = (*ves′*, *ves′′*)
by (*metis surj-pair*)
have *ves′-length*:*length* (*rev ves′*) = *n*
using *split-n-length*[*OF true-defs outer-True*] *inj-basic-econst length-rev map-injective*
by *blast*
show *?thesis*
proof (*cases cl*)
case (*Func-native i′ tf fts fes*)
hence *s′* = *s vs′* = *vs es′* = (*vs-to-es ves′′* @ [*Local* (*length t2s*) *i′* (*rev ves′* @ (*n-zeros fts*)) [$*Block* ([] -> *t2s*) *fes*]])
using *2(3) Callcl local-defs outer-True true-defs*
unfolding *cl-type-def*
by *auto*
moreover
have ⦇*s*;*vs*;(*vs-to-es ves′*)@[*Callcl cl*]⦈ ⇝-*i* ⦇*s*;*vs*;([*Local* (*length t2s*) *i′* (*rev ves′* @ (*n-zeros fts*)) [$*Block* ([] -> *t2s*) *fes*]])⦈
using *reduce.intros(5) local-defs(1,2) Func-native ves′-length*
unfolding *cl-type-def*
by *fastforce*
ultimately
show *?thesis*
using *Callcl progress-L0-left is-const-list*[*of -* (*rev ves′′*)]
unfolding *split-n-conv-app*[*OF true-defs(1)*]
by *auto*
next

```
          case (Func-host x21 x22)
          thus ?thesis
          proof (cases host-apply-impl s (t1s -> t2s) x22 (rev ves'))
            case None
            hence s = s'
                  vs = vs'
                  es' = vs-to-es ves'' @ [Trap]
              using 2(3) Callcl local-defs outer-True true-defs Func-host
              unfolding cl-type-def
              by auto
            thus ?thesis
              using is-const-list[of - (rev ves'')]
                    reduce.intros(7)[OF - - ves'-length local-defs(2)]
                    split-n-conv-app[OF true-defs]
                    progress-L0-left Callcl Func-host local-defs(1)
              unfolding cl-type-def
              by fastforce
          next
            case (Some a)
            show ?thesis
            proof (cases a)
            case (Pair rs rves)
              thus ?thesis
                using 2(3) Callcl local-defs outer-True true-defs Func-host Some
                unfolding cl-type-def
              proof (cases list-all2 types-agree t2s rves)
                case True
                hence rs = s'
                      vs = vs'
                      es' = vs-to-es ves'' @ ($$* rves)
                      using 2(3) Callcl local-defs outer-True true-defs Func-host Pair
Some
                  unfolding cl-type-def
                  by auto
                thus ?thesis
                 using progress-L0-left reduce.intros(6)[OF - - ves'-length local-defs(2)]
Pair
                        Callcl Func-host local-defs(1) True is-const-list[of - (rev ves'')]
                       split-n-conv-app[OF true-defs] host-apply-impl-correct[OF Some]
                  unfolding cl-type-def
                  by fastforce
              qed auto
            qed
          qed
        qed
      next
        case False
        thus ?thesis
          using 2(3) Callcl local-defs
```

unfolding *cl-type-def*
by (*cases cl*) *auto*
qed
next
case (*Label ln les es*)
thus *?thesis*
proof (*cases es-is-trap es*)
case *True*
thus *?thesis*
using *2*(*3*) *is-const-list-vs-to-es-list*
Label progress-L0[*OF reduce.intros*(*1*)[*OF reduce-simple.intros*(*32*)]]
by *fastforce*
next
case *False*
note *outer-outer-false* = *False*
show *?thesis*
proof (*cases* (*const-list es*))
case *True*
thus *?thesis*
using *2*(*3*) *outer-outer-false Label reduce.intros*(*1*)[*OF reduce-simple.intros*(*31*)]
progress-L0[*OF - is-const-list-vs-to-es-list*[*of rev ves*], **where** *?es-c*=[]]
by *fastforce*
next
case *False*
obtain *s″ vs″ es″* **where** *run-step-is*:*run-step d i* (*s*, *vs*, *es*) = (*s″*, *vs″*, *es″*)
by (*metis surj-pair*)
show *?thesis*
proof (*cases es″*)
case *RSCrash*
thus *?thesis*
using *outer-outer-false False run-step-is Label 2*(*3*)
by *auto*
next
case (*RSBreak bn bvs*)
thus *?thesis*
proof (*cases bn*)
case *0*
have *run-step-is-break0*:*run-step d i* (*s*, *vs*, *es*) = (*s″*, *vs″*, *RSBreak 0 bvs*)
using *run-step-is RSBreak 0*
by *simp*
hence *es′-def*:*es′* = ((*vs-to-es* ((*take ln bvs*)@*ves*))@*les*) ∧ *s′* = *s″* ∧ *vs′* = *vs″* ∧ *ln* ≤ *length bvs*
using *outer-outer-false False run-step-is Label 2*(*3*) *RSBreak*
by (*cases ln* ≤ *length bvs*) *auto*
then obtain *n lfilled es-c* **where** *local-eqs*:*s*=*s′ vs*=*vs′ ln* ≤ *length bvs Lfilled-exact n lfilled* ((*vs-to-es bvs*) @ [$*Br n*] @ *es-c*) *es*
using *run-step-break-imp-lfilled*[*OF run-step-is-break0*] *RSBreak es′-def*

by *fastforce*
then obtain *lfilled′* **where** *lfilled-int*:*Lfilled n lfilled′* ((*vs-to-es bvs*) @ [$*Br n*]) *es*
using *lfilled-collapse2*[*OF Lfilled-exact-imp-Lfilled*]
by *fastforce*
obtain *lfilled′′* **where** *Lfilled n lfilled′′* ((*drop* (*length bvs* − *ln*) (*vs-to-es bvs*)) @ [$*Br n*]) *es*
using *lfilled-collapse1*[*OF lfilled-int*] *is-const-list-vs-to-es-list*[*of rev bvs*] *local-eqs*(*3*)
by *fastforce*
hence ⦇[*Label ln les es*]⦈ ⇝ ⦇(*drop* (*length bvs* − *ln*) (*vs-to-es bvs*))@*les*⦈
using *reduce-simple.intros*(*33*) *local-eqs*(*3*) *is-const-list-vs-to-es-list*
unfolding *drop-map*
by *fastforce*
hence *1*:⦇*s*;*vs*;[*Label ln les es*]⦈ ⇝-*i* ⦇*s′*;*vs′*;(*drop* (*length bvs* − *ln*) (*vs-to-es bvs*))@*les*⦈
using *reduce.intros*(*1*) *local-eqs*(*1*,*2*)
by *fastforce*
have ⦇*s*;*vs*;(*vs-to-es ves*)@[*e*]⦈ ⇝-*i* ⦇*s′*;*vs′*;(*vs-to-es ves*)@(*drop* (*length bvs* − *ln*) (*vs-to-es bvs*))@*les*⦈
using *progress-L0*[*OF 1 is-const-list-vs-to-es-list*[*of rev ves*], *of* []] *Label*
by *fastforce*
thus *?thesis*
using *es′-def*
unfolding *drop-map rev-take*[*symmetric*]
by *auto*
next
case (*Suc nat*)
thus *?thesis*
using *outer-outer-false False run-step-is Label 2*(*3*) *RSBreak*
by *auto*
qed
next
case (*RSReturn x3*)
thus *?thesis*
using *outer-outer-false False run-step-is Label 2*(*3*)
by *auto*
next
case (*RSNormal x4*)
hence *es′* = (*vs-to-es ves*)@[*Label ln les x4*] *s′* = *s′′* *vs′* = *vs′′*
using *outer-outer-false False run-step-is Label 2*(*3*) *run-step-is*
by *auto*
moreover
have *Lfilled 1* (*LRec* (*vs-to-es ves*) *ln les* (*LBase* [] [])) []) *es* ((*vs-to-es ves*)@[*Label ln les es*])
using *Lfilled.intros*(*1*)[*of* [] - [] *es*]
Lfilled.intros(*2*)
is-const-list-vs-to-es-list[*of rev ves*]

```
          unfolding const-list-def
          by fastforce
        moreover
         have Lfilled 1 (LRec (vs-to-es ves) ln les (LBase [] [])) []) x4 ((vs-to-es
ves)@[Label ln les x4])
          using Lfilled.intros(1)[of [] - [] x4]
                Lfilled.intros(2)
                is-const-list-vs-to-es-list[of rev ves]
          unfolding const-list-def
          by fastforce
        moreover
        have inner-reduce:(|s;vs;es|) ⇝-i (|s'';vs'';x4|)
          using 2(1)[OF Label outer-outer-false False] run-step-is RSNormal
          by auto
        ultimately
        show ?thesis
          using Label 2(3) outer-outer-false False run-step-is
                reduce.intros(23)[OF inner-reduce]
          by fastforce
      qed
    qed
  qed
next
  case (Local ln j vls es)
  thus ?thesis
  proof (cases es-is-trap es)
    case True
    thus ?thesis
      using 2(3) is-const-list-vs-to-es-list
            Local progress-L0[OF reduce.intros(1)[OF reduce-simple.intros(39)]]
        by fastforce
  next
    case False
    note outer-outer-false = False
    show ?thesis
    proof (cases (const-list es))
      case True
      note outer-true = True
      thus ?thesis
      proof (cases length es = ln)
        case True
        thus ?thesis
         using 2(3) Local outer-true outer-outer-false is-const-list-vs-to-es-list[of
rev ves]
              reduce.intros(1)[OF reduce-simple.intros(38)[OF outer-true True]]
              progress-L0[where ?es-c=[]]
              by fastforce
      next
        case False
```

thus *?thesis*
using *2*(*3*) *Local outer-outer-false outer-true is-const-list-vs-to-es-list*[*of rev ves*]
by *auto*
qed
next
case *False*
show *?thesis*
proof (*cases d*)
case *0*
thus *?thesis*
using *2*(*3*) *Local outer-outer-false False is-const-list-vs-to-es-list*[*of rev ves*]
by *auto*
next
case (*Suc d'*)
obtain s'' *vls' les'* **where** *run-step-is*:*run-step d' j* (*s*, *vls*, *es*) = (s'', *vls'*, *les'*)
by (*metis surj-pair*)
show *?thesis*
proof (*cases les'*)
case *RSCrash*
thus *?thesis*
using *outer-outer-false False run-step-is Local 2*(*3*) *Suc*
by *auto*
next
case (*RSBreak x21 x22*)
thus *?thesis*
using *outer-outer-false False run-step-is Local 2*(*3*) *Suc*
by *auto*
next
case (*RSReturn x3*)
hence *es'-def*:$es' = (vs\text{-}to\text{-}es\ ((take\ ln\ x3)@ves)) \wedge s' = s'' \wedge vs = vs' \wedge ln \leq length\ x3$
using *outer-outer-false False run-step-is Local 2*(*3*) *Suc*
by (*cases* $ln \leq length\ x3$) *auto*
then obtain *n lfilled es-c* **where** *local-eqs*:$s=s'$ $vs=vs'$ $ln \leq length\ x3$ *Lfilled-exact n lfilled* ((*vs-to-es x3*) @ [$*Return*] @ *es-c*) *es*
using *run-step-is run-step-return-imp-lfilled RSReturn*
by *fastforce*
then obtain *lfilled'* **where** *lfilled-int*:*Lfilled n lfilled'* ((*vs-to-es x3*) @ [$*Return*]) *es*
using *lfilled-collapse2*[*OF Lfilled-exact-imp-Lfilled*]
by *fastforce*
obtain *lfilled''* **where** *Lfilled n lfilled''* ((*drop* (*length x3* − *ln*) (*vs-to-es x3*)) @ [$*Return*]) *es*
using *lfilled-collapse1*[*OF lfilled-int*] *is-const-list-vs-to-es-list*[*of rev x3*] *local-eqs*(*3*)
by *fastforce*

hence ⦇[*Local ln j vls es*]⦈ ⇝ ⦇(*drop* (*length x3* − *ln*) (*vs-to-es x3*))⦈
using *reduce-simple.intros*(*40*) *local-eqs*(*3*) *is-const-list-vs-to-es-list*
unfolding *drop-map*
by *fastforce*
hence *1*:⦇*s*;*vs*;[*Local ln j vls es*]⦈ ⇝-*i* ⦇*s*′;*vs*′;(*drop* (*length x3* − *ln*) (*vs-to-es x3*))⦈
using *reduce.intros*(*1*) *local-eqs*(*1*,*2*)
by *fastforce*
have ⦇*s*;*vs*;(*vs-to-es ves*)@[*e*]⦈ ⇝-*i* ⦇*s*′;*vs*′;(*vs-to-es ves*)@(*drop* (*length x3* − *ln*) (*vs-to-es x3*))⦈
using *progress-L0*[*OF 1 is-const-list-vs-to-es-list*[*of rev ves*], *of* []] *Local*
by *fastforce*
thus *?thesis*
using *es*′*-def*
unfolding *drop-map rev-take*[*symmetric*]
by *auto*
next
case (*RSNormal x4*)
hence *inner-reduce*:⦇*s*;*vls*;*es*⦈ ⇝-*j* ⦇*s*′′;*vls*′;*x4*⦈
using *2*(*2*)[*OF Local outer-outer-false False*] *run-step-is Suc*
by *auto*
thus *?thesis*
using *Local 2*(*3*) *Local outer-outer-false False run-step-is Suc*
reduce.intros(*24*)[*OF inner-reduce*] *RSNormal*
progress-L0-left is-const-list-vs-to-es-list[*of rev ves*]
by (*auto simp del*: *run-step.simps*)
qed
qed
qed
qed
qed
qed
thus *?thesis*
using *assms*
by *blast*
qed

end

References

[1] A. Haas, A. Rossberg, D. L. Schuff, B. L. Titzer, M. Holman, D. Gohman, L. Wagner, A. Zakai, and J. Bastien. Bringing the web up to speed with webassembly. In *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI 2017, pages 185–200, New York, NY, USA, 2017. ACM.

[2] 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.