

# JinjaDCI: a Java semantics with dynamic class initialization

Susannah Mansky

March 17, 2025

**Abstract.** This work is an extension of the Jinja semantics for Java and the JVM by Klein and Nipkow to include static fields and methods and dynamic class initialization. In Java, class initialization methods are run dynamically, called when classes are first used. Such calls are handled by the running of an initialization procedure, which interrupts execution and determines which initialization methods must be run before execution continues. This interrupting is modeled here in a couple of ways. In the Java semantics, evaluation is performed via expressions that are manipulated through evaluation until a final value is reached. In JinjaDCI, we have added two types of initialization expressions whose evaluations produce the steps of the initialization procedure. These expressions can occur during evaluation and store the calling expression away to continue being evaluated once the procedure is complete. In the JVM semantics, since programs are static sequences of instructions, the initialization procedure is run instead by the execution function. This function performs steps of the procedure rather than calling instructions when the initialization procedure has been called.

This extension includes the necessary updates to all major proofs from the original Jinja, including type safety and correctness of compilation from the Java semantics to the JVM semantics.

This work is partially described in [1].

# Contents

<b>1</b>	<b>Jinja Source Language</b>	<b>5</b>
1.1	Auxiliary Definitions	5
1.2	Jinja types	7
1.3	Class Declarations and Programs	8
1.4	Relations between Jinja Types	9
1.5	Jinja Values	15
1.6	Objects and the Heap	15
1.7	Exceptions	18
1.8	Expressions	21
1.9	Well-typedness of Jinja expressions	27
1.10	Runtime Well-typedness	30
1.11	Program State	33
1.12	System Classes	33
1.13	Generic Well-formedness of programs	34
1.14	Weak well-formedness of Jinja programs	38
1.15	Big Step Semantics	39
1.16	Definite assignment	47
1.17	Conformance Relations for Type Soundness Proofs	49
1.18	Small Step Semantics	51
1.19	Expression conformance properties	60
1.20	Progress of Small Step Semantics	65
1.21	Well-formedness Constraints	67
1.22	Type Safety Proof	67
1.23	Equivalence of Big Step and Small Step Semantics	70
1.24	Program annotation	83
<b>2</b>	<b>Jinja Virtual Machine</b>	<b>85</b>
2.1	State of the JVM	85
2.2	Instructions of the JVM	86
2.3	Exception handling in the JVM	87
2.4	Program Execution in the JVM	89
2.5	Program Execution in the JVM in full small step style	93
2.6	A Defensive JVM	95
2.7	The Jinja Type System as a Semilattice	99
2.8	The JVM Type System as Semilattice	101
2.9	Effect of Instructions on the State Type	102
2.10	Monotonicity of $\text{eff}$ and $\text{app}$	109

2.11	The Bytecode Verifier . . . . .	110
2.12	The Typing Framework for the JVM . . . . .	111
2.13	Kildall for the JVM . . . . .	113
2.14	LBV for the JVM . . . . .	115
2.15	BV Type Safety Invariant . . . . .	116
2.16	Property preservation under <i>class-add</i> . . . . .	122
2.17	Properties and types of the starting program . . . . .	128
2.18	BV Type Safety Proof . . . . .	131
2.19	Welltyped Programs produce no Type Errors . . . . .	135
<b>3</b>	<b>Compilation</b> . . . . .	<b>137</b>
3.1	An Intermediate Language . . . . .	137
3.2	Well-Formedness of Intermediate Language . . . . .	145
3.3	Program Compilation . . . . .	151
3.4	Compilation Stage 1 . . . . .	154
3.5	Correctness of Stage 1 . . . . .	155
3.6	Compilation Stage 2 . . . . .	157
3.7	Correctness of Stage 2 . . . . .	160
3.8	Combining Stages 1 and 2 . . . . .	174
3.9	Preservation of Well-Typedness . . . . .	174

# Chapter 1

## Jinja Source Language

### 1.1 Auxiliary Definitions

**theory** *Auxiliary* imports *Main* begin

**lemma** *nat-add-max-le[simp]*:  
 $((n::nat) + \max i j \leq m) = (n + i \leq m \wedge n + j \leq m)$   
 $\langle proof \rangle$

**lemma** *Suc-add-max-le[simp]*:  
 $(Suc(n + \max i j) \leq m) = (Suc(n + i) \leq m \wedge Suc(n + j) \leq m) \langle proof \rangle$

**notation** *Some* ( $\langle \langle [-] \rangle \rangle$ )

#### 1.1.1 *distinct-fst*

**definition** *distinct-fst* ::  $('a \times 'b) \text{ list} \Rightarrow \text{bool}$

**where**

$\text{distinct-fst} \equiv \text{distinct} \circ \text{map fst}$

**lemma** *distinct-fst-Nil [simp]*:  
 $\text{distinct-fst } []$   
 $\langle proof \rangle$

**lemma** *distinct-fst-Cons [simp]*:  
 $\text{distinct-fst } ((k,x)\#kxs) = (\text{distinct-fst } kxs \wedge (\forall y. (k,y) \notin \text{set } kxs)) \langle proof \rangle$

**lemma** *distinct-fst-appendD*:  
 $\text{distinct-fst}(kxs @ kxs') \Longrightarrow \text{distinct-fst } kxs \wedge \text{distinct-fst } kxs' \langle proof \rangle$

**lemma** *map-of-SomeI*:  
 $\llbracket \text{distinct-fst } kxs; (k,x) \in \text{set } kxs \rrbracket \Longrightarrow \text{map-of } kxs k = \text{Some } x \langle proof \rangle$

#### 1.1.2 Using *list-all2* for relations

**definition** *fun-of* ::  $('a \times 'b) \text{ set} \Rightarrow 'a \Rightarrow 'b \Rightarrow \text{bool}$

**where**

$\text{fun-of } S \equiv \lambda x y. (x,y) \in S$

Convenience lemmas

**lemma** *rel-list-all2-Cons [iff]*:  
 $\text{list-all2 } (\text{fun-of } S) (x\#xs) (y\#ys) =$   
 $((x,y) \in S \wedge \text{list-all2 } (\text{fun-of } S) xs ys)$

*<proof>*

**lemma** *rel-list-all2-Cons1:*

$list\text{-}all2\ (fun\text{-}of\ S)\ (x\#\!xs)\ ys =$   
 $(\exists z\ zs.\ ys = z\#\!zs \wedge (x,z) \in S \wedge list\text{-}all2\ (fun\text{-}of\ S)\ xs\ zs)$   
*<proof>*

**lemma** *rel-list-all2-Cons2:*

$list\text{-}all2\ (fun\text{-}of\ S)\ xs\ (y\#\!ys) =$   
 $(\exists z\ zs.\ xs = z\#\!zs \wedge (z,y) \in S \wedge list\text{-}all2\ (fun\text{-}of\ S)\ zs\ ys)$   
*<proof>*

**lemma** *rel-list-all2-refl:*

$(\bigwedge x.\ (x,x) \in S) \implies list\text{-}all2\ (fun\text{-}of\ S)\ xs\ xs$   
*<proof>*

**lemma** *rel-list-all2-antisym:*

$\llbracket (\bigwedge x\ y.\ \llbracket (x,y) \in S; (y,x) \in T \rrbracket \implies x = y);$   
 $list\text{-}all2\ (fun\text{-}of\ S)\ xs\ ys; list\text{-}all2\ (fun\text{-}of\ T)\ ys\ xs \rrbracket \implies xs = ys$   
*<proof>*

**lemma** *rel-list-all2-trans:*

$\llbracket \bigwedge a\ b\ c.\ \llbracket (a,b) \in R; (b,c) \in S \rrbracket \implies (a,c) \in T;$   
 $list\text{-}all2\ (fun\text{-}of\ R)\ as\ bs; list\text{-}all2\ (fun\text{-}of\ S)\ bs\ cs \rrbracket$   
 $\implies list\text{-}all2\ (fun\text{-}of\ T)\ as\ cs$   
*<proof>*

**lemma** *rel-list-all2-update-cong:*

$\llbracket i < size\ xs; list\text{-}all2\ (fun\text{-}of\ S)\ xs\ ys; (x,y) \in S \rrbracket$   
 $\implies list\text{-}all2\ (fun\text{-}of\ S)\ (xs[i:=x])\ (ys[i:=y])$   
*<proof>*

**lemma** *rel-list-all2-nthD:*

$\llbracket list\text{-}all2\ (fun\text{-}of\ S)\ xs\ ys; p < size\ xs \rrbracket \implies (xs!p, ys!p) \in S$   
*<proof>*

**lemma** *rel-list-all2I:*

$\llbracket length\ a = length\ b; \bigwedge n.\ n < length\ a \implies (a!n, b!n) \in S \rrbracket \implies list\text{-}all2\ (fun\text{-}of\ S)\ a\ b$   
*<proof>*

### 1.1.3 Auxiliary properties of *map-of* function

**lemma** *map-of-set-pcs-notin:*  $C \notin (\lambda t.\ snd\ (fst\ t))\ \text{'set FDTs'} \implies map\text{-}of\ FDTs\ (F, C) = None$ *<proof>*

**lemma** *map-of-insertmap-SomeD':*

$map\text{-}of\ fs\ F = Some\ y \implies map\text{-}of\ (map\ (\lambda(F, y).\ (F, D, y))\ fs)\ F = Some(D,y)$ *<proof>*

**lemma** *map-of-reinsert-neq-None:*

$Ca \neq D \implies map\text{-}of\ (map\ (\lambda(F, y).\ ((F, Ca), y))\ fs)\ (F, D) = None$ *<proof>*

**lemma** *map-of-remap-insertmap:*

$map\text{-}of\ (map\ ((\lambda((F, D), b, T).\ (F, D, b, T)) \circ (\lambda(F, y).\ ((F, D), y))))\ fs$   
 $= map\text{-}of\ (map\ (\lambda(F, y).\ (F, D, y))\ fs)$ *<proof>*

**lemma** *map-of-reinsert-SomeD:*

$map\text{-}of\ (map\ (\lambda(F, y).\ ((F, D), y))\ fs)\ (F, D) = Some\ T \implies map\text{-}of\ fs\ F = Some\ T$ *<proof>*

**lemma** *map-of-filtered-SomeD:*

$map\text{-}of\ fs\ (F,D) = Some\ (a, T) \implies Q\ ((F,D),a,T) \implies$   
 $map\text{-}of\ (map\ (\lambda((F,D), b, T).\ ((F,D), P\ T))\ (filter\ Q\ fs))$   
 $(F,D) = Some\ (P\ T)$ *<proof>*

**lemma** *map-of-remove-filtered-SomeD:*

$map\text{-}of\ fs\ (F,C) = Some\ (a, T) \implies Q\ ((F,C),a,T) \implies$   
 $map\text{-}of\ (map\ (\lambda((F,D), b, T).\ (F, P\ T))\ [(F, D), b, T] \leftarrow fs.\ Q\ ((F, D), b, T) \wedge D = C])$   
 $F = Some\ (P\ T)$ *<proof>*

**lemma** *map-of-Some-None-split*:

**assumes**  $t = \text{map } (\lambda(F, y). ((F, C), y)) \text{ fs} @ t'$  *map-of*  $t' (F, C) = \text{None}$  *map-of*  $t (F, C) = \text{Some } y$   
**shows** *map-of*  $(\text{map } (\lambda((F, D), b, T). (F, D, b, T)) t) F = \text{Some } (C, y)$  *<proof>*  
**end**

## 1.2 Jinja types

**theory** *Type* **imports** *Auxiliary* **begin**

**type-synonym** *cname* = *string* — class names

**type-synonym** *mname* = *string* — method name

**type-synonym** *vname* = *string* — names for local/field variables

**definition** *Object* :: *cname*

**where**

*Object*  $\equiv$  "*Object*"

**definition** *this* :: *vname*

**where**

*this*  $\equiv$  "*this*"

**definition** *clinit* :: *string* **where** *clinit* = "<*clinit*>"

**definition** *init* :: *string* **where** *init* = "<*init*>"

**definition** *start-m* :: *string* **where** *start-m* = "<*start*>"

**definition** *Start* :: *string* **where** *Start* = "<*Start*>"

**lemma** *start-m-neq-clinit* [*simp*]: *start-m*  $\neq$  *clinit* *<proof>*

**lemma** *Object-neq-Start* [*simp*]: *Object*  $\neq$  *Start* *<proof>*

**lemma** *Start-neq-Object* [*simp*]: *Start*  $\neq$  *Object* *<proof>*

**datatype** *staticb* = *Static* | *NonStatic*

— types

**datatype** *ty*

= *Void* — type of statements

| *Boolean*

| *Integer*

| *NT* — null type

| *Class cname* — class type

**definition** *is-refT* :: *ty*  $\Rightarrow$  *bool*

**where**

*is-refT T*  $\equiv T = \text{NT} \vee (\exists C. T = \text{Class } C)$

**lemma** [*iff*]: *is-refT NT* *<proof>*

**lemma** [*iff*]: *is-refT (Class C)* *<proof>*

**lemma** *refTE*:

$\llbracket \text{is-refT } T; T = \text{NT} \implies P; \bigwedge C. T = \text{Class } C \implies P \rrbracket \implies P$  *<proof>*

**lemma** *not-refTE*:

$\llbracket \neg \text{is-refT } T; T = \text{Void} \vee T = \text{Boolean} \vee T = \text{Integer} \implies P \rrbracket \implies P$  *<proof>*

**end**

### 1.3 Class Declarations and Programs

**theory** *Decl* **imports** *Type* **begin**

**type-synonym**

$fdecl = vname \times staticb \times ty$  — field declaration

**type-synonym**

$'m\ mdecl = mname \times staticb \times ty\ list \times ty \times 'm$  — method = name, static flag, arg. types, return type, body

**type-synonym**

$'m\ class = cname \times fdecl\ list \times 'm\ mdecl\ list$  — class = superclass, fields, methods

**type-synonym**

$'m\ cdecl = cname \times 'm\ class$  — class declaration

**type-synonym**

$'m\ prog = 'm\ cdecl\ list$  — program

**definition**  $class :: 'm\ prog \Rightarrow cname \rightarrow 'm\ class$

**where**

$class \equiv map-of$

**lemma** *class-cons*:  $\llbracket C \neq fst\ x \rrbracket \Longrightarrow class\ (x \# P)\ C = class\ P\ C$   
 $\langle proof \rangle$

**definition**  $is-class :: 'm\ prog \Rightarrow cname \Rightarrow bool$

**where**

$is-class\ P\ C \equiv class\ P\ C \neq None$

**lemma** *finite-is-class*:  $finite\ \{C.\ is-class\ P\ C\}$  $\langle proof \rangle$

**definition**  $is-type :: 'm\ prog \Rightarrow ty \Rightarrow bool$

**where**

$is-type\ P\ T \equiv$   
 $(case\ T\ of\ Void \Rightarrow True \mid Boolean \Rightarrow True \mid Integer \Rightarrow True \mid NT \Rightarrow True$   
 $\mid Class\ C \Rightarrow is-class\ P\ C)$

**lemma** *is-type-simps* [*simp*]:

$is-type\ P\ Void \wedge is-type\ P\ Boolean \wedge is-type\ P\ Integer \wedge$   
 $is-type\ P\ NT \wedge is-type\ P\ (Class\ C) = is-class\ P\ C \langle proof \rangle$

**abbreviation**

$types\ P == Collect\ (is-type\ P)$

**lemma** *class-exists-equiv*:

$(\exists x.\ fst\ x = cn \wedge x \in set\ P) = (class\ P\ cn \neq None)$   
 $\langle proof \rangle$

**lemma** *class-exists-equiv2*:

$(\exists x.\ fst\ x = cn \wedge x \in set\ (P1\ @\ P2)) = (class\ P1\ cn \neq None \vee class\ P2\ cn \neq None)$   
 $\langle proof \rangle$

**end**



## 1.4 Relations between Jinja Types

**theory** *TypeRel* **imports**

*HOL-Library.Transitive-Closure-Table*

*Decl*

**begin**

### 1.4.1 The subclass relations

**inductive-set**

*subcls1* :: 'm prog ⇒ (cname × cname) set

**and** *subcls1'* :: 'm prog ⇒ [cname, cname] ⇒ bool (⟦- ⊢ - <<sup>1</sup> -> [71,71,71] 70)

**for** *P* :: 'm prog

**where**

$P \vdash C \prec^1 D \equiv (C, D) \in \text{subcls1 } P$

| *subcls1I*:  $\llbracket \text{class } P \ C = \text{Some } (D, \text{rest}); C \neq \text{Object} \rrbracket \implies P \vdash C \prec^1 D$

**abbreviation**

*subcls* :: 'm prog ⇒ [cname, cname] ⇒ bool (⟦- ⊢ - ≼\* -> [71,71,71] 70)

**where**  $P \vdash C \preceq^* D \equiv (C, D) \in (\text{subcls1 } P)^*$

**lemma** *subcls1D*:  $P \vdash C \prec^1 D \implies C \neq \text{Object} \wedge (\exists fs \ ms. \text{class } P \ C = \text{Some } (D, fs, ms)) \langle \text{proof} \rangle$

**lemma** [*iff*]:  $\neg P \vdash \text{Object} \prec^1 C \langle \text{proof} \rangle$

**lemma** [*iff*]:  $(P \vdash \text{Object} \preceq^* C) = (C = \text{Object}) \langle \text{proof} \rangle$

**lemma** *subcls1-def2*:

*subcls1* *P* =

$(\text{SIGMA } C: \{C. \text{is-class } P \ C\}. \{D. C \neq \text{Object} \wedge \text{fst } (\text{the } (\text{class } P \ C)) = D\}) \langle \text{proof} \rangle$

**lemma** *finite-subcls1*: *finite* (*subcls1* *P*) ⟨proof⟩

**primrec** *supercls-lst* :: 'm prog ⇒ cname list ⇒ bool **where**

*supercls-lst* *P* (*C* # *Cs*) =  $(\forall C' \in \text{set } Cs. P \vdash C' \preceq^* C) \wedge \text{supercls-lst } P \ Cs$  |

*supercls-lst* *P* [] = True

**lemma** *supercls-lst-app*:

$\llbracket \text{supercls-lst } P \ (C \# Cs); P \vdash C \preceq^* C' \rrbracket \implies \text{supercls-lst } P \ (C' \# C \# Cs)$

⟨proof⟩

### 1.4.2 The subtype relations

**inductive**

*widen* :: 'm prog ⇒ ty ⇒ ty ⇒ bool (⟦- ⊢ - ≤ -> [71,71,71] 70)

**for** *P* :: 'm prog

**where**

*widen-refl*[*iff*]:  $P \vdash T \leq T$

| *widen-subcls*:  $P \vdash C \preceq^* D \implies P \vdash \text{Class } C \leq \text{Class } D$

| *widen-null*[*iff*]:  $P \vdash \text{NT} \leq \text{Class } C$

**abbreviation**

*widens* :: 'm prog ⇒ ty list ⇒ ty list ⇒ bool

(⟦- ⊢ - ≤> [71,71,71] 70) **where**

*widens* *P* *Ts* *Ts'* ≡ *list-all2* (*widen* *P*) *Ts* *Ts'*

**lemma** [*iff*]:  $(P \vdash T \leq \text{Void}) = (T = \text{Void}) \langle \text{proof} \rangle$

**lemma** [*iff*]:  $(P \vdash T \leq \text{Boolean}) = (T = \text{Boolean}) \langle \text{proof} \rangle$

**lemma** [*iff*]:  $(P \vdash T \leq \text{Integer}) = (T = \text{Integer}) \langle \text{proof} \rangle$

**lemma** [*iff*]:  $(P \vdash \text{Void} \leq T) = (T = \text{Void}) \langle \text{proof} \rangle$

**lemma** [iff]:  $(P \vdash \text{Boolean} \leq T) = (T = \text{Boolean})\langle \text{proof} \rangle$

**lemma** [iff]:  $(P \vdash \text{Integer} \leq T) = (T = \text{Integer})\langle \text{proof} \rangle$

**lemma** *Class-widen*:  $P \vdash \text{Class } C \leq T \implies \exists D. T = \text{Class } D\langle \text{proof} \rangle$

**lemma** [iff]:  $(P \vdash T \leq NT) = (T = NT)\langle \text{proof} \rangle$

**lemma** *Class-widen-Class* [iff]:  $(P \vdash \text{Class } C \leq \text{Class } D) = (P \vdash C \preceq^* D)\langle \text{proof} \rangle$

**lemma** *widen-Class*:  $(P \vdash T \leq \text{Class } C) = (T = NT \vee (\exists D. T = \text{Class } D \wedge P \vdash D \preceq^* C))\langle \text{proof} \rangle$

**lemma** *widen-trans*[trans]:  $\llbracket P \vdash S \leq U; P \vdash U \leq T \rrbracket \implies P \vdash S \leq T\langle \text{proof} \rangle$

**lemma** *widens-trans* [trans]:  $\llbracket P \vdash Ss \llbracket \leq \rrbracket Ts; P \vdash Ts \llbracket \leq \rrbracket Us \rrbracket \implies P \vdash Ss \llbracket \leq \rrbracket Us\langle \text{proof} \rangle$

### 1.4.3 Method lookup

**inductive**

*Methods* :: [*m prog*, *cname*, *mname*  $\rightarrow$  (*staticb*  $\times$  *ty list*  $\times$  *ty*  $\times$  *m*)  $\times$  *cname*]  $\Rightarrow$  *bool*  
 ( $\vdash \vdash$  - *sees'-methods*  $\rightarrow$  [51,51,51] 50)

**for** *P* :: *m prog*

**where**

*sees-methods-Object*:

$\llbracket \text{class } P \text{ Object} = \text{Some}(D,fs,ms); Mm = \text{map-option } (\lambda m. (m, \text{Object})) \circ \text{map-of } ms \rrbracket$   
 $\implies P \vdash \text{Object sees-methods } Mm$

| *sees-methods-rec*:

$\llbracket \text{class } P \text{ C} = \text{Some}(D,fs,ms); C \neq \text{Object}; P \vdash D \text{ sees-methods } Mm;$   
 $Mm' = Mm ++ (\text{map-option } (\lambda m. (m, C)) \circ \text{map-of } ms) \rrbracket$   
 $\implies P \vdash C \text{ sees-methods } Mm'$

**lemma** *sees-methods-fun*:

**assumes** 1:  $P \vdash C \text{ sees-methods } Mm$

**shows**  $\bigwedge Mm'. P \vdash C \text{ sees-methods } Mm' \implies Mm' = Mm$   
 $\langle \text{proof} \rangle$

**lemma** *visible-methods-exist*:

$P \vdash C \text{ sees-methods } Mm \implies Mm \text{ M} = \text{Some}(m, D) \implies$   
 $(\exists D' fs ms. \text{class } P \text{ D} = \text{Some}(D', fs, ms) \wedge \text{map-of } ms \text{ M} = \text{Some } m)$   
 $\langle \text{proof} \rangle$

**lemma** *sees-methods-decl-above*:

**assumes** *Csees*:  $P \vdash C \text{ sees-methods } Mm$

**shows**  $Mm \text{ M} = \text{Some}(m, D) \implies P \vdash C \preceq^* D$   
 $\langle \text{proof} \rangle$

**lemma** *sees-methods-idemp*:

**assumes** *Cmethods*:  $P \vdash C \text{ sees-methods } Mm$

**shows**  $\bigwedge m D. Mm \text{ M} = \text{Some}(m, D) \implies$   
 $\exists Mm'. (P \vdash D \text{ sees-methods } Mm') \wedge Mm' \text{ M} = \text{Some}(m, D)\langle \text{proof} \rangle$

**lemma** *sees-methods-decl-mono*:

**assumes** *sub*:  $P \vdash C' \preceq^* C$

**shows**  $P \vdash C \text{ sees-methods } Mm \implies$   
 $\exists Mm' Mm_2. P \vdash C' \text{ sees-methods } Mm' \wedge Mm' = Mm ++ Mm_2 \wedge$   
 $(\forall M m D. Mm_2 \text{ M} = \text{Some}(m, D) \longrightarrow P \vdash D \preceq^* C)\langle \text{proof} \rangle$

**lemma** *sees-methods-is-class-Object*:

$P \vdash D \text{ sees-methods } Mm \implies \text{is-class } P \text{ Object}$   
 $\langle \text{proof} \rangle$

**lemma** *sees-methods-sub-Obj*:  $P \vdash C \text{ sees-methods } Mm \implies P \vdash C \preceq^* \text{Object}$

$\langle \text{proof} \rangle$

**definition** *Method* :: 'm prog  $\Rightarrow$  cname  $\Rightarrow$  mname  $\Rightarrow$  staticb  $\Rightarrow$  ty list  $\Rightarrow$  ty  $\Rightarrow$  'm  $\Rightarrow$  cname  $\Rightarrow$  bool  
 ( $\langle \vdash - \text{sees} -, - : - \rightarrow - = - \text{ in } \rightarrow [51,51,51,51,51,51,51,51] 50 \rangle$ )

**where**

$P \vdash C \text{ sees } M, b: Ts \rightarrow T = m \text{ in } D \equiv$   
 $\exists Mm. P \vdash C \text{ sees-methods } Mm \wedge Mm M = \text{Some}((b, Ts, T, m), D)$

**definition** *has-method* :: 'm prog  $\Rightarrow$  cname  $\Rightarrow$  mname  $\Rightarrow$  staticb  $\Rightarrow$  bool  
 ( $\langle \vdash - \text{has} -, \rightarrow [51,0,0,51] 50 \rangle$ )

**where**

$P \vdash C \text{ has } M, b \equiv \exists Ts T m D. P \vdash C \text{ sees } M, b: Ts \rightarrow T = m \text{ in } D$

**lemma** *sees-method-fun*:

$\llbracket P \vdash C \text{ sees } M, b: TS \rightarrow T = m \text{ in } D; P \vdash C \text{ sees } M, b': TS' \rightarrow T' = m' \text{ in } D' \rrbracket$   
 $\implies b = b' \wedge TS' = TS \wedge T' = T \wedge m' = m \wedge D' = D$

$\langle \text{proof} \rangle$

**lemma** *sees-method-decl-above*:

$P \vdash C \text{ sees } M, b: Ts \rightarrow T = m \text{ in } D \implies P \vdash C \preceq^* D$

$\langle \text{proof} \rangle$

**lemma** *visible-method-exists*:

$P \vdash C \text{ sees } M, b: Ts \rightarrow T = m \text{ in } D \implies$   
 $\exists D' fs ms. \text{class } P D = \text{Some}(D', fs, ms) \wedge \text{map-of } ms M = \text{Some}(b, Ts, T, m) \langle \text{proof} \rangle$

**lemma** *sees-method-idemp*:

$P \vdash C \text{ sees } M, b: Ts \rightarrow T = m \text{ in } D \implies P \vdash D \text{ sees } M, b: Ts \rightarrow T = m \text{ in } D$

$\langle \text{proof} \rangle$

**lemma** *sees-method-decl-mono*:

**assumes** *sub*:  $P \vdash C' \preceq^* C$  **and**

*C-sees*:  $P \vdash C \text{ sees } M, b: Ts \rightarrow T = m \text{ in } D$  **and**

*C'-sees*:  $P \vdash C' \text{ sees } M, b': Ts' \rightarrow T' = m' \text{ in } D'$

**shows**  $P \vdash D' \preceq^* D$

$\langle \text{proof} \rangle$

**lemma** *sees-methods-is-class*:  $P \vdash C \text{ sees-methods } Mm \implies \text{is-class } P C \langle \text{proof} \rangle$

**lemma** *sees-method-is-class*:

$\llbracket P \vdash C \text{ sees } M, b: Ts \rightarrow T = m \text{ in } D \rrbracket \implies \text{is-class } P C \langle \text{proof} \rangle$

**lemma** *sees-method-is-class'*:

$\llbracket P \vdash C \text{ sees } M, b: Ts \rightarrow T = m \text{ in } D \rrbracket \implies \text{is-class } P D \langle \text{proof} \rangle$

**lemma** *sees-method-sub-Obj*:  $P \vdash C \text{ sees } M, b: Ts \rightarrow T = m \text{ in } D \implies P \vdash C \preceq^* \text{Object}$

$\langle \text{proof} \rangle$

#### 1.4.4 Field lookup

**inductive**

*Fields* :: ['m prog, cname, ((vname  $\times$  cname)  $\times$  staticb  $\times$  ty) list]  $\Rightarrow$  bool  
 ( $\langle \vdash - \text{has}'\text{-fields} \rightarrow [51,51,51] 50 \rangle$ )

**for**  $P :: 'm \text{ prog}$

**where**

*has-fields-rec*:

$\llbracket \text{class } P C = \text{Some}(D, fs, ms); C \neq \text{Object}; P \vdash D \text{ has-fields } FDTs;$   
 $FDTs' = \text{map}(\lambda(F, b, T). ((F, C), b, T)) fs @ FDTs \rrbracket$   
 $\implies P \vdash C \text{ has-fields } FDTs'$

| *has-fields-Object*:

$\llbracket \text{class } P \text{ Object} = \text{Some}(D, fs, ms); \text{FDTs} = \text{map } (\lambda(F, b, T). ((F, \text{Object}), b, T)) \text{ fs} \rrbracket$   
 $\implies P \vdash \text{Object has-fields FDTs}$

**lemma** *has-fields-is-class*:

$P \vdash C \text{ has-fields FDTs} \implies \text{is-class } P \ C \langle \text{proof} \rangle$

**lemma** *has-fields-fun*:

**assumes** 1:  $P \vdash C \text{ has-fields FDTs}$

**shows**  $\bigwedge \text{FDTs}' . P \vdash C \text{ has-fields FDTs}' \implies \text{FDTs}' = \text{FDTs}$   
 $\langle \text{proof} \rangle$

**lemma** *all-fields-in-has-fields*:

**assumes** *sub*:  $P \vdash C \text{ has-fields FDTs}$

**shows**  $\llbracket P \vdash C \preceq^* D; \text{class } P \ D = \text{Some}(D', fs, ms); (F, b, T) \in \text{set } fs \rrbracket$   
 $\implies ((F, D), b, T) \in \text{set } \text{FDTs} \langle \text{proof} \rangle$

**lemma** *has-fields-decl-above*:

**assumes** *fields*:  $P \vdash C \text{ has-fields FDTs}$

**shows**  $((F, D), b, T) \in \text{set } \text{FDTs} \implies P \vdash C \preceq^* D \langle \text{proof} \rangle$

**lemma** *subcls-notin-has-fields*:

**assumes** *fields*:  $P \vdash C \text{ has-fields FDTs}$

**shows**  $((F, D), b, T) \in \text{set } \text{FDTs} \implies (D, C) \notin (\text{subcls1 } P)^+ \langle \text{proof} \rangle$

**lemma** *subcls-notin-has-fields2*:

**assumes** *fields*:  $P \vdash C \text{ has-fields FDTs}$

**shows**  $\llbracket C \neq \text{Object}; P \vdash C \prec^1 D \rrbracket \implies (D, C) \notin (\text{subcls1 } P)^*$   
 $\langle \text{proof} \rangle$

**lemma** *has-fields-mono-lem*:

**assumes** *sub*:  $P \vdash D \preceq^* C$

**shows**  $P \vdash C \text{ has-fields FDTs}$

$\implies \exists \text{pre} . P \vdash D \text{ has-fields } \text{pre} @ \text{FDTs} \wedge \text{dom}(\text{map-of } \text{pre}) \cap \text{dom}(\text{map-of } \text{FDTs}) = \{\} \langle \text{proof} \rangle$

**lemma** *has-fields-declaring-classes*:

**shows**  $P \vdash C \text{ has-fields FDTs}$

$\implies \exists \text{pre } \text{FDTs}' . \text{FDTs} = \text{pre} @ \text{FDTs}'$

$\wedge (C \neq \text{Object} \longrightarrow (\exists D \text{ fs } ms . \text{class } P \ C = \llbracket (D, fs, ms) \rrbracket \wedge P \vdash D \text{ has-fields FDTs}'))$

$\wedge \text{set}(\text{map } (\lambda t . \text{snd}(\text{fst } t)) \text{ pre}) \subseteq \{C\}$

$\wedge \text{set}(\text{map } (\lambda t . \text{snd}(\text{fst } t)) \text{FDTs}') \subseteq \{C' . C' \neq C \wedge P \vdash C \preceq^* C'\}$

$\langle \text{proof} \rangle$

**lemma** *has-fields-mono-lem2*:

**assumes** *hf*:  $P \vdash C \text{ has-fields FDTs}$

**and** *cls*:  $\text{class } P \ C = \text{Some}(D, fs, ms)$  **and** *map-of*:  $\text{map-of } \text{FDTs} (F, C) = \llbracket (b, T) \rrbracket$

**shows**  $\exists \text{FDTs}' . \text{FDTs} = (\text{map } (\lambda(F, b, T). ((F, C), b, T)) \text{ fs}) @ \text{FDTs}' \wedge \text{map-of } \text{FDTs}' (F, C) = \text{None}$   
 $\langle \text{proof} \rangle$

**lemma** *has-fields-is-class-Object*:

$P \vdash D \text{ has-fields FDTs} \implies \text{is-class } P \ \text{Object}$

$\langle \text{proof} \rangle$

**lemma** *Object-fields*:

$\llbracket P \vdash \text{Object has-fields FDTs}; C \neq \text{Object} \rrbracket \implies \text{map-of } \text{FDTs} (F, C) = \text{None}$

$\langle \text{proof} \rangle$

**definition** *has-field* :: 'm prog  $\Rightarrow$  cname  $\Rightarrow$  vname  $\Rightarrow$  staticb  $\Rightarrow$  ty  $\Rightarrow$  cname  $\Rightarrow$  bool

( $\langle \vdash - \text{ has } -, :- \text{ in } \rightarrow [51, 51, 51, 51, 51, 51] \ 50$ )

**where**

$P \vdash C \text{ has } F, b: T \text{ in } D \equiv$   
 $\exists \text{ FDTs. } P \vdash C \text{ has-fields FDTs} \wedge \text{ map-of FDTs } (F, D) = \text{Some } (b, T)$

**lemma has-field-mono:**

**assumes** *has*:  $P \vdash C \text{ has } F, b: T \text{ in } D$  **and** *sub*:  $P \vdash C' \preceq^* C$

**shows**  $P \vdash C' \text{ has } F, b: T \text{ in } D$   $\langle \text{proof} \rangle$

**lemma has-field-fun:**

$\llbracket P \vdash C \text{ has } F, b: T \text{ in } D; P \vdash C \text{ has } F, b': T' \text{ in } D \rrbracket \implies b = b' \wedge T' = T$   $\langle \text{proof} \rangle$

**lemma has-field-idemp:**

**assumes** *has*:  $P \vdash C \text{ has } F, b: T \text{ in } D$

**shows**  $P \vdash D \text{ has } F, b: T \text{ in } D$   $\langle \text{proof} \rangle$

**lemma visible-fields-exist:**

**assumes** *fields*:  $P \vdash C \text{ has-fields FDTs}$  **and**

*FDTs*:  $\text{map-of FDTs } (F, D) = \text{Some } (b, T)$

**shows**  $\exists D' \text{ fs ms. class } P \ D = \text{Some}(D', \text{fs}, \text{ms}) \wedge \text{map-of fs } F = \text{Some}(b, T)$

$\langle \text{proof} \rangle$

**lemma map-of-remap-SomeD:**

$\text{map-of } (\text{map } (\lambda((k, k'), x). (k, (k', x))) \ t) \ k = \text{Some } (k', x) \implies \text{map-of } t \ (k, k') = \text{Some } x$   $\langle \text{proof} \rangle$

**lemma map-of-remap-SomeD2:**

$\text{map-of } (\text{map } (\lambda((k, k'), x, x'). (k, (k', x, x'))) \ t) \ k = \text{Some } (k', x, x') \implies \text{map-of } t \ (k, k') = \text{Some } (x, x')$   $\langle \text{proof} \rangle$

**lemma has-field-decl-above:**

$P \vdash C \text{ has } F, b: T \text{ in } D \implies P \vdash C \preceq^* D$   $\langle \text{proof} \rangle$

**definition sees-field** ::  $'m \ \text{prog} \Rightarrow \text{cname} \Rightarrow \text{vname} \Rightarrow \text{staticb} \Rightarrow \text{ty} \Rightarrow \text{cname} \Rightarrow \text{bool}$

( $\langle \vdash - \text{ sees } -, :- \text{ in } \rightarrow [51, 51, 51, 51, 51, 51] \ 50$ )

**where**

$P \vdash C \text{ sees } F, b: T \text{ in } D \equiv$

$\exists \text{ FDTs. } P \vdash C \text{ has-fields FDTs} \wedge$

$\text{map-of } (\text{map } (\lambda((F, D), b, T). (F, (D, b, T))) \ \text{FDTs}) \ F = \text{Some}(D, b, T)$

**lemma has-visible-field:**

$P \vdash C \text{ sees } F, b: T \text{ in } D \implies P \vdash C \text{ has } F, b: T \text{ in } D$   $\langle \text{proof} \rangle$

**lemma sees-field-fun:**

$\llbracket P \vdash C \text{ sees } F, b: T \text{ in } D; P \vdash C \text{ sees } F, b': T' \text{ in } D \rrbracket \implies b = b' \wedge T' = T \wedge D' = D$   $\langle \text{proof} \rangle$

**lemma sees-field-decl-above:**

$P \vdash C \text{ sees } F, b: T \text{ in } D \implies P \vdash C \preceq^* D$   $\langle \text{proof} \rangle$

**lemma sees-field-idemp:**

**assumes** *sees*:  $P \vdash C \text{ sees } F, b: T \text{ in } D$

**shows**  $P \vdash D \text{ sees } F, b: T \text{ in } D$   $\langle \text{proof} \rangle$

**lemma has-field-sees-aux:**

**assumes** *hf*:  $P \vdash C \text{ has-fields FDTs}$  **and** *map*:  $\text{map-of FDTs } (F, C) = \llbracket (b, T) \rrbracket$

**shows**  $\text{map-of } (\text{map } (\lambda((F, D), b, T). (F, D, b, T)) \ \text{FDTs}) \ F = \llbracket (C, b, T) \rrbracket$

$\langle \text{proof} \rangle$

**lemma has-field-sees**:  $P \vdash C \text{ has } F, b: T \text{ in } C \implies P \vdash C \text{ sees } F, b: T \text{ in } C$

$\langle \text{proof} \rangle$

**lemma has-field-is-class:**

$P \vdash C \text{ has } F, b: T \text{ in } D \implies \text{is-class } P C \langle \text{proof} \rangle$

**lemma** *has-field-is-class'*:

$P \vdash C \text{ has } F, b: T \text{ in } D \implies \text{is-class } P D \langle \text{proof} \rangle$

### 1.4.5 Functional lookup

**definition** *method* :: 'm prog  $\Rightarrow$  cname  $\Rightarrow$  mname  $\Rightarrow$  cname  $\times$  staticb  $\times$  ty list  $\times$  ty  $\times$  'm

**where**

$\text{method } P C M \equiv \text{THE } (D, b, Ts, T, m). P \vdash C \text{ sees } M, b: Ts \rightarrow T = m \text{ in } D$

**definition** *field* :: 'm prog  $\Rightarrow$  cname  $\Rightarrow$  vname  $\Rightarrow$  cname  $\times$  staticb  $\times$  ty

**where**

$\text{field } P C F \equiv \text{THE } (D, b, T). P \vdash C \text{ sees } F, b: T \text{ in } D$

**definition** *fields* :: 'm prog  $\Rightarrow$  cname  $\Rightarrow$  ((vname  $\times$  cname)  $\times$  staticb  $\times$  ty) list

**where**

$\text{fields } P C \equiv \text{THE FDTs}. P \vdash C \text{ has-fields FDTs}$

**lemma** *fields-def2* [simp]:  $P \vdash C \text{ has-fields FDTs} \implies \text{fields } P C = \text{FDTs} \langle \text{proof} \rangle$

**lemma** *field-def2* [simp]:  $P \vdash C \text{ sees } F, b: T \text{ in } D \implies \text{field } P C F = (D, b, T) \langle \text{proof} \rangle$

**lemma** *method-def2* [simp]:  $P \vdash C \text{ sees } M, b: Ts \rightarrow T = m \text{ in } D \implies \text{method } P C M = (D, b, Ts, T, m) \langle \text{proof} \rangle$

The following are the fields for initializing an object (non-static fields) and a class (just that class's static fields), respectively.

**definition** *ifields* :: 'm prog  $\Rightarrow$  cname  $\Rightarrow$  ((vname  $\times$  cname)  $\times$  staticb  $\times$  ty) list

**where**

$\text{ifields } P C \equiv \text{filter } (\lambda((F, D), b, T). b = \text{NonStatic}) (\text{fields } P C)$

**definition** *isfields* :: 'm prog  $\Rightarrow$  cname  $\Rightarrow$  ((vname  $\times$  cname)  $\times$  staticb  $\times$  ty) list

**where**

$\text{isfields } P C \equiv \text{filter } (\lambda((F, D), b, T). b = \text{Static} \wedge D = C) (\text{fields } P C)$

**lemma** *ifields-def2*[simp]:  $\llbracket P \vdash C \text{ has-fields FDTs} \rrbracket \implies \text{ifields } P C = \text{filter } (\lambda((F, D), b, T). b = \text{NonStatic}) \text{ FDTs} \langle \text{proof} \rangle$

**lemma** *isfields-def2*[simp]:  $\llbracket P \vdash C \text{ has-fields FDTs} \rrbracket \implies \text{isfields } P C = \text{filter } (\lambda((F, D), b, T). b = \text{Static} \wedge D = C) \text{ FDTs} \langle \text{proof} \rangle$

**lemma** *ifields-def3*:  $\llbracket P \vdash C \text{ sees } F, b: T \text{ in } D; b = \text{NonStatic} \rrbracket \implies (((F, D), b, T) \in \text{set } (\text{ifields } P C)) \langle \text{proof} \rangle$

**lemma** *isfields-def3*:  $\llbracket P \vdash C \text{ sees } F, b: T \text{ in } D; b = \text{Static}; D = C \rrbracket \implies (((F, D), b, T) \in \text{set } (\text{isfields } P C)) \langle \text{proof} \rangle$

**definition** *seeing-class* :: 'm prog  $\Rightarrow$  cname  $\Rightarrow$  mname  $\Rightarrow$  cname option **where**

$\text{seeing-class } P C M =$

(if  $\exists Ts T m D. P \vdash C \text{ sees } M, \text{Static}: Ts \rightarrow T = m \text{ in } D$

then  $\text{Some } (\text{fst}(\text{method } P C M))$

else  $\text{None}$ )

**lemma** *seeing-class-def2*[simp]:

$P \vdash C \text{ sees } M, \text{Static}: Ts \rightarrow T = m \text{ in } D \implies \text{seeing-class } P C M = \text{Some } D$

$\langle \text{proof} \rangle$

## 1.5 Jinja Values

**theory** *Value* **imports** *TypeRel* **begin**

**type-synonym** *addr* = *nat*

**datatype** *val*

= *Unit* — dummy result value of void expressions  
 | *Null* — null reference  
 | *Bool bool* — Boolean value  
 | *Intg int* — integer value  
 | *Addr addr* — addresses of objects in the heap

**primrec** *the-Intg* :: *val*  $\Rightarrow$  *int* **where**

*the-Intg* (*Intg i*) = *i*

**primrec** *the-Addr* :: *val*  $\Rightarrow$  *addr* **where**

*the-Addr* (*Addr a*) = *a*

**primrec** *default-val* :: *ty*  $\Rightarrow$  *val* — default value for all types **where**

*default-val Void* = *Unit*  
 | *default-val Boolean* = *Bool False*  
 | *default-val Integer* = *Intg 0*  
 | *default-val NT* = *Null*  
 | *default-val (Class C)* = *Null*

**end**

## 1.6 Objects and the Heap

**theory** *Objects* **imports** *TypeRel Value* **begin**

### 1.6.1 Objects

**type-synonym**

*fields* = *vname*  $\times$  *cname*  $\rightarrow$  *val* — field name, defining class, value

**type-synonym**

*obj* = *cname*  $\times$  *fields* — class instance with class name and fields

**type-synonym**

*sfields* = *vname*  $\rightarrow$  *val* — field name to value

**definition** *obj-ty* :: *obj*  $\Rightarrow$  *ty*

**where**

*obj-ty obj*  $\equiv$  *Class (fst obj)*

— initializes a given list of fields

**definition** *init-fields* :: ((*vname*  $\times$  *cname*)  $\times$  *staticb*  $\times$  *ty*) *list*  $\Rightarrow$  *fields*

**where**

*init-fields FDTs*  $\equiv$  (*map-of*  $\circ$  *map* ( $\lambda((F,D),b,T). ((F,D),\text{default-val } T)$ )) *FDTs*

**definition** *init-sfields* :: ((*vname*  $\times$  *cname*)  $\times$  *staticb*  $\times$  *ty*) *list*  $\Rightarrow$  *sfields*

**where**

*init-sfields FDTs*  $\equiv$  (*map-of*  $\circ$  *map* ( $\lambda((F,D),b,T). (F,\text{default-val } T)$ )) *FDTs*

— a new, blank object with default values for instance fields:

**definition**  $blank :: 'm\ prog \Rightarrow cname \Rightarrow obj$

**where**

$blank\ P\ C \equiv (C, init-fields\ (ifields\ P\ C))$

— a new, blank object with default values for static fields:

**definition**  $sblank :: 'm\ prog \Rightarrow cname \Rightarrow sfields$

**where**

$sblank\ P\ C \equiv init-sfields\ (isfields\ P\ C)$

**lemma**  $[simp]: obj-ty\ (C,fs) = Class\ C\langle proof \rangle$

**translations**

$(type)\ fields\ <= (type)\ char\ list \times char\ list \Rightarrow val\ option$

$(type)\ obj\ <= (type)\ char\ list \times fields$

$(type)\ sfields\ <= (type)\ char\ list \Rightarrow val\ option$

## 1.6.2 Heap

**type-synonym**  $heap = addr \rightarrow obj$

**translations**

$(type)\ heap\ <= (type)\ nat \Rightarrow obj\ option$

**abbreviation**

$cname-of :: heap \Rightarrow addr \Rightarrow cname$  **where**

$cname-of\ hp\ a == fst\ (the\ (hp\ a))$

**definition**  $new-Addr :: heap \Rightarrow addr\ option$

**where**

$new-Addr\ h \equiv if\ \exists a. h\ a = None\ then\ Some(LEAST\ a.\ h\ a = None)\ else\ None$

**definition**  $cast-ok :: 'm\ prog \Rightarrow cname \Rightarrow heap \Rightarrow val \Rightarrow bool$

**where**

$cast-ok\ P\ C\ h\ v \equiv v = Null \vee P \vdash cname-of\ h\ (the-Addr\ v) \preceq^* C$

**definition**  $hext :: heap \Rightarrow heap \Rightarrow bool\ (\prec \trianglelefteq \rightarrow [51,51]\ 50)$

**where**

$h \trianglelefteq h' \equiv \forall a\ C\ fs. h\ a = Some(C,fs) \longrightarrow (\exists fs'. h'\ a = Some(C,fs'))$

**primrec**  $typeof-h :: heap \Rightarrow val \Rightarrow ty\ option\ (\langle typeof-\rangle)$

**where**

$typeof_h\ Unit = Some\ Void$   
 $|\ typeof_h\ Null = Some\ NT$   
 $| \ typeof_h\ (Bool\ b) = Some\ Boolean$   
 $| \ typeof_h\ (Intg\ i) = Some\ Integer$   
 $| \ typeof_h\ (Addr\ a) = (case\ h\ a\ of\ None \Rightarrow None\ | \ Some(C,fs) \Rightarrow Some(Class\ C))$

**lemma**  $new-Addr-SomeD:$

$new-Addr\ h = Some\ a \implies h\ a = None$

$\langle proof \rangle$

**lemma**  $[simp]: (typeof_h\ v = Some\ Boolean) = (\exists b. v = Bool\ b)$

$\langle proof \rangle$



**lemma** [simp]:  $(\text{typeof}_h v = \text{Some Integer}) = (\exists i. v = \text{Intg } i)\langle \text{proof} \rangle$

**lemma** [simp]:  $(\text{typeof}_h v = \text{Some NT}) = (v = \text{Null})$   
 $\langle \text{proof} \rangle$

**lemma** [simp]:  $(\text{typeof}_h v = \text{Some}(\text{Class } C)) = (\exists a fs. v = \text{Addr } a \wedge h a = \text{Some}(C, fs))$   
 $\langle \text{proof} \rangle$

**lemma** [simp]:  $h a = \text{Some}(C, fs) \implies \text{typeof}(h(a \mapsto (C, fs))) v = \text{typeof}_h v$   
 $\langle \text{proof} \rangle$

For literal values the first parameter of *typeof* can be set to  $\{\}$  because they do not contain addresses:

#### abbreviation

*typeof* ::  $val \Rightarrow ty \text{ option}$  **where**  
*typeof*  $v == \text{typeof-h Map.empty } v$

**lemma** *typeof-lit-typeof*:

$\text{typeof } v = \text{Some } T \implies \text{typeof}_h v = \text{Some } T$   
 $\langle \text{proof} \rangle$

**lemma** *typeof-lit-is-type*:

$\text{typeof } v = \text{Some } T \implies \text{is-type } P T$   
 $\langle \text{proof} \rangle$

### 1.6.3 Heap extension $\triangleleft$

**lemma** *hextI*:  $\forall a C fs. h a = \text{Some}(C, fs) \longrightarrow (\exists fs'. h' a = \text{Some}(C, fs')) \implies h \triangleleft h' \langle \text{proof} \rangle$

**lemma** *hext-objD*:  $\llbracket h \triangleleft h'; h a = \text{Some}(C, fs) \rrbracket \implies \exists fs'. h' a = \text{Some}(C, fs') \langle \text{proof} \rangle$

**lemma** *hext-refl* [iff]:  $h \triangleleft h \langle \text{proof} \rangle$

**lemma** *hext-new* [simp]:  $h a = \text{None} \implies h \triangleleft h(a \mapsto x) \langle \text{proof} \rangle$

**lemma** *hext-trans*:  $\llbracket h \triangleleft h'; h' \triangleleft h'' \rrbracket \implies h \triangleleft h'' \langle \text{proof} \rangle$

**lemma** *hext-upd-obj*:  $h a = \text{Some}(C, fs) \implies h \triangleleft h(a \mapsto (C, fs')) \langle \text{proof} \rangle$

**lemma** *hext-typeof-mono*:  $\llbracket h \triangleleft h'; \text{typeof}_h v = \text{Some } T \rrbracket \implies \text{typeof}_{h'} v = \text{Some } T \langle \text{proof} \rangle$

### 1.6.4 Static field information function

**datatype** *init-state* = *Done* | *Processing* | *Prepared* | *Error*

- *Done* = initialized
- *Processing* = currently being initialized
- *Prepared* = uninitialized and not currently being initialized
- *Error* = previous initialization attempt resulted in erroneous state

**inductive** *iprogram* :: *init-state*  $\Rightarrow$  *init-state*  $\Rightarrow$  *bool* ( $\langle \cdot \leq_i \cdot \rightarrow [51, 51] 50 \rangle$ )

**where**

- [simp]: *Prepared*  $\leq_i$  *i*
- | [simp]: *Processing*  $\leq_i$  *Done*
- | [simp]: *Processing*  $\leq_i$  *Error*
- | [simp]: *i*  $\leq_i$  *i*

**lemma** *iprogram-Done*[simp]:  $(\text{Done} \leq_i i) = (i = \text{Done})$   
 $\langle \text{proof} \rangle$

**lemma** *iprogram-Error*[simp]:  $(\text{Error} \leq_i i) = (i = \text{Error})$   
 $\langle \text{proof} \rangle$

**lemma** *iprogram-Processing*[simp]:  $(\text{Processing} \leq_i i) = (i = \text{Done} \vee i = \text{Error} \vee i = \text{Processing})$   
 $\langle \text{proof} \rangle$

**lemma** *iprogram-trans*:  $\llbracket i \leq_i i'; i' \leq_i i'' \rrbracket \Longrightarrow i \leq_i i'' \langle \text{proof} \rangle$

### 1.6.5 Static Heap

The static heap (sheap) is used for storing information about static field values and initialization status for classes.

**type-synonym**

*sheap* = *cname*  $\rightarrow$  *sfields*  $\times$  *init-state*

**translations**

(*type*) *sheap*  $\Leftarrow$  (*type*) *char list*  $\Rightarrow$  (*sfields*  $\times$  *init-state*) *option*

**definition** *shext* :: *sheap*  $\Rightarrow$  *sheap*  $\Rightarrow$  *bool* ( $\langle \cdot \rangle_s \rightarrow [51, 51]$  50)

**where**

$sh \leq_s sh' \equiv \forall C \text{ sfs } i. sh \ C = \text{Some}(sfs, i) \longrightarrow (\exists sfs' \ i'. sh' \ C = \text{Some}(sfs', i') \wedge i \leq_i i')$

**lemma** *shextI*:  $\forall C \text{ sfs } i. sh \ C = \text{Some}(sfs, i) \longrightarrow (\exists sfs' \ i'. sh' \ C = \text{Some}(sfs', i') \wedge i \leq_i i') \Longrightarrow sh \leq_s sh' \langle \text{proof} \rangle$

**lemma** *shext-objD*:  $\llbracket sh \leq_s sh'; sh \ C = \text{Some}(sfs, i) \rrbracket \Longrightarrow \exists sfs' \ i'. sh' \ C = \text{Some}(sfs', i') \wedge i \leq_i i' \langle \text{proof} \rangle$

**lemma** *shext-refl* [*iff*]:  $sh \leq_s sh \langle \text{proof} \rangle$

**lemma** *shext-new* [*simp*]:  $sh \ C = \text{None} \Longrightarrow sh \leq_s sh(C \mapsto x) \langle \text{proof} \rangle$

**lemma** *shext-trans*:  $\llbracket sh \leq_s sh'; sh' \leq_s sh'' \rrbracket \Longrightarrow sh \leq_s sh'' \langle \text{proof} \rangle$

**lemma** *shext-upd-obj*:  $\llbracket sh \ C = \text{Some}(sfs, i); i \leq_i i' \rrbracket \Longrightarrow sh \leq_s sh(C \mapsto (sfs', i')) \langle \text{proof} \rangle$

**end**

## 1.7 Exceptions

**theory** *Exceptions* **imports** *Objects* **begin**

**definition** *ErrorCl* :: *string* **where** *ErrorCl* = "Error"

**definition** *ThrowCl* :: *string* **where** *ThrowCl* = "Throwable"

**definition** *NullPointer* :: *cname*

**where**

*NullPointer*  $\equiv$  "NullPointer"

**definition** *ClassCast* :: *cname*

**where**

*ClassCast*  $\equiv$  "ClassCast"

**definition** *OutOfMemory* :: *cname*

**where**

*OutOfMemory*  $\equiv$  "OutOfMemory"

**definition** *NoClassDefFoundError* :: *cname*

**where**

*NoClassDefFoundError*  $\equiv$  "NoClassDefFoundError"

**definition** *IncompatibleClassChangeError* :: *cname*

**where**

*IncompatibleClassChangeError*  $\equiv$  "IncompatibleClassChangeError"

**definition** *NoSuchFieldError* :: *cname*

**where**

*NoSuchFieldError*  $\equiv$  "NoSuchFieldError"

**definition** *NoSuchMethodError* :: *cname*

**where**

*NoSuchMethodError*  $\equiv$  "NoSuchMethodError"

**definition** *sys-xcpts* :: *cname set*

**where**

*sys-xcpts*  $\equiv$  {*NullPointer*, *ClassCast*, *OutOfMemory*, *NoClassDefFoundError*,  
*IncompatibleClassChangeError*,  
*NoSuchFieldError*, *NoSuchMethodError*}

**definition** *addr-of-sys-xcpt* :: *cname*  $\Rightarrow$  *addr*

**where**

*addr-of-sys-xcpt* *s*  $\equiv$  if *s* = *NullPointer* then 0 else  
if *s* = *ClassCast* then 1 else  
if *s* = *OutOfMemory* then 2 else  
if *s* = *NoClassDefFoundError* then 3 else  
if *s* = *IncompatibleClassChangeError* then 4 else  
if *s* = *NoSuchFieldError* then 5 else  
if *s* = *NoSuchMethodError* then 6 else undefined

**lemmas** *sys-xcpts-defs* = *NullPointer-def* *ClassCast-def* *OutOfMemory-def* *NoClassDefFoundError-def*  
*IncompatibleClassChangeError-def* *NoSuchFieldError-def* *NoSuchMethodError-def*

**lemma** *Start-nsys-xcpts*: *Start*  $\notin$  *sys-xcpts*

*<proof>*

**lemma** *Start-nsys-xcpts1* [*simp*]: *Start*  $\neq$  *NullPointer* *Start*  $\neq$  *ClassCast*

*Start*  $\neq$  *OutOfMemory* *Start*  $\neq$  *NoClassDefFoundError*

*Start*  $\neq$  *IncompatibleClassChangeError* *Start*  $\neq$  *NoSuchFieldError*

*Start*  $\neq$  *NoSuchMethodError*

*<proof>*

**lemma** *Start-nsys-xcpts2* [*simp*]: *NullPointer*  $\neq$  *Start* *ClassCast*  $\neq$  *Start*

*OutOfMemory*  $\neq$  *Start* *NoClassDefFoundError*  $\neq$  *Start*

*IncompatibleClassChangeError*  $\neq$  *Start* *NoSuchFieldError*  $\neq$  *Start*

*NoSuchMethodError*  $\neq$  *Start*

*<proof>*

**definition** *start-heap* :: 'c *prog*  $\Rightarrow$  *heap*

**where**

*start-heap* *G*  $\equiv$  *Map.empty* (*addr-of-sys-xcpt* *NullPointer*  $\mapsto$  *blank* *G* *NullPointer*,  
*addr-of-sys-xcpt* *ClassCast*  $\mapsto$  *blank* *G* *ClassCast*,  
*addr-of-sys-xcpt* *OutOfMemory*  $\mapsto$  *blank* *G* *OutOfMemory*,  
*addr-of-sys-xcpt* *NoClassDefFoundError*  $\mapsto$  *blank* *G* *NoClassDefFoundError*,  
*addr-of-sys-xcpt* *IncompatibleClassChangeError*  $\mapsto$  *blank* *G* *IncompatibleClass-*  
*ChangeError*,  
*addr-of-sys-xcpt* *NoSuchFieldError*  $\mapsto$  *blank* *G* *NoSuchFieldError*,

$addr\text{-}of\text{-}sys\text{-}xcpt\ NoSuchMethodError \mapsto blank\ G\ NoSuchMethodError$ )

**definition**  $preallocated :: heap \Rightarrow bool$

**where**

$preallocated\ h \equiv \forall C \in sys\text{-}xcpts. \exists fs. h(addr\text{-}of\text{-}sys\text{-}xcpt\ C) = Some\ (C, fs)$

### 1.7.1 System exceptions

**lemma**  $sys\text{-}xcpts\text{-}incl$  [*simp*]:  $NullPointer \in sys\text{-}xcpts \wedge OutOfMemory \in sys\text{-}xcpts$   
 $\wedge ClassCast \in sys\text{-}xcpts \wedge NoClassDefFoundError \in sys\text{-}xcpts$   
 $\wedge IncompatibleClassChangeError \in sys\text{-}xcpts \wedge NoSuchFieldError \in sys\text{-}xcpts$   
 $\wedge NoSuchMethodError \in sys\text{-}xcpts$ ⟨*proof*⟩

**lemma**  $sys\text{-}xcpts\text{-}cases$  [*consumes 1, cases set*]:

$\llbracket C \in sys\text{-}xcpts; P\ NullPointer; P\ OutOfMemory; P\ ClassCast; P\ NoClassDefFoundError;$   
 $P\ IncompatibleClassChangeError; P\ NoSuchFieldError;$   
 $P\ NoSuchMethodError \rrbracket \Longrightarrow P\ C$ ⟨*proof*⟩

### 1.7.2 Starting heap

**lemma**  $start\text{-}heap\text{-}sys\text{-}xcpts$ :

**assumes**  $C \in sys\text{-}xcpts$

**shows**  $start\text{-}heap\ P\ (addr\text{-}of\text{-}sys\text{-}xcpt\ C) = Some(blank\ P\ C)$

⟨*proof*⟩

**lemma**  $start\text{-}heap\text{-}classes$ :

$start\text{-}heap\ P\ a = Some(C, fs) \Longrightarrow C \in sys\text{-}xcpts$

⟨*proof*⟩

**lemma**  $start\text{-}heap\text{-}nStart$ :  $start\text{-}heap\ P\ a = Some\ obj \Longrightarrow fst(obj) \neq Start$

⟨*proof*⟩

### 1.7.3 preallocated

**lemma**  $preallocated\text{-}dom$  [*simp*]:

$\llbracket preallocated\ h; C \in sys\text{-}xcpts \rrbracket \Longrightarrow addr\text{-}of\text{-}sys\text{-}xcpt\ C \in dom\ h$ ⟨*proof*⟩

**lemma**  $preallocatedD$ :

$\llbracket preallocated\ h; C \in sys\text{-}xcpts \rrbracket \Longrightarrow \exists fs. h(addr\text{-}of\text{-}sys\text{-}xcpt\ C) = Some\ (C, fs)$ ⟨*proof*⟩

**lemma**  $preallocatedE$  [*elim?*]:

$\llbracket preallocated\ h; C \in sys\text{-}xcpts; \bigwedge fs. h(addr\text{-}of\text{-}sys\text{-}xcpt\ C) = Some(C, fs) \Longrightarrow P\ h\ C \rrbracket$   
 $\Longrightarrow P\ h\ C$ ⟨*proof*⟩

**lemma**  $cname\text{-}of\text{-}xcp$  [*simp*]:

$\llbracket preallocated\ h; C \in sys\text{-}xcpts \rrbracket \Longrightarrow cname\text{-}of\ h\ (addr\text{-}of\text{-}sys\text{-}xcpt\ C) = C$ ⟨*proof*⟩

**lemma**  $typeof\text{-}ClassCast$  [*simp*]:

$preallocated\ h \Longrightarrow typeof_h\ (Addr(addr\text{-}of\text{-}sys\text{-}xcpt\ ClassCast)) = Some(Class\ ClassCast)$ ⟨*proof*⟩

**lemma**  $typeof\text{-}OutOfMemory$  [*simp*]:

$preallocated\ h \Longrightarrow typeof_h\ (Addr(addr\text{-}of\text{-}sys\text{-}xcpt\ OutOfMemory)) = Some(Class\ OutOfMemory)$ ⟨*proof*⟩

**lemma**  $typeof\text{-}NullPointer$  [*simp*]:

$preallocated\ h \Longrightarrow typeof_h\ (Addr(addr\text{-}of\text{-}sys\text{-}xcpt\ NullPointer)) = Some(Class\ NullPointer)$ ⟨*proof*⟩

**lemma**  $typeof\text{-}NoClassDefFoundError$  [*simp*]:

$preallocated\ h \implies \text{typeof}_h (\text{Addr}(\text{addr-of-sys-xcpt}\ \text{NoClassDefFoundError})) = \text{Some}(\text{Class}\ \text{NoClassDefFoundError})$  $\langle\text{proof}\rangle$

**lemma** *typeof-IncompatibleClassChangeError* [simp]:

$preallocated\ h \implies \text{typeof}_h (\text{Addr}(\text{addr-of-sys-xcpt}\ \text{IncompatibleClassChangeError})) = \text{Some}(\text{Class}\ \text{IncompatibleClassChangeError})$  $\langle\text{proof}\rangle$

**lemma** *typeof-NoSuchFieldError* [simp]:

$preallocated\ h \implies \text{typeof}_h (\text{Addr}(\text{addr-of-sys-xcpt}\ \text{NoSuchFieldError})) = \text{Some}(\text{Class}\ \text{NoSuchFieldError})$  $\langle\text{proof}\rangle$

**lemma** *typeof-NoSuchMethodError* [simp]:

$preallocated\ h \implies \text{typeof}_h (\text{Addr}(\text{addr-of-sys-xcpt}\ \text{NoSuchMethodError})) = \text{Some}(\text{Class}\ \text{NoSuchMethodError})$  $\langle\text{proof}\rangle$

**lemma** *preallocated-hext*:

$\llbracket\ preallocated\ h; h \leq h' \rrbracket \implies preallocated\ h'$  $\langle\text{proof}\rangle$

**lemma** *preallocated-start*:

$preallocated\ (\text{start-heap}\ P)$   
 $\langle\text{proof}\rangle$

end

## 1.8 Expressions

**theory** *Expr*

**imports** *../Common/Exceptions*

**begin**

**datatype** *bop* = *Eq* | *Add* — names of binary operations

**datatype** *'a exp*

= *new cname* — class instance creation

| *Cast cname ('a exp)* — type cast

| *Val val* — value

| *BinOp ('a exp) bop ('a exp)* ( $\langle\text{--}\langle\text{--}\rangle\ [80,0,81]\ 80$ ) — binary operation

| *Var 'a* — local variable (incl. parameter)

| *LAss 'a ('a exp)* ( $\langle\text{:-}\rangle\ [90,90]90$ ) — local assignment

| *FAcc ('a exp) vname cname* ( $\langle\text{--}\{-}\rangle\ [10,90,99]90$ ) — field access

| *SFAcc cname vname cname* ( $\langle\text{--}_s\{-}\rangle\ [10,90,99]90$ ) — static field access

| *FAss ('a exp) vname cname ('a exp)* ( $\langle\text{--}\{-}\rangle\ :=\ \rightarrow\ [10,90,99,90]90$ ) — field assignment

| *SFAss cname vname cname ('a exp)* ( $\langle\text{--}_s\{-}\rangle\ :=\ \rightarrow\ [10,90,99,90]90$ ) — static field assignment

| *Call ('a exp) mname ('a exp list)* ( $\langle\text{--}\{-}\rangle\ [90,99,0]\ 90$ ) — method call

| *SCall cname mname ('a exp list)* ( $\langle\text{--}_s\{-}\rangle\ [90,99,0]\ 90$ ) — static method call

| *Block 'a ty ('a exp)* ( $\langle\text{'}\{-};\ -\rangle$ )

| *Seq ('a exp) ('a exp)* ( $\langle\text{--};\ / \rightarrow\ [61,60]60$ )

| *Cond ('a exp) ('a exp) ('a exp)* ( $\langle\text{if}\ \{-}\ / \text{else}\ \rightarrow\ [80,79,79]70$ )

| *While ('a exp) ('a exp)* ( $\langle\text{while}\ \{-}\ \rightarrow\ [80,79]70$ )

| *throw ('a exp)*

| *TryCatch ('a exp) cname 'a ('a exp)* ( $\langle\text{try}\ / \text{catch}\ \{-}\ \rightarrow\ [0,99,80,79]70$ )

| *INIT cname cname list bool ('a exp)* ( $\langle\text{INIT}\ \{-}\ \leftarrow\ \rightarrow\ [60,60,60,60]60$ ) — internal initialization command: class, list of superclasses to initialize, preparation flag; command on hold

| *RI cname ('a exp) cname list ('a exp)* ( $\langle\text{RI}\ \{-}\ ;\ -\ \leftarrow\ \rightarrow\ [60,60,60,60]60$ ) — running of the initialization procedure for class with expression, classes still to initialize command on hold

**type-synonym**

*expr* = *vname exp* — Jinja expression

**type-synonym**

$J\text{-mb} = \text{vname list} \times \text{expr}$  — Jinja method body: parameter names and expression

**type-synonym**

$J\text{-prog} = J\text{-mb prog}$  — Jinja program

**type-synonym**

$\text{init-stack} = \text{expr list} \times \text{bool}$  — Stack of expressions waiting on initialization in small step; indicator boolean True if current expression has been init checked

The semantics of binary operators:

**fun**  $\text{binop} :: \text{bop} \times \text{val} \times \text{val} \Rightarrow \text{val option}$  **where**

$\text{binop}(\text{Eq}, v_1, v_2) = \text{Some}(\text{Bool}(v_1 = v_2))$   
 $\text{binop}(\text{Add}, \text{Intg } i_1, \text{Intg } i_2) = \text{Some}(\text{Intg}(i_1 + i_2))$   
 $\text{binop}(\text{bop}, v_1, v_2) = \text{None}$

**lemma**  $[\text{simp}]$ :

$(\text{binop}(\text{Add}, v_1, v_2) = \text{Some } v) = (\exists i_1 i_2. v_1 = \text{Intg } i_1 \wedge v_2 = \text{Intg } i_2 \wedge v = \text{Intg}(i_1 + i_2)) \langle \text{proof} \rangle$

**lemma**  $\text{map-Val-throw-eq}$ :

$\text{map Val } vs @ \text{throw } ex \# es = \text{map Val } vs' @ \text{throw } ex' \# es' \Longrightarrow ex = ex' \langle \text{proof} \rangle$

**lemma**  $\text{map-Val-nthrow-neq}$ :

$\text{map Val } vs = \text{map Val } vs' @ \text{throw } ex' \# es' \Longrightarrow \text{False} \langle \text{proof} \rangle$

**lemma**  $\text{map-Val-eq}$ :

$\text{map Val } vs = \text{map Val } vs' \Longrightarrow vs = vs' \langle \text{proof} \rangle$

**lemma**  $\text{init-rhs-neq} [\text{simp}]$ :  $e \neq \text{INIT } C (Cs, b) \leftarrow e$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{init-rhs-neq}' [\text{simp}]$ :  $\text{INIT } C (Cs, b) \leftarrow e \neq e$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{ri-rhs-neq} [\text{simp}]$ :  $e \neq \text{RI}(C, e'); Cs \leftarrow e$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{ri-rhs-neq}' [\text{simp}]$ :  $\text{RI}(C, e'); Cs \leftarrow e \neq e$   
 $\langle \text{proof} \rangle$

### 1.8.1 Syntactic sugar

**abbreviation** (*input*)

$\text{InitBlock} :: 'a \Rightarrow \text{ty} \Rightarrow 'a \text{ exp} \Rightarrow 'a \text{ exp} \Rightarrow 'a \text{ exp} \quad (\langle (1 \{ \text{:-} := \text{-} / \text{-} \}) \rangle)$  **where**  
 $\text{InitBlock } V T e1 e2 == \{ V : T; V := e1;; e2 \}$

**abbreviation**  $\text{unit}$  **where**  $\text{unit} == \text{Val Unit}$

**abbreviation**  $\text{null}$  **where**  $\text{null} == \text{Val Null}$

**abbreviation**  $\text{addr } a == \text{Val}(\text{Addr } a)$

**abbreviation**  $\text{true} == \text{Val}(\text{Bool True})$

**abbreviation**  $\text{false} == \text{Val}(\text{Bool False})$

**abbreviation**

$\text{Throw} :: \text{addr} \Rightarrow 'a \text{ exp}$  **where**

$\text{Throw } a == \text{throw}(\text{Val}(\text{Addr } a))$

**abbreviation**

*THROW* :: *cname*  $\Rightarrow$  'a *exp* **where**  
*THROW* *xc* == *Throw(addr-of-sys-xcpt xc)*

### 1.8.2 Free Variables

**primrec** *fv* :: *expr*  $\Rightarrow$  *vname set* **and** *fvs* :: *expr list*  $\Rightarrow$  *vname set* **where**

*fv*(*new C*) = {}  
| *fv*(*Cast C e*) = *fv e*  
| *fv*(*Val v*) = {}  
| *fv*(*e<sub>1</sub> «bop» e<sub>2</sub>*) = *fv e<sub>1</sub> ∪ fv e<sub>2</sub>*  
| *fv*(*Var V*) = {*V*}  
| *fv*(*LAss V e*) = {*V*}  $\cup$  *fv e*  
| *fv*(*e · F{D}*) = *fv e*  
| *fv*(*C ·<sub>s</sub> F{D}*) = {}  
| *fv*(*e<sub>1</sub> · F{D} := e<sub>2</sub>*) = *fv e<sub>1</sub> ∪ fv e<sub>2</sub>*  
| *fv*(*C ·<sub>s</sub> F{D} := e<sub>2</sub>*) = *fv e<sub>2</sub>*  
| *fv*(*e · M(es)*) = *fv e ∪ fvs es*  
| *fv*(*C ·<sub>s</sub> M(es)*) = *fvs es*  
| *fv*({*V:T*; *e*}) = *fv e - {V}*  
| *fv*(*e<sub>1</sub>;; e<sub>2</sub>*) = *fv e<sub>1</sub> ∪ fv e<sub>2</sub>*  
| *fv*(*if (b) e<sub>1</sub> else e<sub>2</sub>*) = *fv b ∪ fv e<sub>1</sub> ∪ fv e<sub>2</sub>*  
| *fv*(*while (b) e*) = *fv b ∪ fv e*  
| *fv*(*throw e*) = *fv e*  
| *fv*(*try e<sub>1</sub> catch (C V) e<sub>2</sub>*) = *fv e<sub>1</sub> ∪ (fv e<sub>2</sub> - {V})*  
| *fv*(*INIT C (Cs,b) ← e*) = *fv e*  
| *fv*(*RI (C,e); Cs ← e'*) = *fv e ∪ fv e'*  
| *fvs*([]) = {}  
| *fvs*(*e#es*) = *fv e ∪ fvs es*

**lemma** [*simp*]: *fvs(es<sub>1</sub> @ es<sub>2</sub>)* = *fvs es<sub>1</sub> ∪ fvs es<sub>2</sub>*⟨*proof*⟩

**lemma** [*simp*]: *fvs(map Val vs)* = {}⟨*proof*⟩

### 1.8.3 Accessing expression constructor arguments

**fun** *val-of* :: 'a *exp*  $\Rightarrow$  *val option* **where**

*val-of* (*Val v*) = *Some v* |  
*val-of* - = *None*

**lemma** *val-of-spec*: *val-of e* = *Some v*  $\implies$  *e* = *Val v*  
⟨*proof*⟩

**fun** *lass-val-of* :: 'a *exp*  $\Rightarrow$  ('a  $\times$  *val*) *option* **where**

*lass-val-of* (*V:=Val v*) = *Some (V, v)* |  
*lass-val-of* - = *None*

**lemma** *lass-val-of-spec*:

**assumes** *lass-val-of e* = [*a*]

**shows** *e* = (*fst a* := *Val (snd a)*)

⟨*proof*⟩

**fun** *map-vals-of* :: 'a *exp list*  $\Rightarrow$  *val list option* **where**

*map-vals-of* (*e#es*) = (*case val-of e of Some v*  $\Rightarrow$  (*case map-vals-of es of Some vs*  $\Rightarrow$  *Some (v#vs)*  
| -  $\Rightarrow$  *None*) |

| -  $\Rightarrow$  *None*) |

*map-vals-of* [] = *Some* []

**lemma** *map-vals-of-spec*: *map-vals-of es = Some vs*  $\implies$  *es = map Val vs*  
 ⟨*proof*⟩

**lemma** *map-vals-of-Vals[simp]*: *map-vals-of (map Val vs) = [vs]* ⟨*proof*⟩

**lemma** *map-vals-of-throw[simp]*:  
*map-vals-of (map Val vs @ throw e # es')* = *None*  
 ⟨*proof*⟩

**fun** *bool-of* :: 'a exp  $\Rightarrow$  bool option **where**  
*bool-of true* = *Some True* |  
*bool-of false* = *Some False* |  
*bool-of -* = *None*

**lemma** *bool-of-specT*:  
**assumes** *bool-of e = Some True* **shows** *e = true*  
 ⟨*proof*⟩

**lemma** *bool-of-specF*:  
**assumes** *bool-of e = Some False* **shows** *e = false*  
 ⟨*proof*⟩

**fun** *throw-of* :: 'a exp  $\Rightarrow$  'a exp option **where**  
*throw-of (throw e')* = *Some e'* |  
*throw-of -* = *None*

**lemma** *throw-of-spec*: *throw-of e = Some e'*  $\implies$  *e = throw e'*  
 ⟨*proof*⟩

**fun** *init-exp-of* :: 'a exp  $\Rightarrow$  'a exp option **where**  
*init-exp-of (INIT C (Cs,b)  $\leftarrow$  e)* = *Some e* |  
*init-exp-of (RI(C,e');Cs  $\leftarrow$  e)* = *Some e* |  
*init-exp-of -* = *None*

**lemma** *init-exp-of-neq [simp]*: *init-exp-of e = [e']*  $\implies$  *e'  $\neq$  e* ⟨*proof*⟩

**lemma** *init-exp-of-neq'[simp]*: *init-exp-of e = [e']*  $\implies$  *e  $\neq$  e'* ⟨*proof*⟩

#### 1.8.4 Class initialization

This section defines a few functions that return information about an expression's current initialization status.

**primrec** *sub-RI* :: 'a exp  $\Rightarrow$  bool **and** *sub-RIs* :: 'a exp list  $\Rightarrow$  bool **where**  
*sub-RI(new C)* = *False*  
 | *sub-RI(Cast C e)* = *sub-RI e*  
 | *sub-RI(Val v)* = *False*  
 | *sub-RI(e<sub>1</sub> «bop» e<sub>2</sub>)* = (*sub-RI e<sub>1</sub>  $\vee$  sub-RI e<sub>2</sub>*)  
 | *sub-RI(Var V)* = *False*  
 | *sub-RI(LAss V e)* = *sub-RI e*  
 | *sub-RI(e.F{D})* = *sub-RI e*  
 | *sub-RI(C.<sub>s</sub>F{D})* = *False*



$| \text{sub-RI}(e_1 \cdot F\{D\};=e_2) = (\text{sub-RI } e_1 \vee \text{sub-RI } e_2)$   
 $| \text{sub-RI}(C \cdot_s F\{D\};=e_2) = \text{sub-RI } e_2$   
 $| \text{sub-RI}(e \cdot M(es)) = (\text{sub-RI } e \vee \text{sub-RIs } es)$   
 $| \text{sub-RI}(C \cdot_s M(es)) = (M = \text{clinit} \vee \text{sub-RIs } es)$   
 $| \text{sub-RI}(\{V:T; e\}) = \text{sub-RI } e$   
 $| \text{sub-RI}(e_1;;e_2) = (\text{sub-RI } e_1 \vee \text{sub-RI } e_2)$   
 $| \text{sub-RI}(\text{if } (b) e_1 \text{ else } e_2) = (\text{sub-RI } b \vee \text{sub-RI } e_1 \vee \text{sub-RI } e_2)$   
 $| \text{sub-RI}(\text{while } (b) e) = (\text{sub-RI } b \vee \text{sub-RI } e)$   
 $| \text{sub-RI}(\text{throw } e) = \text{sub-RI } e$   
 $| \text{sub-RI}(\text{try } e_1 \text{ catch}(C V) e_2) = (\text{sub-RI } e_1 \vee \text{sub-RI } e_2)$   
 $| \text{sub-RI}(\text{INIT } C (Cs,b) \leftarrow e) = \text{True}$   
 $| \text{sub-RI}(\text{RI } (C,e);Cs \leftarrow e') = \text{True}$   
 $| \text{sub-RIs}(\[]) = \text{False}$   
 $| \text{sub-RIs}(e\#es) = (\text{sub-RI } e \vee \text{sub-RIs } es)$

**lemmas** *sub-RI-sub-RIs-induct* = *sub-RI.induct sub-RIs.induct*

**lemma** *nsub-RIs-def[simp]*:

$\neg \text{sub-RIs } es \implies \forall e \in \text{set } es. \neg \text{sub-RI } e$

*<proof>*

**lemma** *sub-RI-base*:

$e = \text{INIT } C (Cs, b) \leftarrow e' \vee e = \text{RI}(C, e_0);Cs \leftarrow e' \implies \text{sub-RI } e$

*<proof>*

**lemma** *nsub-RI-Vals[simp]*:  $\neg \text{sub-RIs } (\text{map Val } vs)$

*<proof>*

**lemma** *lass-val-of-nsub-RI*:  $\text{lass-val-of } e = [a] \implies \neg \text{sub-RI } e$

*<proof>*

**primrec** *not-init* :: *cname*  $\Rightarrow$  'a *exp*  $\Rightarrow$  *bool* **and** *not-inits* :: *cname*  $\Rightarrow$  'a *exp list*  $\Rightarrow$  *bool* **where**

$\text{not-init } C' (\text{new } C) = \text{True}$   
 $| \text{not-init } C' (\text{Cast } C e) = \text{not-init } C' e$   
 $| \text{not-init } C' (\text{Val } v) = \text{True}$   
 $| \text{not-init } C' (e_1 \ll \text{bop} \gg e_2) = (\text{not-init } C' e_1 \wedge \text{not-init } C' e_2)$   
 $| \text{not-init } C' (\text{Var } V) = \text{True}$   
 $| \text{not-init } C' (\text{LAss } V e) = \text{not-init } C' e$   
 $| \text{not-init } C' (e \cdot F\{D\}) = \text{not-init } C' e$   
 $| \text{not-init } C' (C \cdot_s F\{D\}) = \text{True}$   
 $| \text{not-init } C' (e_1 \cdot F\{D\};=e_2) = (\text{not-init } C' e_1 \wedge \text{not-init } C' e_2)$   
 $| \text{not-init } C' (C \cdot_s F\{D\};=e_2) = \text{not-init } C' e_2$   
 $| \text{not-init } C' (e \cdot M(es)) = (\text{not-init } C' e \wedge \text{not-inits } C' es)$   
 $| \text{not-init } C' (C \cdot_s M(es)) = \text{not-inits } C' es$   
 $| \text{not-init } C' (\{V:T; e\}) = \text{not-init } C' e$   
 $| \text{not-init } C' (e_1;;e_2) = (\text{not-init } C' e_1 \wedge \text{not-init } C' e_2)$   
 $| \text{not-init } C' (\text{if } (b) e_1 \text{ else } e_2) = (\text{not-init } C' b \wedge \text{not-init } C' e_1 \wedge \text{not-init } C' e_2)$   
 $| \text{not-init } C' (\text{while } (b) e) = (\text{not-init } C' b \wedge \text{not-init } C' e)$   
 $| \text{not-init } C' (\text{throw } e) = \text{not-init } C' e$   
 $| \text{not-init } C' (\text{try } e_1 \text{ catch}(C V) e_2) = (\text{not-init } C' e_1 \wedge \text{not-init } C' e_2)$   
 $| \text{not-init } C' (\text{INIT } C (Cs,b) \leftarrow e) = ((b \longrightarrow Cs = \text{Nil} \vee C' \neq \text{hd } Cs) \wedge C' \notin \text{set}(\text{tl } Cs) \wedge \text{not-init } C' e)$   
 $| \text{not-init } C' (\text{RI } (C,e);Cs \leftarrow e') = (C' \notin \text{set}(C\#Cs) \wedge \text{not-init } C' e \wedge \text{not-init } C' e')$   
 $| \text{not-inits } C' (\[]) = \text{True}$

|  $\text{not-inits } C' (e\#es) = (\text{not-init } C' e \wedge \text{not-inits } C' es)$

**lemma** *not-inits-def'[simp]*:

$\text{not-inits } C es \implies \forall e \in \text{set } es. \text{not-init } C e$   
 ⟨proof⟩

**lemma** *nsub-RIs-not-inits-aux*:  $\forall e \in \text{set } es. \neg \text{sub-RI } e \implies \text{not-init } C e$

$\implies \neg \text{sub-RIs } es \implies \text{not-inits } C es$   
 ⟨proof⟩

**lemma** *nsub-RI-not-init*:  $\neg \text{sub-RI } e \implies \text{not-init } C e$

⟨proof⟩

**lemma** *nsub-RIs-not-inits*:  $\neg \text{sub-RIs } es \implies \text{not-inits } C es$

⟨proof⟩

### 1.8.5 Subexpressions

**primrec** *subexp* :: 'a exp  $\Rightarrow$  'a exp set **and** *subexps* :: 'a exp list  $\Rightarrow$  'a exp set **where**

$\text{subexp}(\text{new } C) = \{\}$   
 |  $\text{subexp}(\text{Cast } C e) = \{e\} \cup \text{subexp } e$   
 |  $\text{subexp}(\text{Val } v) = \{\}$   
 |  $\text{subexp}(e_1 \ll \text{bop} \gg e_2) = \{e_1, e_2\} \cup \text{subexp } e_1 \cup \text{subexp } e_2$   
 |  $\text{subexp}(\text{Var } V) = \{\}$   
 |  $\text{subexp}(\text{LAss } V e) = \{e\} \cup \text{subexp } e$   
 |  $\text{subexp}(e \cdot F\{D\}) = \{e\} \cup \text{subexp } e$   
 |  $\text{subexp}(C \cdot_s F\{D\}) = \{\}$   
 |  $\text{subexp}(e_1 \cdot F\{D\} := e_2) = \{e_1, e_2\} \cup \text{subexp } e_1 \cup \text{subexp } e_2$   
 |  $\text{subexp}(C \cdot_s F\{D\} := e_2) = \{e_2\} \cup \text{subexp } e_2$   
 |  $\text{subexp}(e \cdot M(es)) = \{e\} \cup \text{set } es \cup \text{subexp } e \cup \text{subexps } es$   
 |  $\text{subexp}(C \cdot_s M(es)) = \text{set } es \cup \text{subexps } es$   
 |  $\text{subexp}(\{V:T; e\}) = \{e\} \cup \text{subexp } e$   
 |  $\text{subexp}(e_1 ;; e_2) = \{e_1, e_2\} \cup \text{subexp } e_1 \cup \text{subexp } e_2$   
 |  $\text{subexp}(\text{if } (b) e_1 \text{ else } e_2) = \{b, e_1, e_2\} \cup \text{subexp } b \cup \text{subexp } e_1 \cup \text{subexp } e_2$   
 |  $\text{subexp}(\text{while } (b) e) = \{b, e\} \cup \text{subexp } b \cup \text{subexp } e$   
 |  $\text{subexp}(\text{throw } e) = \{e\} \cup \text{subexp } e$   
 |  $\text{subexp}(\text{try } e_1 \text{ catch } (C V) e_2) = \{e_1, e_2\} \cup \text{subexp } e_1 \cup \text{subexp } e_2$   
 |  $\text{subexp}(\text{INIT } C (Cs, b) \leftarrow e) = \{e\} \cup \text{subexp } e$   
 |  $\text{subexp}(\text{RI } (C, e); Cs \leftarrow e') = \{e, e'\} \cup \text{subexp } e \cup \text{subexp } e'$   
 |  $\text{subexps}(\[]) = \{\}$   
 |  $\text{subexps}(e\#es) = \{e\} \cup \text{subexp } e \cup \text{subexps } es$

**lemmas** *subexp-subexps-induct* = *subexp.induct subexps.induct*

**abbreviation** *subexp-of* :: 'a exp  $\Rightarrow$  'a exp  $\Rightarrow$  bool **where**

$\text{subexp-of } e e' \equiv e \in \text{subexp } e'$

**lemma** *subexp-size-le*:

$(e' \in \text{subexp } e \implies \text{size } e' < \text{size } e) \wedge (e' \in \text{subexps } es \implies \text{size } e' < \text{size-list } \text{size } es)$   
 ⟨proof⟩

**lemma** *subexps-def2*:  $\text{subexps } es = \text{set } es \cup (\bigcup e \in \text{set } es. \text{subexp } e)$  ⟨proof⟩

**lemma** *shows subexp-induct[consumes 1]*:

$(\bigwedge e. \text{subexp } e = \{\} \implies R e) \implies (\bigwedge e. (\bigwedge e'. e' \in \text{subexp } e \implies R e') \implies R e)$   
 $\implies (\bigwedge es. (\bigwedge e'. e' \in \text{subexps } es \implies R e') \implies Rs es) \implies (\forall e'. e' \in \text{subexp } e \implies R e') \wedge R e$   
**and** *subexps-induct*[consumes 1]:  
 $(\bigwedge es. \text{subexps } es = \{\} \implies Rs es) \implies (\bigwedge e. (\bigwedge e'. e' \in \text{subexp } e \implies R e') \implies R e)$   
 $\implies (\bigwedge es. (\bigwedge e'. e' \in \text{subexps } es \implies R e') \implies Rs es) \implies (\forall e'. e' \in \text{subexps } es \implies R e') \wedge Rs es$   
 $\langle \text{proof} \rangle$

### 1.8.6 Final expressions

**definition** *final* :: 'a exp  $\Rightarrow$  bool

**where**

*final* e  $\equiv$   $(\exists v. e = \text{Val } v) \vee (\exists a. e = \text{Throw } a)$

**definition** *finals*:: 'a exp list  $\Rightarrow$  bool

**where**

*finals* es  $\equiv$   $(\exists vs. es = \text{map Val } vs) \vee (\exists vs a es'. es = \text{map Val } vs @ \text{Throw } a \# es')$

**lemma** [*simp*]: *final*(Val v) $\langle \text{proof} \rangle$

**lemma** [*simp*]: *final*(throw e) =  $(\exists a. e = \text{addr } a) \langle \text{proof} \rangle$

**lemma** *finalE*:  $\llbracket \text{final } e; \bigwedge v. e = \text{Val } v \implies R; \bigwedge a. e = \text{Throw } a \implies R \rrbracket \implies R \langle \text{proof} \rangle$

**lemma** *final-fv*[*iff*]: *final* e  $\implies \text{fv } e = \{\}$   
 $\langle \text{proof} \rangle$

**lemma** *finalsE*:

$\llbracket \text{finals } es; \bigwedge vs. es = \text{map Val } vs \implies R; \bigwedge vs a es'. es = \text{map Val } vs @ \text{Throw } a \# es' \implies R \rrbracket \implies R \langle \text{proof} \rangle$

**lemma** [*iff*]: *finals* [] $\langle \text{proof} \rangle$

**lemma** [*iff*]: *finals* (Val v # es) = *finals* es $\langle \text{proof} \rangle$

**lemma** *finals-app-map*[*iff*]: *finals* (map Val vs @ es) = *finals* es $\langle \text{proof} \rangle$

**lemma** [*iff*]: *finals* (map Val vs) $\langle \text{proof} \rangle$

**lemma** [*iff*]: *finals* (throw e # es) =  $(\exists a. e = \text{addr } a) \langle \text{proof} \rangle$

**lemma** *not-finals-ConsI*:  $\neg \text{final } e \implies \neg \text{finals}(e \# es) \langle \text{proof} \rangle$

**lemma** *not-finals-ConsI2*:  $e = \text{Val } v \implies \neg \text{finals } es \implies \neg \text{finals}(e \# es) \langle \text{proof} \rangle$

**end**

## 1.9 Well-typedness of Jinja expressions

**theory** *WellType*

**imports** ../Common/Objects Expr

**begin**

**type-synonym**

*env* = *vname*  $\rightarrow$  *ty*

**inductive**

*WT* :: [*J-prog*, *env*, *expr* , *ty* ]  $\Rightarrow$  bool

$(\langle -, \vdash - \rangle \rightarrow [51, 51, 51] 50)$

**and** *WTs* :: [*J-prog*, *env*, *expr list*, *ty list*]  $\Rightarrow$  bool

$(\langle -, \vdash - [::] \rangle \rightarrow [51, 51, 51] 50)$

**for** *P* :: *J-prog*

**where**

*WTNew*:

$is\text{-}class\ P\ C \implies$   
 $P, E \vdash new\ C :: Class\ C$

|  $WTCast:$   
 $\llbracket P, E \vdash e :: Class\ D; is\text{-}class\ P\ C; P \vdash C \preceq^* D \vee P \vdash D \preceq^* C \rrbracket$   
 $\implies P, E \vdash Cast\ C\ e :: Class\ C$

|  $WTVal:$   
 $typeof\ v = Some\ T \implies$   
 $P, E \vdash Val\ v :: T$

|  $WTVar:$   
 $E\ V = Some\ T \implies$   
 $P, E \vdash Var\ V :: T$

|  $WTBinOpEq:$   
 $\llbracket P, E \vdash e_1 :: T_1; P, E \vdash e_2 :: T_2; P \vdash T_1 \leq T_2 \vee P \vdash T_2 \leq T_1 \rrbracket$   
 $\implies P, E \vdash e_1 \langle Eq \rangle e_2 :: Boolean$

|  $WTBinOpAdd:$   
 $\llbracket P, E \vdash e_1 :: Integer; P, E \vdash e_2 :: Integer \rrbracket$   
 $\implies P, E \vdash e_1 \langle Add \rangle e_2 :: Integer$

|  $WTLAss:$   
 $\llbracket E\ V = Some\ T; P, E \vdash e :: T'; P \vdash T' \leq T; V \neq this \rrbracket$   
 $\implies P, E \vdash V := e :: Void$

|  $WTFAcc:$   
 $\llbracket P, E \vdash e :: Class\ C; P \vdash C\ sees\ F, NonStatic:T\ in\ D \rrbracket$   
 $\implies P, E \vdash e \cdot F\{D\} :: T$

|  $WTSFAcc:$   
 $\llbracket P \vdash C\ sees\ F, Static:T\ in\ D \rrbracket$   
 $\implies P, E \vdash C \cdot_s F\{D\} :: T$

|  $WTFAss:$   
 $\llbracket P, E \vdash e_1 :: Class\ C; P \vdash C\ sees\ F, NonStatic:T\ in\ D; P, E \vdash e_2 :: T'; P \vdash T' \leq T \rrbracket$   
 $\implies P, E \vdash e_1 \cdot F\{D\} := e_2 :: Void$

|  $WTSFAss:$   
 $\llbracket P \vdash C\ sees\ F, Static:T\ in\ D; P, E \vdash e_2 :: T'; P \vdash T' \leq T \rrbracket$   
 $\implies P, E \vdash C \cdot_s F\{D\} := e_2 :: Void$

|  $WTCall:$   
 $\llbracket P, E \vdash e :: Class\ C; P \vdash C\ sees\ M, NonStatic:T_s \rightarrow T = (pns, body)\ in\ D;$   
 $P, E \vdash es\ [::]\ T_s'; P \vdash T_s' [\leq] T_s \rrbracket$   
 $\implies P, E \vdash e \cdot M(es) :: T$

|  $WTSCall:$   
 $\llbracket P \vdash C\ sees\ M, Static:T_s \rightarrow T = (pns, body)\ in\ D;$   
 $P, E \vdash es\ [::]\ T_s'; P \vdash T_s' [\leq] T_s; M \neq\ clinit \rrbracket$   
 $\implies P, E \vdash C \cdot_s M(es) :: T$

|  $WTBlock:$

$$\llbracket \text{is-type } P T; P, E(V \mapsto T) \vdash e :: T' \rrbracket \\ \implies P, E \vdash \{V:T; e\} :: T'$$
| *WTSeq*:
$$\llbracket P, E \vdash e_1 :: T_1; P, E \vdash e_2 :: T_2 \rrbracket \\ \implies P, E \vdash e_1 ;; e_2 :: T_2$$
| *WTCond*:
$$\llbracket P, E \vdash e :: \text{Boolean}; P, E \vdash e_1 :: T_1; P, E \vdash e_2 :: T_2; \\ P \vdash T_1 \leq T_2 \vee P \vdash T_2 \leq T_1; P \vdash T_1 \leq T_2 \longrightarrow T = T_2; P \vdash T_2 \leq T_1 \longrightarrow T = T_1 \rrbracket \\ \implies P, E \vdash \text{if } (e) e_1 \text{ else } e_2 :: T$$
| *WTWhile*:
$$\llbracket P, E \vdash e :: \text{Boolean}; P, E \vdash c :: T \rrbracket \\ \implies P, E \vdash \text{while } (e) c :: \text{Void}$$
| *WTThrow*:
$$P, E \vdash e :: \text{Class } C \implies \\ P, E \vdash \text{throw } e :: \text{Void}$$
| *WTTry*:
$$\llbracket P, E \vdash e_1 :: T; P, E(V \mapsto \text{Class } C) \vdash e_2 :: T; \text{is-class } P C \rrbracket \\ \implies P, E \vdash \text{try } e_1 \text{ catch}(C V) e_2 :: T$$

— well-typed expression lists

| *WTNil*:
$$P, E \vdash [] [::] []$$
| *WTCons*:
$$\llbracket P, E \vdash e :: T; P, E \vdash es [::] Ts \rrbracket \\ \implies P, E \vdash e \# es [::] T \# Ts$$

**lemma** *init-nwt* [simp]:  $\neg P, E \vdash \text{INIT } C (Cs, b) \leftarrow e :: T$   
 $\langle \text{proof} \rangle$

**lemma** *ri-nwt* [simp]:  $\neg P, E \vdash \text{RI}(C, e); Cs \leftarrow e' :: T$   
 $\langle \text{proof} \rangle$

**lemma** [iff]:  $(P, E \vdash [] [::] Ts) = (Ts = []) \langle \text{proof} \rangle$

**lemma** [iff]:  $(P, E \vdash e \# es [::] T \# Ts) = (P, E \vdash e :: T \wedge P, E \vdash es [::] Ts) \langle \text{proof} \rangle$

**lemma** [iff]:  $(P, E \vdash (e \# es) [::] Ts) =$   
 $(\exists U Us. Ts = U \# Us \wedge P, E \vdash e :: U \wedge P, E \vdash es [::] Us) \langle \text{proof} \rangle$

**lemma** [iff]:  $\bigwedge Ts. (P, E \vdash es_1 @ es_2 [::] Ts) =$   
 $(\exists Ts_1 Ts_2. Ts = Ts_1 @ Ts_2 \wedge P, E \vdash es_1 [::] Ts_1 \wedge P, E \vdash es_2 [::] Ts_2) \langle \text{proof} \rangle$

**lemma** [iff]:  $P, E \vdash \text{Val } v :: T = (\text{typeof } v = \text{Some } T) \langle \text{proof} \rangle$

**lemma** [iff]:  $P, E \vdash \text{Var } V :: T = (E V = \text{Some } T) \langle \text{proof} \rangle$

**lemma** [iff]:  $P, E \vdash e_1 ;; e_2 :: T_2 = (\exists T_1. P, E \vdash e_1 :: T_1 \wedge P, E \vdash e_2 :: T_2) \langle \text{proof} \rangle$

**lemma** [iff]:  $(P, E \vdash \{V:T; e\} :: T') = (\text{is-type } P T \wedge P, E(V \mapsto T) \vdash e :: T') \langle \text{proof} \rangle$

**lemma** *wt-env-mono*:

$$P, E \vdash e :: T \implies (\bigwedge E'. E \subseteq_m E' \implies P, E' \vdash e :: T) \text{ and} \\ P, E \vdash es [::] Ts \implies (\bigwedge E'. E \subseteq_m E' \implies P, E' \vdash es [::] Ts) \langle \text{proof} \rangle$$

**lemma** *WT-fv*:  $P, E \vdash e :: T \implies \text{fv } e \subseteq \text{dom } E$

and  $P, E \vdash es \text{ [::] } Ts \implies fvs\ es \subseteq dom\ E \langle proof \rangle$   
 lemma  $WT\text{-}nsub\text{-}RI: P, E \vdash e \text{ [::] } T \implies \neg sub\text{-}RI\ e$   
 and  $WTs\text{-}nsub\text{-}RIs: P, E \vdash es \text{ [::] } Ts \implies \neg sub\text{-}RIs\ es \langle proof \rangle$

## 1.10 Runtime Well-typedness

theory  $WellTypeRT$   
 imports  $WellType$   
 begin

inductive

$WTrt \text{ [::] } J\text{-}prog \implies heap \implies sheap \implies env \implies expr \implies ty \implies bool$   
 and  $WTrts \text{ [::] } J\text{-}prog \implies heap \implies sheap \implies env \implies expr\ list \implies ty\ list \implies bool$   
 and  $WTrt2 \text{ [::] } [J\text{-}prog, env, heap, sheap, expr, ty] \implies bool$   
 ( $\langle -, -, -, - \vdash - \text{ [::] } [51, 51, 51, 51] 50 \rangle$ )  
 and  $WTrts2 \text{ [::] } [J\text{-}prog, env, heap, sheap, expr\ list, ty\ list] \implies bool$   
 ( $\langle -, -, -, - \vdash - \text{ [::] } [51, 51, 51, 51] 50 \rangle$ )  
 for  $P \text{ [::] } J\text{-}prog$  and  $h \text{ [::] } heap$  and  $sh \text{ [::] } sheap$

where

$P, E, h, sh \vdash e \text{ [::] } T \equiv WTrt\ P\ h\ sh\ E\ e\ T$   
 $P, E, h, sh \vdash es \text{ [::] } Ts \equiv WTrts\ P\ h\ sh\ E\ es\ Ts$

|  $WTrtNew$ :  
 $is\text{-}class\ P\ C \implies$   
 $P, E, h, sh \vdash new\ C \text{ [::] } Class\ C$

|  $WTrtCast$ :  
 $[ [ P, E, h, sh \vdash e \text{ [::] } T; is\text{-}refT\ T; is\text{-}class\ P\ C ] ]$   
 $\implies P, E, h, sh \vdash Cast\ C\ e \text{ [::] } Class\ C$

|  $WTrtVal$ :  
 $typeof_h\ v = Some\ T \implies$   
 $P, E, h, sh \vdash Val\ v \text{ [::] } T$

|  $WTrtVar$ :  
 $E\ V = Some\ T \implies$   
 $P, E, h, sh \vdash Var\ V \text{ [::] } T$

|  $WTrtBinOpEq$ :  
 $[ [ P, E, h, sh \vdash e_1 \text{ [::] } T_1; P, E, h, sh \vdash e_2 \text{ [::] } T_2 ] ]$   
 $\implies P, E, h, sh \vdash e_1 \ll Eq \gg e_2 \text{ [::] } Boolean$

|  $WTrtBinOpAdd$ :  
 $[ [ P, E, h, sh \vdash e_1 \text{ [::] } Integer; P, E, h, sh \vdash e_2 \text{ [::] } Integer ] ]$   
 $\implies P, E, h, sh \vdash e_1 \ll Add \gg e_2 \text{ [::] } Integer$

|  $WTrtLAss$ :  
 $[ [ E\ V = Some\ T; P, E, h, sh \vdash e \text{ [::] } T'; P \vdash T' \leq T ] ]$   
 $\implies P, E, h, sh \vdash V := e \text{ [::] } Void$

|  $WTrtFAcc$ :  
 $[ [ P, E, h, sh \vdash e \text{ [::] } Class\ C; P \vdash C\ has\ F, NonStatic: T\ in\ D ] ] \implies$

$$P, E, h, sh \vdash e \cdot F\{D\} : T$$

| *WTrtFAccNT*:

$$P, E, h, sh \vdash e : NT \implies \\ P, E, h, sh \vdash e \cdot F\{D\} : T$$

| *WTrtSFAcc*:

$$\llbracket P \vdash C \text{ has } F, \text{Static}:T \text{ in } D \rrbracket \implies \\ P, E, h, sh \vdash C \cdot_s F\{D\} : T$$

| *WTrtFAss*:

$$\llbracket P, E, h, sh \vdash e_1 : \text{Class } C; P \vdash C \text{ has } F, \text{NonStatic}:T \text{ in } D; P, E, h, sh \vdash e_2 : T_2; P \vdash T_2 \leq T \rrbracket \\ \implies P, E, h, sh \vdash e_1 \cdot F\{D\} := e_2 : \text{Void}$$

| *WTrtFAssNT*:

$$\llbracket P, E, h, sh \vdash e_1 : NT; P, E, h, sh \vdash e_2 : T_2 \rrbracket \\ \implies P, E, h, sh \vdash e_1 \cdot F\{D\} := e_2 : \text{Void}$$

| *WTrtSFAss*:

$$\llbracket P, E, h, sh \vdash e_2 : T_2; P \vdash C \text{ has } F, \text{Static}:T \text{ in } D; P \vdash T_2 \leq T \rrbracket \\ \implies P, E, h, sh \vdash C \cdot_s F\{D\} := e_2 : \text{Void}$$

| *WTrtCall*:

$$\llbracket P, E, h, sh \vdash e : \text{Class } C; P \vdash C \text{ sees } M, \text{NonStatic}:Ts \rightarrow T = (pns, body) \text{ in } D; \\ P, E, h, sh \vdash es \text{ [:] } Ts'; P \vdash Ts' \llbracket \leq \rrbracket Ts \rrbracket \\ \implies P, E, h, sh \vdash e \cdot M(es) : T$$

| *WTrtCallNT*:

$$\llbracket P, E, h, sh \vdash e : NT; P, E, h, sh \vdash es \text{ [:] } Ts \rrbracket \\ \implies P, E, h, sh \vdash e \cdot M(es) : T$$

| *WTrtSCall*:

$$\llbracket P \vdash C \text{ sees } M, \text{Static}:Ts \rightarrow T = (pns, body) \text{ in } D; \\ P, E, h, sh \vdash es \text{ [:] } Ts'; P \vdash Ts' \llbracket \leq \rrbracket Ts; \\ M = \text{clinit} \longrightarrow sh D = \llbracket (sfs, Processing) \rrbracket \wedge es = \text{map Val } vs \rrbracket \\ \implies P, E, h, sh \vdash C \cdot_s M(es) : T$$

| *WTrtBlock*:

$$P, E(V \mapsto T), h, sh \vdash e : T' \implies \\ P, E, h, sh \vdash \{V:T; e\} : T'$$

| *WTrtSeq*:

$$\llbracket P, E, h, sh \vdash e_1 : T_1; P, E, h, sh \vdash e_2 : T_2 \rrbracket \\ \implies P, E, h, sh \vdash e_1 ; e_2 : T_2$$

| *WTrtCond*:

$$\llbracket P, E, h, sh \vdash e : \text{Boolean}; P, E, h, sh \vdash e_1 : T_1; P, E, h, sh \vdash e_2 : T_2; \\ P \vdash T_1 \leq T_2 \vee P \vdash T_2 \leq T_1; P \vdash T_1 \leq T_2 \longrightarrow T = T_2; P \vdash T_2 \leq T_1 \longrightarrow T = T_1 \rrbracket \\ \implies P, E, h, sh \vdash \text{if } (e) \ e_1 \ \text{else } e_2 : T$$

| *WTrtWhile*:

$$\llbracket P, E, h, sh \vdash e : \text{Boolean}; P, E, h, sh \vdash c : T \rrbracket \\ \implies P, E, h, sh \vdash \text{while}(e) \ c : \text{Void}$$

| *WTrtThrow*:

$$\llbracket P, E, h, sh \vdash e : T_r; \text{is-refT } T_r \rrbracket \implies \\ P, E, h, sh \vdash \text{throw } e : T$$

| *WTrtTry*:

$$\llbracket P, E, h, sh \vdash e_1 : T_1; P, E(V \mapsto \text{Class } C), h, sh \vdash e_2 : T_2; P \vdash T_1 \leq T_2 \rrbracket \\ \implies P, E, h, sh \vdash \text{try } e_1 \text{ catch}(C \ V) \ e_2 : T_2$$

| *WTrtInit*:

$$\llbracket P, E, h, sh \vdash e : T; \forall C' \in \text{set } (C \# Cs). \text{is-class } P \ C'; \neg \text{sub-RI } e; \\ \forall C' \in \text{set } (tl \ Cs). \exists \text{sfs. sh } C' = \llbracket (sfs, \text{Processing}) \rrbracket; \\ b \longrightarrow (\forall C' \in \text{set } Cs. \exists \text{sfs. sh } C' = \llbracket (sfs, \text{Processing}) \rrbracket); \\ \text{distinct } Cs; \text{supercls-lst } P \ Cs \rrbracket \\ \implies P, E, h, sh \vdash \text{INIT } C \ (Cs, b) \leftarrow e : T$$

| *WTrtRI*:

$$\llbracket P, E, h, sh \vdash e : T; P, E, h, sh \vdash e' : T'; \forall C' \in \text{set } (C \# Cs). \text{is-class } P \ C'; \neg \text{sub-RI } e'; \\ \forall C' \in \text{set } (C \# Cs). \text{not-init } C' \ e; \\ \forall C' \in \text{set } Cs. \exists \text{sfs. sh } C' = \llbracket (sfs, \text{Processing}) \rrbracket; \\ \exists \text{sfs. sh } C = \llbracket (sfs, \text{Processing}) \rrbracket \vee (sh \ C = \llbracket (sfs, \text{Error}) \rrbracket) \wedge e = \text{THROW NoClassDefFoundError}; \\ \text{distinct } (C \# Cs); \text{supercls-lst } P \ (C \# Cs) \rrbracket \\ \implies P, E, h, sh \vdash \text{RI}(C, e); Cs \leftarrow e' : T'$$

— well-typed expression lists

| *WTrtNil*:

$$P, E, h, sh \vdash [] \ [:] \ []$$

| *WTrtCons*:

$$\llbracket P, E, h, sh \vdash e : T; P, E, h, sh \vdash es \ [:] \ Ts \rrbracket \\ \implies P, E, h, sh \vdash e \# es \ [:] \ T \# Ts$$

### 1.10.1 Easy consequences

**lemma** [*iff*]:  $(P, E, h, sh \vdash [] \ [:] \ Ts) = (Ts = []) \langle \text{proof} \rangle$

**lemma** [*iff*]:  $(P, E, h, sh \vdash e \# es \ [:] \ T \# Ts) = (P, E, h, sh \vdash e : T \wedge P, E, h, sh \vdash es \ [:] \ Ts) \langle \text{proof} \rangle$

**lemma** [*iff*]:  $(P, E, h, sh \vdash (e \# es) \ [:] \ Ts) = \\ (\exists U \ Us. Ts = U \# Us \wedge P, E, h, sh \vdash e : U \wedge P, E, h, sh \vdash es \ [:] \ Us) \langle \text{proof} \rangle$

**lemma** [*simp*]:  $\forall Ts. (P, E, h, sh \vdash es_1 \ @ \ es_2 \ [:] \ Ts) = \\ (\exists Ts_1 \ Ts_2. Ts = Ts_1 \ @ \ Ts_2 \wedge P, E, h, sh \vdash es_1 \ [:] \ Ts_1 \ \& \ P, E, h, sh \vdash es_2 \ [:] \ Ts_2) \langle \text{proof} \rangle$

**lemma** [*iff*]:  $P, E, h, sh \vdash \text{Val } v : T = (\text{typeof}_h \ v = \text{Some } T) \langle \text{proof} \rangle$

**lemma** [*iff*]:  $P, E, h, sh \vdash \text{Var } v : T = (E \ v = \text{Some } T) \langle \text{proof} \rangle$

**lemma** [*iff*]:  $P, E, h, sh \vdash e_1; e_2 : T_2 = (\exists T_1. P, E, h, sh \vdash e_1 : T_1 \wedge P, E, h, sh \vdash e_2 : T_2) \langle \text{proof} \rangle$

**lemma** [*iff*]:  $P, E, h, sh \vdash \{V:T; e\} : T' = (P, E(V \mapsto T), h, sh \vdash e : T') \langle \text{proof} \rangle$

### 1.10.2 Some interesting lemmas

**lemma** *WTrts-Val[simp]*:

$$\bigwedge Ts. (P, E, h, sh \vdash \text{map Val } vs \ [:] \ Ts) = (\text{map } (\text{typeof}_h) \ vs = \text{map Some } Ts) \langle \text{proof} \rangle$$

**lemma** *WTrts-same-length*:  $\bigwedge Ts. P, E, h, sh \vdash es \ [:] \ Ts \implies \text{length } es = \text{length } Ts \langle \text{proof} \rangle$

**lemma** *WTrt-env-mono*:



$P, E, h, sh \vdash e : T \implies (\bigwedge E'. E \subseteq_m E' \implies P, E', h, sh \vdash e : T)$  **and**  
 $P, E, h, sh \vdash es \text{ [:] } Ts \implies (\bigwedge E'. E \subseteq_m E' \implies P, E', h, sh \vdash es \text{ [:] } Ts)$  *(proof)*

**lemma** *WTrt-hext-mono*:  $P, E, h, sh \vdash e : T \implies h \trianglelefteq h' \implies P, E, h', sh \vdash e : T$

**and** *WTrts-hext-mono*:  $P, E, h, sh \vdash es \text{ [:] } Ts \implies h \trianglelefteq h' \implies P, E, h', sh \vdash es \text{ [:] } Ts$  *(proof)*

**lemma** *WTrt-shext-mono*:  $P, E, h, sh \vdash e : T \implies sh \trianglelefteq_s sh' \implies \neg \text{sub-RI } e \implies P, E, h, sh' \vdash e : T$

**and** *WTrts-shext-mono*:  $P, E, h, sh \vdash es \text{ [:] } Ts \implies sh \trianglelefteq_s sh' \implies \neg \text{sub-RIs } es \implies P, E, h, sh' \vdash es \text{ [:] } Ts$  *(proof)*

**lemma** *WTrt-hext-shext-mono*:  $P, E, h, sh \vdash e : T$

$\implies h \trianglelefteq h' \implies sh \trianglelefteq_s sh' \implies \neg \text{sub-RI } e \implies P, E, h', sh' \vdash e : T$

*(proof)*

**lemma** *WTrts-hext-shext-mono*:  $P, E, h, sh \vdash es \text{ [:] } Ts$

$\implies h \trianglelefteq h' \implies sh \trianglelefteq_s sh' \implies \neg \text{sub-RIs } es \implies P, E, h', sh' \vdash es \text{ [:] } Ts$

*(proof)*

**lemma** *WT-implies-WTrt*:  $P, E \vdash e :: T \implies P, E, h, sh \vdash e : T$

**and** *WTs-implies-WTrts*:  $P, E \vdash es \text{ [:] } Ts \implies P, E, h, sh \vdash es \text{ [:] } Ts$  *(proof)*

**end**

## 1.11 Program State

**theory** *State* **imports** *../Common/Exceptions* **begin**

**type-synonym**

$locals = vname \rightarrow val$  — local vars, incl. params and “this”

**type-synonym**

$state = heap \times locals \times sheap$

**definition**  $hp :: state \Rightarrow heap$

**where**

$hp \equiv fst$

**definition**  $lcl :: state \Rightarrow locals$

**where**

$lcl \equiv fst \circ snd$

**definition**  $shp :: state \Rightarrow sheap$

**where**

$shp \equiv snd \circ snd$

**end**

## 1.12 System Classes

**theory** *SystemClasses*

**imports** *Decl Exceptions*

**begin**

This theory provides definitions for the *Object* class, and the system exceptions.

**definition**  $ObjectC :: 'm \text{ cdecl}$

**where**

$ObjectC \equiv (Object, (undefined, [], []))$

**definition** *NullPointerC* :: 'm cdecl

**where**

*NullPointerC*  $\equiv$  (*NullPointer*, (*Object*, [], []))

**definition** *ClassCastC* :: 'm cdecl

**where**

*ClassCastC*  $\equiv$  (*ClassCast*, (*Object*, [], []))

**definition** *OutOfMemoryC* :: 'm cdecl

**where**

*OutOfMemoryC*  $\equiv$  (*OutOfMemory*, (*Object*, [], []))

**definition** *NoClassDefFoundC* :: 'm cdecl

**where**

*NoClassDefFoundC*  $\equiv$  (*NoClassDefFoundError*, (*Object*, [], []))

**definition** *IncompatibleClassChangeC* :: 'm cdecl

**where**

*IncompatibleClassChangeC*  $\equiv$  (*IncompatibleClassChangeError*, (*Object*, [], []))

**definition** *NoSuchFieldC* :: 'm cdecl

**where**

*NoSuchFieldC*  $\equiv$  (*NoSuchFieldError*, (*Object*, [], []))

**definition** *NoSuchMethodC* :: 'm cdecl

**where**

*NoSuchMethodC*  $\equiv$  (*NoSuchMethodError*, (*Object*, [], []))

**definition** *SystemClasses* :: 'm cdecl list

**where**

*SystemClasses*  $\equiv$  [*ObjectC*, *NullPointerC*, *ClassCastC*, *OutOfMemoryC*, *NoClassDefFoundC*,  
*IncompatibleClassChangeC*, *NoSuchFieldC*, *NoSuchMethodC*]

**end**

## 1.13 Generic Well-formedness of programs

**theory** *WellForm* **imports** *TypeRel* *SystemClasses* **begin**

This theory defines global well-formedness conditions for programs but does not look inside method bodies. Hence it works for both Jinja and JVM programs. Well-typing of expressions is defined elsewhere (in theory *WellType*).

Because Jinja does not have method overloading, its policy for method overriding is the classical one: *covariant in the result type but contravariant in the argument types*. This means the result type of the overriding method becomes more specific, the argument types become more general.

**type-synonym** 'm wf-mdecl-test = 'm prog  $\Rightarrow$  cname  $\Rightarrow$  'm mdecl  $\Rightarrow$  bool

**definition** *wf-fdecl* :: 'm prog  $\Rightarrow$  fdecl  $\Rightarrow$  bool

**where**

*wf-fdecl* *P*  $\equiv$   $\lambda(F, b, T). \text{is-type } P \ T$

**definition** *wf-mdecl* :: 'm wf-mdecl-test  $\Rightarrow$  'm wf-mdecl-test

**where**

$wf\text{-mdecl } wf\text{-md } P \ C \equiv \lambda(M,b,Ts,T,m).$   
 $(\forall T \in set \ Ts. \ is\text{-type } P \ T) \wedge \ is\text{-type } P \ T \wedge \ wf\text{-md } P \ C \ (M,b,Ts,T,m)$

**definition**  $wf\text{-clinit} :: 'm \ mdecl \ list \Rightarrow \ bool$  **where**

$wf\text{-clinit } ms = (\exists m. (clinit,Static,[],Void,m) \in set \ ms)$

**definition**  $wf\text{-cdecl} :: 'm \ wf\text{-mdecl}\text{-test} \Rightarrow 'm \ prog \Rightarrow 'm \ cdecl \Rightarrow \ bool$

**where**

$wf\text{-cdecl } wf\text{-md } P \equiv \lambda(C,(D,fs,ms)).$   
 $(\forall f \in set \ fs. \ wf\text{-fdecl } P \ f) \wedge \ distinct\text{-fst } fs \wedge$   
 $(\forall m \in set \ ms. \ wf\text{-mdecl } wf\text{-md } P \ C \ m) \wedge \ distinct\text{-fst } ms \wedge$   
 $(C \neq Object \longrightarrow$   
 $\ is\text{-class } P \ D \wedge \neg P \vdash D \preceq^* C \wedge$   
 $(\forall (M,b,Ts,T,m) \in set \ ms.$   
 $\ \forall D' \ b' \ Ts' \ T' \ m'. \ P \vdash D \ sees \ M,b':Ts' \rightarrow T' = m' \ in \ D' \longrightarrow$   
 $\ \ b = b' \wedge P \vdash Ts' [\leq] Ts \wedge P \vdash T \leq T') \wedge$   
 $wf\text{-clinit } ms$

**definition**  $wf\text{-syscls} :: 'm \ prog \Rightarrow \ bool$

**where**

$wf\text{-syscls } P \equiv \{Object\} \cup \ sys\text{-xcpts} \subseteq \ set(map \ fst \ P)$

**definition**  $wf\text{-prog} :: 'm \ wf\text{-mdecl}\text{-test} \Rightarrow 'm \ prog \Rightarrow \ bool$

**where**

$wf\text{-prog } wf\text{-md } P \equiv \ wf\text{-syscls } P \wedge (\forall c \in \ set \ P. \ wf\text{-cdecl } wf\text{-md } P \ c) \wedge \ distinct\text{-fst } P$

### 1.13.1 Well-formedness lemmas

**lemma**  $class\text{-wf}$ :

$\llbracket class \ P \ C = Some \ c; \ wf\text{-prog } wf\text{-md } P \rrbracket \Longrightarrow \ wf\text{-cdecl } wf\text{-md } P \ (C,c) \langle proof \rangle$

**lemma**  $class\text{-Object}$  [simp]:

$wf\text{-prog } wf\text{-md } P \Longrightarrow \exists C \ fs \ ms. \ class \ P \ Object = Some \ (C,fs,ms) \langle proof \rangle$

**lemma**  $is\text{-class}\text{-Object}$  [simp]:

$wf\text{-prog } wf\text{-md } P \Longrightarrow \ is\text{-class } P \ Object \langle proof \rangle$

**lemma**  $is\text{-class}\text{-supclass}$ :

**assumes**  $wf$ :  $wf\text{-prog } wf\text{-md } P$  **and**  $sub$ :  $P \vdash C \preceq^* D$

**shows**  $is\text{-class } P \ C \Longrightarrow \ is\text{-class } P \ D \langle proof \rangle$

**lemma**  $is\text{-class}\text{-xcpt}$ :

$\llbracket C \in \ sys\text{-xcpts}; \ wf\text{-prog } wf\text{-md } P \rrbracket \Longrightarrow \ is\text{-class } P \ C \langle proof \rangle$

**lemma**  $subcls1\text{-wfD}$ :

**assumes**  $sub1$ :  $P \vdash C \prec^1 D$  **and**  $wf$ :  $wf\text{-prog } wf\text{-md } P$

**shows**  $D \neq C \wedge (D,C) \notin (subcls1 \ P)^+ \langle proof \rangle$

**lemma**  $wf\text{-cdecl}\text{-supD}$ :

$\llbracket wf\text{-cdecl } wf\text{-md } P \ (C,D,r); \ C \neq Object \rrbracket \Longrightarrow \ is\text{-class } P \ D \langle proof \rangle$

**lemma**  $subcls\text{-asym}$ :

$\llbracket wf\text{-prog } wf\text{-md } P; \ (C,D) \in (subcls1 \ P)^+ \rrbracket \Longrightarrow \ (D,C) \notin (subcls1 \ P)^+ \langle proof \rangle$

**lemma**  $subcls\text{-irrefl}$ :

$\llbracket \text{wf-prog wf-md } P; (C,D) \in (\text{subcls1 } P)^+ \rrbracket \Longrightarrow C \neq D \langle \text{proof} \rangle$

**lemma** *acyclic-subcls1*:

$\text{wf-prog wf-md } P \Longrightarrow \text{acyclic } (\text{subcls1 } P) \langle \text{proof} \rangle$

**lemma** *wf-subcls1*:

$\text{wf-prog wf-md } P \Longrightarrow \text{wf } ((\text{subcls1 } P)^{-1}) \langle \text{proof} \rangle$

**lemma** *single-valued-subcls1*:

$\text{wf-prog wf-md } G \Longrightarrow \text{single-valued } (\text{subcls1 } G) \langle \text{proof} \rangle$

**lemma** *subcls-induct*:

$\llbracket \text{wf-prog wf-md } P; \bigwedge C. \forall D. (C,D) \in (\text{subcls1 } P)^+ \longrightarrow Q D \Longrightarrow Q C \rrbracket \Longrightarrow Q C \langle \text{proof} \rangle$

**lemma** *subcls1-induct-aux*:

**assumes** *is-class*  $P C$  **and** *wf*:  $\text{wf-prog wf-md } P$  **and** *QObj*:  $Q \text{ Object}$

**shows**

$\llbracket \bigwedge C D \text{ fs } ms.$

$\llbracket C \neq \text{Object}; \text{is-class } P C; \text{class } P C = \text{Some } (D, \text{fs}, ms) \wedge$

$\text{wf-cdecl wf-md } P (C, D, \text{fs}, ms) \wedge P \vdash C \prec^1 D \wedge \text{is-class } P D \wedge Q D \rrbracket \Longrightarrow Q C$

$\Longrightarrow Q C \langle \text{proof} \rangle$

**lemma** *subcls1-induct* [*consumes 2, case-names Object Subcls*]:

$\llbracket \text{wf-prog wf-md } P; \text{is-class } P C; Q \text{ Object};$

$\bigwedge C D. \llbracket C \neq \text{Object}; P \vdash C \prec^1 D; \text{is-class } P D; Q D \rrbracket \Longrightarrow Q C$

$\Longrightarrow Q C \langle \text{proof} \rangle$

**lemma** *subcls-C-Object*:

**assumes** *class*:  $\text{is-class } P C$  **and** *wf*:  $\text{wf-prog wf-md } P$

**shows**  $P \vdash C \preceq^* \text{Object} \langle \text{proof} \rangle$

**lemma** *is-type-pTs*:

**assumes**  $\text{wf-prog wf-md } P$  **and**  $(C, S, \text{fs}, ms) \in \text{set } P$  **and**  $(M, b, Ts, T, m) \in \text{set } ms$

**shows**  $\text{set } Ts \subseteq \text{types } P \langle \text{proof} \rangle$

**lemma** *wf-supercls-distinct-app*:

**assumes** *wf*:  $\text{wf-prog wf-md } P$

**and** *nObj*:  $C \neq \text{Object}$  **and** *cls*:  $\text{class } P C = \llbracket (D, \text{fs}, ms) \rrbracket$

**and** *super*:  $\text{supercls-1st } P (C \# Cs)$  **and** *dist*:  $\text{distinct } (C \# Cs)$

**shows**  $\text{distinct } (D \# C \# Cs)$

$\langle \text{proof} \rangle$

### 1.13.2 Well-formedness and method lookup

**lemma** *sees-wf-mdecl*:

**assumes** *wf*:  $\text{wf-prog wf-md } P$  **and** *sees*:  $P \vdash C \text{ sees } M, b: Ts \rightarrow T = m \text{ in } D$

**shows**  $\text{wf-mdecl wf-md } P D (M, b, Ts, T, m) \langle \text{proof} \rangle$

**lemma** *sees-method-mono* [*rule-format (no-asm)*]:

**assumes** *sub*:  $P \vdash C' \preceq^* C$  **and** *wf*:  $\text{wf-prog wf-md } P$

**shows**  $\forall D b Ts T m. P \vdash C \text{ sees } M, b: Ts \rightarrow T = m \text{ in } D \longrightarrow$

$(\exists D' Ts' T' m'. P \vdash C' \text{ sees } M, b: Ts' \rightarrow T' = m' \text{ in } D' \wedge P \vdash Ts \llbracket \leq \rrbracket Ts' \wedge P \vdash T' \leq T) \langle \text{proof} \rangle$

**lemma** *sees-method-mono2*:

$\llbracket P \vdash C' \preceq^* C; \text{wf-prog wf-md } P;$

$P \vdash C \text{ sees } M, b: Ts \rightarrow T = m \text{ in } D; P \vdash C' \text{ sees } M, b': Ts' \rightarrow T' = m' \text{ in } D' \rrbracket$

$\implies b = b' \wedge P \vdash Ts \leq Ts' \wedge P \vdash T' \leq T \langle \text{proof} \rangle$

**lemma** *mdecls-visible*:

**assumes** *wf*: *wf-prog wf-md P* **and** *class*: *is-class P C*

**shows**  $\bigwedge D fs ms. \text{class } P C = \text{Some}(D, fs, ms)$

$\implies \exists Mm. P \vdash C \text{ sees-methods } Mm \wedge (\forall (M, b, Ts, T, m) \in \text{set } ms. Mm M = \text{Some}((b, Ts, T, m), C)) \langle \text{proof} \rangle$

**lemma** *mdecl-visible*:

**assumes** *wf*: *wf-prog wf-md P* **and** *C*:  $(C, S, fs, ms) \in \text{set } P$  **and** *m*:  $(M, b, Ts, T, m) \in \text{set } ms$

**shows**  $P \vdash C \text{ sees } M, b: Ts \rightarrow T = m \text{ in } C \langle \text{proof} \rangle$

**lemma** *Call-lemma*:

**assumes** *sees*:  $P \vdash C \text{ sees } M, b: Ts \rightarrow T = m \text{ in } D$  **and** *sub*:  $P \vdash C' \preceq^* C$  **and** *wf*: *wf-prog wf-md P*

**shows**  $\exists D' Ts' T' m'.$

$P \vdash C' \text{ sees } M, b: Ts' \rightarrow T' = m' \text{ in } D' \wedge P \vdash Ts \leq Ts' \wedge P \vdash T' \leq T \wedge P \vdash C' \preceq^* D'$

$\wedge \text{is-type } P T' \wedge (\forall T \in \text{set } Ts'. \text{is-type } P T) \wedge \text{wf-md } P D' (M, b, Ts', T', m') \langle \text{proof} \rangle$

**lemma** *wf-prog-lift*:

**assumes** *wf*: *wf-prog*  $(\lambda P C bd. A P C bd) P$

**and** *rule*:

$\bigwedge \text{wf-md } C M b Ts C T m bd.$

*wf-prog wf-md P*  $\implies$

$P \vdash C \text{ sees } M, b: Ts \rightarrow T = m \text{ in } C \implies$

*set Ts*  $\subseteq$  *types P*  $\implies$

*bd* =  $(M, b, Ts, T, m) \implies$

*A P C bd*  $\implies$

*B P C bd*

**shows** *wf-prog*  $(\lambda P C bd. B P C bd) P \langle \text{proof} \rangle$

**lemma** *wf-sees-clinit*:

**assumes** *wf*: *wf-prog wf-md P* **and** *ex*: *class P C = Some a*

**shows**  $\exists m. P \vdash C \text{ sees } \text{clinit}, \text{Static}: [] \rightarrow \text{Void} = m \text{ in } C$

$\langle \text{proof} \rangle$

**lemma** *wf-sees-clinit1*:

**assumes** *wf*: *wf-prog wf-md P* **and** *ex*: *class P C = Some a*

**and**  $P \vdash C \text{ sees } \text{clinit}, b: Ts \rightarrow T = m \text{ in } D$

**shows**  $b = \text{Static} \wedge Ts = [] \wedge T = \text{Void} \wedge D = C$

$\langle \text{proof} \rangle$

**lemma** *wf-NonStatic-nclinit*:

**assumes** *wf*: *wf-prog wf-md P* **and** *meth*:  $P \vdash C \text{ sees } M, \text{NonStatic}: Ts \rightarrow T = (mxs, mxl, ins, xt) \text{ in } D$

**shows**  $M \neq \text{clinit}$

$\langle \text{proof} \rangle$

### 1.13.3 Well-formedness and field lookup

**lemma** *wf-Fields-Ex*:

**assumes** *wf*: *wf-prog wf-md P* **and** *is-class P C*

**shows**  $\exists FDTs. P \vdash C \text{ has-fields } FDTs \langle \text{proof} \rangle$

**lemma** *has-fields-types*:

$\llbracket P \vdash C \text{ has-fields } FDTs; (FD, b, T) \in \text{set } FDTs; \text{wf-prog wf-md P} \rrbracket \implies \text{is-type } P T \langle \text{proof} \rangle$

**lemma** *sees-field-is-type*:

$\llbracket P \vdash C \text{ sees } F, b: T \text{ in } D; \text{wf-prog wf-md P} \rrbracket \implies \text{is-type } P T \langle \text{proof} \rangle$

**lemma** *wf-syscls*:

*set SystemClasses*  $\subseteq$  *set P*  $\implies$  *wf-syscls P*⟨*proof*⟩

### 1.13.4 Well-formedness and subclassing

**lemma** *wf-subcls-nCls*:

**assumes** *wf*: *wf-prog wf-md P* **and** *ns*:  $\neg$  *is-class P C*

**shows**  $\llbracket P \vdash D \preceq^* D'; D \neq C \rrbracket \implies D' \neq C$

⟨*proof*⟩

**lemma** *wf-subcls-nCls'*:

**assumes** *wf*: *wf-prog wf-md P* **and** *ns*:  $\neg$  *is-class P C<sub>0</sub>*

**shows**  $\bigwedge cd D'. cd \in \text{set } P \implies \neg P \vdash \text{fst } cd \preceq^* C_0$

⟨*proof*⟩

**lemma** *wf-nclass-nsub*:

$\llbracket \text{wf-prog wf-md } P; \text{is-class } P C; \neg \text{is-class } P C' \rrbracket \implies \neg P \vdash C \preceq^* C'$

⟨*proof*⟩

**lemma** *wf-sys-xcpt-nsub-Start*:

**assumes** *wf*: *wf-prog wf-md P* **and** *ns*:  $\neg$  *is-class P Start* **and** *sx*:  $C \in \text{sys-xcpts}$

**shows**  $\neg P \vdash C \preceq^* \text{Start}$

⟨*proof*⟩

**end**

## 1.14 Weak well-formedness of Jinja programs

**theory** *WWellForm* **imports** *../Common/WellForm Expr* **begin**

**definition** *wwf-J-mdecl* :: *J-prog*  $\Rightarrow$  *cname*  $\Rightarrow$  *J-mb mdecl*  $\Rightarrow$  *bool*

**where**

*wwf-J-mdecl P C*  $\equiv$   $\lambda(M, b, Ts, T, (pns, body)).$

*length Ts = length pns*  $\wedge$  *distinct pns*  $\wedge$   $\neg$  *sub-RI body*  $\wedge$

(*case b of*

*NonStatic*  $\Rightarrow$  *this*  $\notin$  *set pns*  $\wedge$  *fv body*  $\subseteq$   $\{\text{this}\} \cup \text{set } pns$

| *Static*  $\Rightarrow$  *fv body*  $\subseteq$  *set pns*)

**lemma** *wwf-J-mdecl-NonStatic[simp]*:

*wwf-J-mdecl P C (M, NonStatic, Ts, T, pns, body) =*

*(length Ts = length pns*  $\wedge$  *distinct pns*  $\wedge$   $\neg$  *sub-RI body*  $\wedge$  *this*  $\notin$  *set pns*  $\wedge$  *fv body*  $\subseteq$   $\{\text{this}\} \cup \text{set } pns)$ ⟨*proof*⟩

**lemma** *wwf-J-mdecl-Static[simp]*:

*wwf-J-mdecl P C (M, Static, Ts, T, pns, body) =*

*(length Ts = length pns*  $\wedge$  *distinct pns*  $\wedge$   $\neg$  *sub-RI body*  $\wedge$  *fv body*  $\subseteq$  *set pns*)⟨*proof*⟩

**abbreviation**

*wwf-J-prog* :: *J-prog*  $\Rightarrow$  *bool* **where**

*wwf-J-prog*  $\equiv$  *wf-prog wwf-J-mdecl*

**lemma** *sees-wwf-nsub-RI*:

$\llbracket \text{wwf-J-prog } P; P \vdash C \text{ sees } M, b : Ts \rightarrow T = (pns, body) \text{ in } D \rrbracket \implies \neg \text{sub-RI body}$ ⟨*proof*⟩

**end**

## 1.15 Big Step Semantics

**theory** *BigStep* **imports** *Expr State WWellForm* **begin**

**inductive**

*eval* :: *J-prog*  $\Rightarrow$  *expr*  $\Rightarrow$  *state*  $\Rightarrow$  *expr*  $\Rightarrow$  *state*  $\Rightarrow$  *bool*  
 $(\langle \cdot \vdash ((1\langle \cdot, / \cdot \rangle) \Rightarrow / (1\langle \cdot, / \cdot \rangle)) \rangle [51, 0, 0, 0, 0] 81)$   
**and** *evals* :: *J-prog*  $\Rightarrow$  *expr list*  $\Rightarrow$  *state*  $\Rightarrow$  *expr list*  $\Rightarrow$  *state*  $\Rightarrow$  *bool*  
 $(\langle \cdot \vdash ((1\langle \cdot, / \cdot \rangle) [\Rightarrow] / (1\langle \cdot, / \cdot \rangle)) \rangle [51, 0, 0, 0, 0] 81)$   
**for** *P* :: *J-prog*

**where**

*New*:

$\llbracket sh\ C = Some\ (sfs,\ Done); new\text{-}Addr\ h = Some\ a;$   
 $P \vdash C\ has\text{-}fields\ FDTs; h' = h(a \mapsto blank\ P\ C) \rrbracket$   
 $\Longrightarrow P \vdash \langle new\ C, (h, l, sh) \rangle \Rightarrow \langle addr\ a, (h', l, sh) \rangle$

| *NewFail*:

$\llbracket sh\ C = Some\ (sfs,\ Done); new\text{-}Addr\ h = None; is\text{-}class\ P\ C \rrbracket \Longrightarrow$   
 $P \vdash \langle new\ C, (h, l, sh) \rangle \Rightarrow \langle THROW\ OutOfMemory, (h, l, sh) \rangle$

| *NewInit*:

$\llbracket \# sfs.\ sh\ C = Some\ (sfs,\ Done); P \vdash \langle INIT\ C\ ([C], False) \leftarrow unit, (h, l, sh) \rangle \Rightarrow \langle Val\ v', (h', l', sh') \rangle;$   
 $new\text{-}Addr\ h' = Some\ a; P \vdash C\ has\text{-}fields\ FDTs; h'' = h'(a \mapsto blank\ P\ C) \rrbracket$   
 $\Longrightarrow P \vdash \langle new\ C, (h, l, sh) \rangle \Rightarrow \langle addr\ a, (h'', l', sh') \rangle$

| *NewInitOOM*:

$\llbracket \# sfs.\ sh\ C = Some\ (sfs,\ Done); P \vdash \langle INIT\ C\ ([C], False) \leftarrow unit, (h, l, sh) \rangle \Rightarrow \langle Val\ v', (h', l', sh') \rangle;$   
 $new\text{-}Addr\ h' = None; is\text{-}class\ P\ C \rrbracket$   
 $\Longrightarrow P \vdash \langle new\ C, (h, l, sh) \rangle \Rightarrow \langle THROW\ OutOfMemory, (h', l', sh') \rangle$

| *NewInitThrow*:

$\llbracket \# sfs.\ sh\ C = Some\ (sfs,\ Done); P \vdash \langle INIT\ C\ ([C], False) \leftarrow unit, (h, l, sh) \rangle \Rightarrow \langle throw\ a, s' \rangle;$   
 $is\text{-}class\ P\ C \rrbracket$   
 $\Longrightarrow P \vdash \langle new\ C, (h, l, sh) \rangle \Rightarrow \langle throw\ a, s' \rangle$

| *Cast*:

$\llbracket P \vdash \langle e, s_0 \rangle \Rightarrow \langle addr\ a, (h, l, sh) \rangle; h\ a = Some(D, fs); P \vdash D \preceq^* C \rrbracket$   
 $\Longrightarrow P \vdash \langle Cast\ C\ e, s_0 \rangle \Rightarrow \langle addr\ a, (h, l, sh) \rangle$

| *CastNull*:

$P \vdash \langle e, s_0 \rangle \Rightarrow \langle null, s_1 \rangle \Longrightarrow$   
 $P \vdash \langle Cast\ C\ e, s_0 \rangle \Rightarrow \langle null, s_1 \rangle$

| *CastFail*:

$\llbracket P \vdash \langle e, s_0 \rangle \Rightarrow \langle addr\ a, (h, l, sh) \rangle; h\ a = Some(D, fs); \neg P \vdash D \preceq^* C \rrbracket$   
 $\Longrightarrow P \vdash \langle Cast\ C\ e, s_0 \rangle \Rightarrow \langle THROW\ ClassCast, (h, l, sh) \rangle$

| *CastThrow*:

$P \vdash \langle e, s_0 \rangle \Rightarrow \langle throw\ e', s_1 \rangle \Longrightarrow$   
 $P \vdash \langle Cast\ C\ e, s_0 \rangle \Rightarrow \langle throw\ e', s_1 \rangle$

| *Val*:

$P \vdash \langle Val\ v, s \rangle \Rightarrow \langle Val\ v, s \rangle$

- | *BinOp*:  

$$\begin{aligned} & \llbracket P \vdash \langle e_1, s_0 \rangle \Rightarrow \langle \text{Val } v_1, s_1 \rangle; P \vdash \langle e_2, s_1 \rangle \Rightarrow \langle \text{Val } v_2, s_2 \rangle; \text{binop}(bop, v_1, v_2) = \text{Some } v \rrbracket \\ & \Longrightarrow P \vdash \langle e_1 \text{ «bop» } e_2, s_0 \rangle \Rightarrow \langle \text{Val } v, s_2 \rangle \end{aligned}$$
- | *BinOpThrow1*:  

$$\begin{aligned} & P \vdash \langle e_1, s_0 \rangle \Rightarrow \langle \text{throw } e, s_1 \rangle \Longrightarrow \\ & P \vdash \langle e_1 \text{ «bop» } e_2, s_0 \rangle \Rightarrow \langle \text{throw } e, s_1 \rangle \end{aligned}$$
- | *BinOpThrow2*:  

$$\begin{aligned} & \llbracket P \vdash \langle e_1, s_0 \rangle \Rightarrow \langle \text{Val } v_1, s_1 \rangle; P \vdash \langle e_2, s_1 \rangle \Rightarrow \langle \text{throw } e, s_2 \rangle \rrbracket \\ & \Longrightarrow P \vdash \langle e_1 \text{ «bop» } e_2, s_0 \rangle \Rightarrow \langle \text{throw } e, s_2 \rangle \end{aligned}$$
- | *Var*:  

$$\begin{aligned} & l \ V = \text{Some } v \Longrightarrow \\ & P \vdash \langle \text{Var } V, (h, l, sh) \rangle \Rightarrow \langle \text{Val } v, (h, l, sh) \rangle \end{aligned}$$
- | *LAss*:  

$$\begin{aligned} & \llbracket P \vdash \langle e, s_0 \rangle \Rightarrow \langle \text{Val } v, (h, l, sh) \rangle; l' = l(V \mapsto v) \rrbracket \\ & \Longrightarrow P \vdash \langle V := e, s_0 \rangle \Rightarrow \langle \text{unit}, (h, l', sh) \rangle \end{aligned}$$
- | *LAssThrow*:  

$$\begin{aligned} & P \vdash \langle e, s_0 \rangle \Rightarrow \langle \text{throw } e', s_1 \rangle \Longrightarrow \\ & P \vdash \langle V := e, s_0 \rangle \Rightarrow \langle \text{throw } e', s_1 \rangle \end{aligned}$$
- | *FAcc*:  

$$\begin{aligned} & \llbracket P \vdash \langle e, s_0 \rangle \Rightarrow \langle \text{addr } a, (h, l, sh) \rangle; h \ a = \text{Some}(C, fs); \\ & \quad P \vdash C \text{ has } F, \text{NonStatic}:t \text{ in } D; \\ & \quad fs(F, D) = \text{Some } v \rrbracket \\ & \Longrightarrow P \vdash \langle e \cdot F\{D\}, s_0 \rangle \Rightarrow \langle \text{Val } v, (h, l, sh) \rangle \end{aligned}$$
- | *FAccNull*:  

$$\begin{aligned} & P \vdash \langle e, s_0 \rangle \Rightarrow \langle \text{null}, s_1 \rangle \Longrightarrow \\ & P \vdash \langle e \cdot F\{D\}, s_0 \rangle \Rightarrow \langle \text{THROW } \text{NullPointer}, s_1 \rangle \end{aligned}$$
- | *FAccThrow*:  

$$\begin{aligned} & P \vdash \langle e, s_0 \rangle \Rightarrow \langle \text{throw } e', s_1 \rangle \Longrightarrow \\ & P \vdash \langle e \cdot F\{D\}, s_0 \rangle \Rightarrow \langle \text{throw } e', s_1 \rangle \end{aligned}$$
- | *FAccNone*:  

$$\begin{aligned} & \llbracket P \vdash \langle e, s_0 \rangle \Rightarrow \langle \text{addr } a, (h, l, sh) \rangle; h \ a = \text{Some}(C, fs); \\ & \quad \neg(\exists b \ t. P \vdash C \text{ has } F, b:t \text{ in } D) \rrbracket \\ & \Longrightarrow P \vdash \langle e \cdot F\{D\}, s_0 \rangle \Rightarrow \langle \text{THROW } \text{NoSuchFieldError}, (h, l, sh) \rangle \end{aligned}$$
- | *FAccStatic*:  

$$\begin{aligned} & \llbracket P \vdash \langle e, s_0 \rangle \Rightarrow \langle \text{addr } a, (h, l, sh) \rangle; h \ a = \text{Some}(C, fs); \\ & \quad P \vdash C \text{ has } F, \text{Static}:t \text{ in } D \rrbracket \\ & \Longrightarrow P \vdash \langle e \cdot F\{D\}, s_0 \rangle \Rightarrow \langle \text{THROW } \text{IncompatibleClassChangeError}, (h, l, sh) \rangle \end{aligned}$$
- | *SFAcc*:  

$$\begin{aligned} & \llbracket P \vdash C \text{ has } F, \text{Static}:t \text{ in } D; \\ & \quad sh \ D = \text{Some}(sfs, \text{Done}); \\ & \quad sfs \ F = \text{Some } v \rrbracket \\ & \Longrightarrow P \vdash \langle C \cdot_s F\{D\}, (h, l, sh) \rangle \Rightarrow \langle \text{Val } v, (h, l, sh) \rangle \end{aligned}$$



| *SFAccInit*:

$$\begin{aligned} & \llbracket P \vdash C \text{ has } F, \text{Static}:t \text{ in } D; \\ & \quad \nexists \text{ sfs. sh } D = \text{Some}(\text{sfs}, \text{Done}); P \vdash \langle \text{INIT } D \ ([D], \text{False}) \leftarrow \text{unit}, (h, l, \text{sh}) \rangle \Rightarrow \langle \text{Val } v', (h', l', \text{sh}') \rangle; \\ & \quad \text{sh}' D = \text{Some}(\text{sfs}, i); \\ & \quad \text{sfs } F = \text{Some } v \rrbracket \\ \Rightarrow & P \vdash \langle C \cdot_s F \{D\}, (h, l, \text{sh}) \rangle \Rightarrow \langle \text{Val } v, (h', l', \text{sh}') \rangle \end{aligned}$$

| *SFAccInitThrow*:

$$\begin{aligned} & \llbracket P \vdash C \text{ has } F, \text{Static}:t \text{ in } D; \\ & \quad \nexists \text{ sfs. sh } D = \text{Some}(\text{sfs}, \text{Done}); P \vdash \langle \text{INIT } D \ ([D], \text{False}) \leftarrow \text{unit}, (h, l, \text{sh}) \rangle \Rightarrow \langle \text{throw } a, s' \rangle \rrbracket \\ \Rightarrow & P \vdash \langle C \cdot_s F \{D\}, (h, l, \text{sh}) \rangle \Rightarrow \langle \text{throw } a, s' \rangle \end{aligned}$$

| *SFAccNone*:

$$\begin{aligned} & \llbracket \neg(\exists b t. P \vdash C \text{ has } F, b:t \text{ in } D) \rrbracket \\ \Rightarrow & P \vdash \langle C \cdot_s F \{D\}, s \rangle \Rightarrow \langle \text{THROW } \text{NoSuchFieldError}, s \rangle \end{aligned}$$

| *SFAccNonStatic*:

$$\begin{aligned} & \llbracket P \vdash C \text{ has } F, \text{NonStatic}:t \text{ in } D \rrbracket \\ \Rightarrow & P \vdash \langle C \cdot_s F \{D\}, s \rangle \Rightarrow \langle \text{THROW } \text{IncompatibleClassChangeError}, s \rangle \end{aligned}$$

| *FAss*:

$$\begin{aligned} & \llbracket P \vdash \langle e_1, s_0 \rangle \Rightarrow \langle \text{addr } a, s_1 \rangle; P \vdash \langle e_2, s_1 \rangle \Rightarrow \langle \text{Val } v, (h_2, l_2, \text{sh}_2) \rangle; \\ & \quad h_2 a = \text{Some}(C, \text{fs}); P \vdash C \text{ has } F, \text{NonStatic}:t \text{ in } D; \\ & \quad \text{fs}' = \text{fs}(F, D) \mapsto v; h_2' = h_2(a \mapsto (C, \text{fs}')) \rrbracket \\ \Rightarrow & P \vdash \langle e_1 \cdot F \{D\} := e_2, s_0 \rangle \Rightarrow \langle \text{unit}, (h_2', l_2, \text{sh}_2) \rangle \end{aligned}$$

| *FAssNull*:

$$\begin{aligned} & \llbracket P \vdash \langle e_1, s_0 \rangle \Rightarrow \langle \text{null}, s_1 \rangle; P \vdash \langle e_2, s_1 \rangle \Rightarrow \langle \text{Val } v, s_2 \rangle \rrbracket \Rightarrow \\ & P \vdash \langle e_1 \cdot F \{D\} := e_2, s_0 \rangle \Rightarrow \langle \text{THROW } \text{NullPointer}, s_2 \rangle \end{aligned}$$

| *FAssThrow1*:

$$\begin{aligned} & P \vdash \langle e_1, s_0 \rangle \Rightarrow \langle \text{throw } e', s_1 \rangle \Rightarrow \\ & P \vdash \langle e_1 \cdot F \{D\} := e_2, s_0 \rangle \Rightarrow \langle \text{throw } e', s_1 \rangle \end{aligned}$$

| *FAssThrow2*:

$$\begin{aligned} & \llbracket P \vdash \langle e_1, s_0 \rangle \Rightarrow \langle \text{Val } v, s_1 \rangle; P \vdash \langle e_2, s_1 \rangle \Rightarrow \langle \text{throw } e', s_2 \rangle \rrbracket \\ \Rightarrow & P \vdash \langle e_1 \cdot F \{D\} := e_2, s_0 \rangle \Rightarrow \langle \text{throw } e', s_2 \rangle \end{aligned}$$

| *FAssNone*:

$$\begin{aligned} & \llbracket P \vdash \langle e_1, s_0 \rangle \Rightarrow \langle \text{addr } a, s_1 \rangle; P \vdash \langle e_2, s_1 \rangle \Rightarrow \langle \text{Val } v, (h_2, l_2, \text{sh}_2) \rangle; \\ & \quad h_2 a = \text{Some}(C, \text{fs}); \neg(\exists b t. P \vdash C \text{ has } F, b:t \text{ in } D) \rrbracket \\ \Rightarrow & P \vdash \langle e_1 \cdot F \{D\} := e_2, s_0 \rangle \Rightarrow \langle \text{THROW } \text{NoSuchFieldError}, (h_2, l_2, \text{sh}_2) \rangle \end{aligned}$$

| *FAssStatic*:

$$\begin{aligned} & \llbracket P \vdash \langle e_1, s_0 \rangle \Rightarrow \langle \text{addr } a, s_1 \rangle; P \vdash \langle e_2, s_1 \rangle \Rightarrow \langle \text{Val } v, (h_2, l_2, \text{sh}_2) \rangle; \\ & \quad h_2 a = \text{Some}(C, \text{fs}); P \vdash C \text{ has } F, \text{Static}:t \text{ in } D \rrbracket \\ \Rightarrow & P \vdash \langle e_1 \cdot F \{D\} := e_2, s_0 \rangle \Rightarrow \langle \text{THROW } \text{IncompatibleClassChangeError}, (h_2, l_2, \text{sh}_2) \rangle \end{aligned}$$

| *SFAss*:

$$\begin{aligned} & \llbracket P \vdash \langle e_2, s_0 \rangle \Rightarrow \langle \text{Val } v, (h_1, l_1, \text{sh}_1) \rangle; \\ & \quad P \vdash C \text{ has } F, \text{Static}:t \text{ in } D; \\ & \quad \text{sh}_1 D = \text{Some}(\text{sfs}, \text{Done}); \text{sfs}' = \text{sfs}(F \mapsto v); \text{sh}_1' = \text{sh}_1(D \mapsto (\text{sfs}', \text{Done})) \rrbracket \\ \Rightarrow & P \vdash \langle C \cdot_s F \{D\} := e_2, s_0 \rangle \Rightarrow \langle \text{unit}, (h_1, l_1, \text{sh}_1') \rangle \end{aligned}$$

- | *SFAssInit*:  

$$\begin{aligned} & \llbracket P \vdash \langle e_2, s_0 \rangle \Rightarrow \langle \text{Val } v, (h_1, l_1, sh_1) \rangle; \\ & \quad P \vdash C \text{ has } F, \text{Static}:t \text{ in } D; \\ & \quad \# \text{ sfs. } sh_1 D = \text{Some}(\text{sfs}, \text{Done}); P \vdash \langle \text{INIT } D \ ([D], \text{False}) \leftarrow \text{unit}, (h_1, l_1, sh_1) \rangle \Rightarrow \langle \text{Val } v', (h', l', sh') \rangle; \\ & \quad sh' D = \text{Some}(\text{sfs}, i); \\ & \quad \text{sfs}' = \text{sfs}(F \mapsto v); sh'' = sh'(D \mapsto (\text{sfs}', i)) \rrbracket \\ \Rightarrow & P \vdash \langle C \cdot_s F \{D\} := e_2, s_0 \rangle \Rightarrow \langle \text{unit}, (h', l', sh') \rangle \end{aligned}$$
- | *SFAssInitThrow*:  

$$\begin{aligned} & \llbracket P \vdash \langle e_2, s_0 \rangle \Rightarrow \langle \text{Val } v, (h_1, l_1, sh_1) \rangle; \\ & \quad P \vdash C \text{ has } F, \text{Static}:t \text{ in } D; \\ & \quad \# \text{ sfs. } sh_1 D = \text{Some}(\text{sfs}, \text{Done}); P \vdash \langle \text{INIT } D \ ([D], \text{False}) \leftarrow \text{unit}, (h_1, l_1, sh_1) \rangle \Rightarrow \langle \text{throw } a, s^\wedge \rangle \rrbracket \\ \Rightarrow & P \vdash \langle C \cdot_s F \{D\} := e_2, s_0 \rangle \Rightarrow \langle \text{throw } a, s^\wedge \rangle \end{aligned}$$
- | *SFAssThrow*:  

$$\begin{aligned} & P \vdash \langle e_2, s_0 \rangle \Rightarrow \langle \text{throw } e', s_2 \rangle \\ \Rightarrow & P \vdash \langle C \cdot_s F \{D\} := e_2, s_0 \rangle \Rightarrow \langle \text{throw } e', s_2 \rangle \end{aligned}$$
- | *SFAssNone*:  

$$\begin{aligned} & \llbracket P \vdash \langle e_2, s_0 \rangle \Rightarrow \langle \text{Val } v, (h_2, l_2, sh_2) \rangle; \\ & \quad \neg(\exists b t. P \vdash C \text{ has } F, b:t \text{ in } D) \rrbracket \\ \Rightarrow & P \vdash \langle C \cdot_s F \{D\} := e_2, s_0 \rangle \Rightarrow \langle \text{THROW } \text{NoSuchFieldError}, (h_2, l_2, sh_2) \rangle \end{aligned}$$
- | *SFAssNonStatic*:  

$$\begin{aligned} & \llbracket P \vdash \langle e_2, s_0 \rangle \Rightarrow \langle \text{Val } v, (h_2, l_2, sh_2) \rangle; \\ & \quad P \vdash C \text{ has } F, \text{NonStatic}:t \text{ in } D \rrbracket \\ \Rightarrow & P \vdash \langle C \cdot_s F \{D\} := e_2, s_0 \rangle \Rightarrow \langle \text{THROW } \text{IncompatibleClassChangeError}, (h_2, l_2, sh_2) \rangle \end{aligned}$$
- | *CallObjThrow*:  

$$\begin{aligned} & P \vdash \langle e, s_0 \rangle \Rightarrow \langle \text{throw } e', s_1 \rangle \Rightarrow \\ & P \vdash \langle e \cdot M(ps), s_0 \rangle \Rightarrow \langle \text{throw } e', s_1 \rangle \end{aligned}$$
- | *CallParamsThrow*:  

$$\begin{aligned} & \llbracket P \vdash \langle e, s_0 \rangle \Rightarrow \langle \text{Val } v, s_1 \rangle; P \vdash \langle es, s_1 \rangle [\Rightarrow] \langle \text{map Val } vs \ @ \ \text{throw } ex \ \# \ es', s_2 \rangle \rrbracket \\ \Rightarrow & P \vdash \langle e \cdot M(es), s_0 \rangle \Rightarrow \langle \text{throw } ex, s_2 \rangle \end{aligned}$$
- | *CallNull*:  

$$\begin{aligned} & \llbracket P \vdash \langle e, s_0 \rangle \Rightarrow \langle \text{null}, s_1 \rangle; P \vdash \langle ps, s_1 \rangle [\Rightarrow] \langle \text{map Val } vs, s_2 \rangle \rrbracket \\ \Rightarrow & P \vdash \langle e \cdot M(ps), s_0 \rangle \Rightarrow \langle \text{THROW } \text{NullPointer}, s_2 \rangle \end{aligned}$$
- | *CallNone*:  

$$\begin{aligned} & \llbracket P \vdash \langle e, s_0 \rangle \Rightarrow \langle \text{addr } a, s_1 \rangle; P \vdash \langle ps, s_1 \rangle [\Rightarrow] \langle \text{map Val } vs, (h_2, l_2, sh_2) \rangle; \\ & \quad h_2 a = \text{Some}(C, fs); \neg(\exists b Ts T m D. P \vdash C \text{ sees } M, b:Ts \rightarrow T = m \text{ in } D) \rrbracket \\ \Rightarrow & P \vdash \langle e \cdot M(ps), s_0 \rangle \Rightarrow \langle \text{THROW } \text{NoSuchMethodError}, (h_2, l_2, sh_2) \rangle \end{aligned}$$
- | *CallStatic*:  

$$\begin{aligned} & \llbracket P \vdash \langle e, s_0 \rangle \Rightarrow \langle \text{addr } a, s_1 \rangle; P \vdash \langle ps, s_1 \rangle [\Rightarrow] \langle \text{map Val } vs, (h_2, l_2, sh_2) \rangle; \\ & \quad h_2 a = \text{Some}(C, fs); P \vdash C \text{ sees } M, \text{Static}:Ts \rightarrow T = m \text{ in } D \rrbracket \\ \Rightarrow & P \vdash \langle e \cdot M(ps), s_0 \rangle \Rightarrow \langle \text{THROW } \text{IncompatibleClassChangeError}, (h_2, l_2, sh_2) \rangle \end{aligned}$$
- | *Call*:  

$$\begin{aligned} & \llbracket P \vdash \langle e, s_0 \rangle \Rightarrow \langle \text{addr } a, s_1 \rangle; P \vdash \langle ps, s_1 \rangle [\Rightarrow] \langle \text{map Val } vs, (h_2, l_2, sh_2) \rangle; \\ & \quad h_2 a = \text{Some}(C, fs); P \vdash C \text{ sees } M, \text{NonStatic}:Ts \rightarrow T = (pns, \text{body}) \text{ in } D; \rrbracket \end{aligned}$$

$$\begin{aligned} & \text{length } vs = \text{length } pns; \quad l_2' = [\text{this} \mapsto \text{Addr } a, \text{ pns}[\mapsto] vs]; \\ & P \vdash \langle \text{body}, (h_2, l_2', sh_2) \rangle \Rightarrow \langle e', (h_3, l_3, sh_3) \rangle \quad \llbracket \\ \Rightarrow & P \vdash \langle e \cdot M(ps), s_0 \rangle \Rightarrow \langle e', (h_3, l_2, sh_3) \rangle \end{aligned}$$

| *SCallParamsThrow*:

$$\begin{aligned} & \llbracket P \vdash \langle es, s_0 \rangle [\Rightarrow] \langle \text{map Val } vs \text{ @ throw } ex \# es', s_2 \rangle \rrbracket \\ \Rightarrow & P \vdash \langle C \cdot_s M(es), s_0 \rangle \Rightarrow \langle \text{throw } ex, s_2 \rangle \end{aligned}$$

| *SCallNone*:

$$\begin{aligned} & \llbracket P \vdash \langle ps, s_0 \rangle [\Rightarrow] \langle \text{map Val } vs, s_2 \rangle; \\ & \quad \neg(\exists b \text{ Ts } T \text{ m } D. P \vdash C \text{ sees } M, b: \text{Ts} \rightarrow T = m \text{ in } D) \rrbracket \\ \Rightarrow & P \vdash \langle C \cdot_s M(ps), s_0 \rangle \Rightarrow \langle \text{THROW NoSuchMethodError}, s_2 \rangle \end{aligned}$$

| *SCallNonStatic*:

$$\begin{aligned} & \llbracket P \vdash \langle ps, s_0 \rangle [\Rightarrow] \langle \text{map Val } vs, s_2 \rangle; \\ & \quad P \vdash C \text{ sees } M, \text{NonStatic}: \text{Ts} \rightarrow T = m \text{ in } D \rrbracket \\ \Rightarrow & P \vdash \langle C \cdot_s M(ps), s_0 \rangle \Rightarrow \langle \text{THROW IncompatibleClassChangeError}, s_2 \rangle \end{aligned}$$

| *SCallInitThrow*:

$$\begin{aligned} & \llbracket P \vdash \langle ps, s_0 \rangle [\Rightarrow] \langle \text{map Val } vs, (h_1, l_1, sh_1) \rangle; \\ & \quad P \vdash C \text{ sees } M, \text{Static}: \text{Ts} \rightarrow T = (pns, \text{body}) \text{ in } D; \\ & \quad \nexists \text{ sfs. } sh_1 \text{ } D = \text{Some}(\text{sfs}, \text{Done}); M \neq \text{clinit}; \\ & \quad P \vdash \langle \text{INIT } D ([D], \text{False}) \leftarrow \text{unit}, (h_1, l_1, sh_1) \rangle \Rightarrow \langle \text{throw } a, s' \rangle \rrbracket \\ \Rightarrow & P \vdash \langle C \cdot_s M(ps), s_0 \rangle \Rightarrow \langle \text{throw } a, s' \rangle \end{aligned}$$

| *SCallInit*:

$$\begin{aligned} & \llbracket P \vdash \langle ps, s_0 \rangle [\Rightarrow] \langle \text{map Val } vs, (h_1, l_1, sh_1) \rangle; \\ & \quad P \vdash C \text{ sees } M, \text{Static}: \text{Ts} \rightarrow T = (pns, \text{body}) \text{ in } D; \\ & \quad \nexists \text{ sfs. } sh_1 \text{ } D = \text{Some}(\text{sfs}, \text{Done}); M \neq \text{clinit}; \\ & \quad P \vdash \langle \text{INIT } D ([D], \text{False}) \leftarrow \text{unit}, (h_1, l_1, sh_1) \rangle \Rightarrow \langle \text{Val } v', (h_2, l_2, sh_2) \rangle; \\ & \quad \text{length } vs = \text{length } pns; \quad l_2' = [pns[\mapsto] vs]; \\ & \quad P \vdash \langle \text{body}, (h_2, l_2', sh_2) \rangle \Rightarrow \langle e', (h_3, l_3, sh_3) \rangle \rrbracket \\ \Rightarrow & P \vdash \langle C \cdot_s M(ps), s_0 \rangle \Rightarrow \langle e', (h_3, l_2, sh_3) \rangle \end{aligned}$$

| *SCall*:

$$\begin{aligned} & \llbracket P \vdash \langle ps, s_0 \rangle [\Rightarrow] \langle \text{map Val } vs, (h_2, l_2, sh_2) \rangle; \\ & \quad P \vdash C \text{ sees } M, \text{Static}: \text{Ts} \rightarrow T = (pns, \text{body}) \text{ in } D; \\ & \quad sh_2 \text{ } D = \text{Some}(\text{sfs}, \text{Done}) \vee (M = \text{clinit} \wedge sh_2 \text{ } D = \text{Some}(\text{sfs}, \text{Processing})); \\ & \quad \text{length } vs = \text{length } pns; \quad l_2' = [pns[\mapsto] vs]; \\ & \quad P \vdash \langle \text{body}, (h_2, l_2', sh_2) \rangle \Rightarrow \langle e', (h_3, l_3, sh_3) \rangle \rrbracket \\ \Rightarrow & P \vdash \langle C \cdot_s M(ps), s_0 \rangle \Rightarrow \langle e', (h_3, l_2, sh_3) \rangle \end{aligned}$$

| *Block*:

$$\begin{aligned} & P \vdash \langle e_0, (h_0, l_0(V := \text{None}), sh_0) \rangle \Rightarrow \langle e_1, (h_1, l_1, sh_1) \rangle \Rightarrow \\ & P \vdash \langle \{V: T; e_0\}, (h_0, l_0, sh_0) \rangle \Rightarrow \langle e_1, (h_1, l_1(V := l_0 \text{ } V), sh_1) \rangle \end{aligned}$$

| *Seq*:

$$\begin{aligned} & \llbracket P \vdash \langle e_0, s_0 \rangle \Rightarrow \langle \text{Val } v, s_1 \rangle; P \vdash \langle e_1, s_1 \rangle \Rightarrow \langle e_2, s_2 \rangle \rrbracket \\ \Rightarrow & P \vdash \langle e_0;; e_1, s_0 \rangle \Rightarrow \langle e_2, s_2 \rangle \end{aligned}$$

| *SeqThrow*:

$$\begin{aligned} & P \vdash \langle e_0, s_0 \rangle \Rightarrow \langle \text{throw } e, s_1 \rangle \Rightarrow \\ & P \vdash \langle e_0;; e_1, s_0 \rangle \Rightarrow \langle \text{throw } e, s_1 \rangle \end{aligned}$$

| *CondT*:

$$\begin{aligned} & \llbracket P \vdash \langle e, s_0 \rangle \Rightarrow \langle \text{true}, s_1 \rangle; P \vdash \langle e_1, s_1 \rangle \Rightarrow \langle e', s_2 \rangle \rrbracket \\ & \Longrightarrow P \vdash \langle \text{if } (e) \ e_1 \ \text{else } e_2, s_0 \rangle \Rightarrow \langle e', s_2 \rangle \end{aligned}$$

| *CondF*:

$$\begin{aligned} & \llbracket P \vdash \langle e, s_0 \rangle \Rightarrow \langle \text{false}, s_1 \rangle; P \vdash \langle e_2, s_1 \rangle \Rightarrow \langle e', s_2 \rangle \rrbracket \\ & \Longrightarrow P \vdash \langle \text{if } (e) \ e_1 \ \text{else } e_2, s_0 \rangle \Rightarrow \langle e', s_2 \rangle \end{aligned}$$

| *CondThrow*:

$$\begin{aligned} & P \vdash \langle e, s_0 \rangle \Rightarrow \langle \text{throw } e', s_1 \rangle \Longrightarrow \\ & P \vdash \langle \text{if } (e) \ e_1 \ \text{else } e_2, s_0 \rangle \Rightarrow \langle \text{throw } e', s_1 \rangle \end{aligned}$$

| *WhileF*:

$$\begin{aligned} & P \vdash \langle e, s_0 \rangle \Rightarrow \langle \text{false}, s_1 \rangle \Longrightarrow \\ & P \vdash \langle \text{while } (e) \ c, s_0 \rangle \Rightarrow \langle \text{unit}, s_1 \rangle \end{aligned}$$

| *WhileT*:

$$\begin{aligned} & \llbracket P \vdash \langle e, s_0 \rangle \Rightarrow \langle \text{true}, s_1 \rangle; P \vdash \langle c, s_1 \rangle \Rightarrow \langle \text{Val } v_1, s_2 \rangle; P \vdash \langle \text{while } (e) \ c, s_2 \rangle \Rightarrow \langle e_3, s_3 \rangle \rrbracket \\ & \Longrightarrow P \vdash \langle \text{while } (e) \ c, s_0 \rangle \Rightarrow \langle e_3, s_3 \rangle \end{aligned}$$

| *WhileCondThrow*:

$$\begin{aligned} & P \vdash \langle e, s_0 \rangle \Rightarrow \langle \text{throw } e', s_1 \rangle \Longrightarrow \\ & P \vdash \langle \text{while } (e) \ c, s_0 \rangle \Rightarrow \langle \text{throw } e', s_1 \rangle \end{aligned}$$

| *WhileBodyThrow*:

$$\begin{aligned} & \llbracket P \vdash \langle e, s_0 \rangle \Rightarrow \langle \text{true}, s_1 \rangle; P \vdash \langle c, s_1 \rangle \Rightarrow \langle \text{throw } e', s_2 \rangle \rrbracket \\ & \Longrightarrow P \vdash \langle \text{while } (e) \ c, s_0 \rangle \Rightarrow \langle \text{throw } e', s_2 \rangle \end{aligned}$$

| *Throw*:

$$\begin{aligned} & P \vdash \langle e, s_0 \rangle \Rightarrow \langle \text{addr } a, s_1 \rangle \Longrightarrow \\ & P \vdash \langle \text{throw } e, s_0 \rangle \Rightarrow \langle \text{Throw } a, s_1 \rangle \end{aligned}$$

| *ThrowNull*:

$$\begin{aligned} & P \vdash \langle e, s_0 \rangle \Rightarrow \langle \text{null}, s_1 \rangle \Longrightarrow \\ & P \vdash \langle \text{throw } e, s_0 \rangle \Rightarrow \langle \text{THROW NullPointer}, s_1 \rangle \end{aligned}$$

| *ThrowThrow*:

$$\begin{aligned} & P \vdash \langle e, s_0 \rangle \Rightarrow \langle \text{throw } e', s_1 \rangle \Longrightarrow \\ & P \vdash \langle \text{throw } e, s_0 \rangle \Rightarrow \langle \text{throw } e', s_1 \rangle \end{aligned}$$

| *Try*:

$$\begin{aligned} & P \vdash \langle e_1, s_0 \rangle \Rightarrow \langle \text{Val } v_1, s_1 \rangle \Longrightarrow \\ & P \vdash \langle \text{try } e_1 \ \text{catch}(C \ V) \ e_2, s_0 \rangle \Rightarrow \langle \text{Val } v_1, s_1 \rangle \end{aligned}$$

| *TryCatch*:

$$\begin{aligned} & \llbracket P \vdash \langle e_1, s_0 \rangle \Rightarrow \langle \text{Throw } a, (h_1, l_1, sh_1) \rangle; h_1 \ a = \text{Some}(D, fs); P \vdash D \preceq^* C; \\ & \quad P \vdash \langle e_2, (h_1, l_1(V \mapsto \text{Addr } a), sh_1) \rangle \Rightarrow \langle e_2', (h_2, l_2, sh_2) \rangle \rrbracket \\ & \Longrightarrow P \vdash \langle \text{try } e_1 \ \text{catch}(C \ V) \ e_2, s_0 \rangle \Rightarrow \langle e_2', (h_2, l_2(V := l_1 \ V), sh_2) \rangle \end{aligned}$$

| *TryThrow*:

$$\begin{aligned} & \llbracket P \vdash \langle e_1, s_0 \rangle \Rightarrow \langle \text{Throw } a, (h_1, l_1, sh_1) \rangle; h_1 \ a = \text{Some}(D, fs); \neg P \vdash D \preceq^* C \rrbracket \\ & \Longrightarrow P \vdash \langle \text{try } e_1 \ \text{catch}(C \ V) \ e_2, s_0 \rangle \Rightarrow \langle \text{Throw } a, (h_1, l_1, sh_1) \rangle \end{aligned}$$

| *Nil*:

$$P \vdash \langle [], s \rangle [\Rightarrow] \langle [], s \rangle$$

| *Cons*:

$$\begin{aligned} & \llbracket P \vdash \langle e, s_0 \rangle \Rightarrow \langle \text{Val } v, s_1 \rangle; P \vdash \langle es, s_1 \rangle [\Rightarrow] \langle es', s_2 \rangle \rrbracket \\ & \Longrightarrow P \vdash \langle e \# es, s_0 \rangle [\Rightarrow] \langle \text{Val } v \# es', s_2 \rangle \end{aligned}$$

| *ConsThrow*:

$$\begin{aligned} & P \vdash \langle e, s_0 \rangle \Rightarrow \langle \text{throw } e', s_1 \rangle \Longrightarrow \\ & P \vdash \langle e \# es, s_0 \rangle [\Rightarrow] \langle \text{throw } e' \# es, s_1 \rangle \end{aligned}$$

— init rules

| *InitFinal*:

$$P \vdash \langle e, s \rangle \Rightarrow \langle e', s' \rangle \Longrightarrow P \vdash \langle \text{INIT } C \text{ (Nil, } b) \leftarrow e, s \rangle \Rightarrow \langle e', s' \rangle$$

| *InitNone*:

$$\begin{aligned} & \llbracket sh \ C = \text{None}; P \vdash \langle \text{INIT } C' \text{ (} C \# Cs, \text{False) } \leftarrow e, (h, l, sh(C \mapsto (\text{sblank } P \ C, \text{Prepared}))) \rangle \Rightarrow \langle e', s' \rangle \rrbracket \\ & \Longrightarrow P \vdash \langle \text{INIT } C' \text{ (} C \# Cs, \text{False) } \leftarrow e, (h, l, sh) \rangle \Rightarrow \langle e', s' \rangle \end{aligned}$$

| *InitDone*:

$$\begin{aligned} & \llbracket sh \ C = \text{Some}(sfs, \text{Done}); P \vdash \langle \text{INIT } C' \text{ (} Cs, \text{True) } \leftarrow e, (h, l, sh) \rangle \Rightarrow \langle e', s' \rangle \rrbracket \\ & \Longrightarrow P \vdash \langle \text{INIT } C' \text{ (} C \# Cs, \text{False) } \leftarrow e, (h, l, sh) \rangle \Rightarrow \langle e', s' \rangle \end{aligned}$$

| *InitProcessing*:

$$\begin{aligned} & \llbracket sh \ C = \text{Some}(sfs, \text{Processing}); P \vdash \langle \text{INIT } C' \text{ (} Cs, \text{True) } \leftarrow e, (h, l, sh) \rangle \Rightarrow \langle e', s' \rangle \rrbracket \\ & \Longrightarrow P \vdash \langle \text{INIT } C' \text{ (} C \# Cs, \text{False) } \leftarrow e, (h, l, sh) \rangle \Rightarrow \langle e', s' \rangle \end{aligned}$$

— note that *RI* will mark all classes in the list *Cs* with the Error flag

| *InitError*:

$$\begin{aligned} & \llbracket sh \ C = \text{Some}(sfs, \text{Error}); \\ & \quad P \vdash \langle \text{RI } (C, \text{THROW NoClassDefFoundError}); Cs \leftarrow e, (h, l, sh) \rangle \Rightarrow \langle e', s' \rangle \rrbracket \\ & \Longrightarrow P \vdash \langle \text{INIT } C' \text{ (} C \# Cs, \text{False) } \leftarrow e, (h, l, sh) \rangle \Rightarrow \langle e', s' \rangle \end{aligned}$$

| *InitObject*:

$$\begin{aligned} & \llbracket sh \ C = \text{Some}(sfs, \text{Prepared}); \\ & \quad C = \text{Object}; \\ & \quad sh' = sh(C \mapsto (sfs, \text{Processing})); \\ & \quad P \vdash \langle \text{INIT } C' \text{ (} C \# Cs, \text{True) } \leftarrow e, (h, l, sh') \rangle \Rightarrow \langle e', s' \rangle \rrbracket \\ & \Longrightarrow P \vdash \langle \text{INIT } C' \text{ (} C \# Cs, \text{False) } \leftarrow e, (h, l, sh) \rangle \Rightarrow \langle e', s' \rangle \end{aligned}$$

| *InitNonObject*:

$$\begin{aligned} & \llbracket sh \ C = \text{Some}(sfs, \text{Prepared}); \\ & \quad C \neq \text{Object}; \\ & \quad \text{class } P \ C = \text{Some } (D, r); \\ & \quad sh' = sh(C \mapsto (sfs, \text{Processing})); \\ & \quad P \vdash \langle \text{INIT } C' \text{ (} D \# C \# Cs, \text{False) } \leftarrow e, (h, l, sh') \rangle \Rightarrow \langle e', s' \rangle \rrbracket \\ & \Longrightarrow P \vdash \langle \text{INIT } C' \text{ (} C \# Cs, \text{False) } \leftarrow e, (h, l, sh) \rangle \Rightarrow \langle e', s' \rangle \end{aligned}$$

| *InitRInit*:

$$\begin{aligned} & P \vdash \langle \text{RI } (C, C \cdot s \text{clinit}([])); Cs \leftarrow e, (h, l, sh) \rangle \Rightarrow \langle e', s' \rangle \\ & \Longrightarrow P \vdash \langle \text{INIT } C' \text{ (} C \# Cs, \text{True) } \leftarrow e, (h, l, sh) \rangle \Rightarrow \langle e', s' \rangle \end{aligned}$$

| *RInit*:

$$\llbracket P \vdash \langle e', s \rangle \Rightarrow \langle \text{Val } v, (h', l', sh') \rangle; \rrbracket$$

$$\begin{aligned}
& sh' C = Some(sfs, i); sh'' = sh'(C \mapsto (sfs, Done)); \\
& C' = last(C \# Cs); \\
& P \vdash \langle INIT C' (Cs, True) \leftarrow e, (h', l', sh'') \rangle \Rightarrow \langle e_1, s_1 \rangle \\
\Rightarrow & P \vdash \langle RI (C, e'); Cs \leftarrow e, s \rangle \Rightarrow \langle e_1, s_1 \rangle
\end{aligned}$$

| *RInitInitFail*:

$$\begin{aligned}
& \llbracket P \vdash \langle e', s \rangle \Rightarrow \langle throw a, (h', l', sh') \rangle; \\
& sh' C = Some(sfs, i); sh'' = sh'(C \mapsto (sfs, Error)); \\
& P \vdash \langle RI (D, throw a); Cs \leftarrow e, (h', l', sh'') \rangle \Rightarrow \langle e_1, s_1 \rangle \rrbracket \\
\Rightarrow & P \vdash \langle RI (C, e'); D \# Cs \leftarrow e, s \rangle \Rightarrow \langle e_1, s_1 \rangle
\end{aligned}$$

| *RInitFailFinal*:

$$\begin{aligned}
& \llbracket P \vdash \langle e', s \rangle \Rightarrow \langle throw a, (h', l', sh') \rangle; \\
& sh' C = Some(sfs, i); sh'' = sh'(C \mapsto (sfs, Error)) \rrbracket \\
\Rightarrow & P \vdash \langle RI (C, e'); Nil \leftarrow e, s \rangle \Rightarrow \langle throw a, (h', l', sh'') \rangle
\end{aligned}$$

### 1.15.1 Final expressions

**lemma** *eval-final*:  $P \vdash \langle e, s \rangle \Rightarrow \langle e', s' \rangle \Rightarrow final e'$   
**and** *evals-final*:  $P \vdash \langle es, s \rangle [\Rightarrow] \langle es', s' \rangle \Rightarrow finals es' \langle proof \rangle$

Only used later, in the small to big translation, but is already a good sanity check:

**lemma** *eval-finalId*:  $final e \Rightarrow P \vdash \langle e, s \rangle \Rightarrow \langle e, s \rangle \langle proof \rangle$   
**lemma** *eval-final-same*:  $\llbracket P \vdash \langle e, s \rangle \Rightarrow \langle e', s' \rangle; final e \rrbracket \Rightarrow e = e' \wedge s = s' \langle proof \rangle$   
**lemma** *eval-finalsId*:  
**assumes** *finals*:  $finals es$  **shows**  $P \vdash \langle es, s \rangle [\Rightarrow] \langle es, s \rangle \langle proof \rangle$   
**lemma** *evals-finals-same*:  
**assumes** *finals*:  $finals es$   
**shows**  $P \vdash \langle es, s \rangle [\Rightarrow] \langle es', s' \rangle \Rightarrow es = es' \wedge s = s'$   
 $\langle proof \rangle$

### 1.15.2 Property preservation

**lemma** *evals-length*:  $P \vdash \langle es, s \rangle [\Rightarrow] \langle es', s' \rangle \Rightarrow length es = length es'$   
 $\langle proof \rangle$

**corollary** *evals-empty*:  $P \vdash \langle es, s \rangle [\Rightarrow] \langle es', s' \rangle \Rightarrow (es = []) = (es' = [])$   
 $\langle proof \rangle$

**theorem** *eval-hext*:  $P \vdash \langle e, (h, l, sh) \rangle \Rightarrow \langle e', (h', l', sh') \rangle \Rightarrow h \trianglelefteq h'$   
**and** *evals-hext*:  $P \vdash \langle es, (h, l, sh) \rangle [\Rightarrow] \langle es', (h', l', sh') \rangle \Rightarrow h \trianglelefteq h' \langle proof \rangle$

**lemma** *eval-lcl-incr*:  $P \vdash \langle e, (h_0, l_0, sh_0) \rangle \Rightarrow \langle e', (h_1, l_1, sh_1) \rangle \Rightarrow dom l_0 \subseteq dom l_1$   
**and** *evals-lcl-incr*:  $P \vdash \langle es, (h_0, l_0, sh_0) \rangle [\Rightarrow] \langle es', (h_1, l_1, sh_1) \rangle \Rightarrow dom l_0 \subseteq dom l_1 \langle proof \rangle$

**lemma**

**shows** *init-ri-same-loc*:  $P \vdash \langle e, (h, l, sh) \rangle \Rightarrow \langle e', (h', l', sh') \rangle$   
 $\Rightarrow (\bigwedge C Cs b M a. e = INIT C (Cs, b) \leftarrow unit \vee e = C \cdot_s M (\square) \vee e = RI (C, Throw a); Cs \leftarrow unit$   
 $\vee e = RI (C, C \cdot_s clinit(\square)); Cs \leftarrow unit$   
 $\Rightarrow l = l')$

**and**  $P \vdash \langle es, (h, l, sh) \rangle [\Rightarrow] \langle es', (h', l', sh') \rangle \Rightarrow True$

$\langle \text{proof} \rangle$

**lemma** *init-same-loc*:  $P \vdash \langle \text{INIT } C (Cs, b) \leftarrow \text{unit}, (h, l, sh) \rangle \Rightarrow \langle e', (h', l', sh') \rangle \Longrightarrow l = l'$   
 $\langle \text{proof} \rangle$

**lemma** *assumes wf*: *wf-J-prog*  $P$

**shows** *eval-proc-pres'*:  $P \vdash \langle e, (h, l, sh) \rangle \Rightarrow \langle e', (h', l', sh') \rangle$   
 $\Longrightarrow \text{not-init } C e \Longrightarrow \exists \text{sfs. sh } C = \llbracket (\text{sfs}, \text{Processing}) \rrbracket \Longrightarrow \exists \text{sfs}'. \text{sh}' C = \llbracket (\text{sfs}', \text{Processing}) \rrbracket$

**and** *evals-proc-pres'*:  $P \vdash \langle es, (h, l, sh) \rangle [\Rightarrow] \langle es', (h', l', sh') \rangle$

$\Longrightarrow \text{not-inits } C es \Longrightarrow \exists \text{sfs. sh } C = \llbracket (\text{sfs}, \text{Processing}) \rrbracket \Longrightarrow \exists \text{sfs}'. \text{sh}' C = \llbracket (\text{sfs}', \text{Processing}) \rrbracket \langle \text{proof} \rangle \langle \text{proof} \rangle \langle \text{proof} \rangle \langle \text{proof} \rangle \langle \text{proof} \rangle$

## 1.16 Definite assignment

**theory** *DefAss* **imports** *BigStep* **begin**

### 1.16.1 Hypersets

**type-synonym** *'a hyperset* = *'a set option*

**definition** *hyperUn* :: *'a hyperset*  $\Rightarrow$  *'a hyperset*  $\Rightarrow$  *'a hyperset* (**infixl**  $\langle \sqcup \rangle$  65)

**where**

$A \sqcup B \equiv \text{case } A \text{ of } \text{None} \Rightarrow \text{None}$   
 $| \llbracket A \rrbracket \Rightarrow (\text{case } B \text{ of } \text{None} \Rightarrow \text{None} | \llbracket B \rrbracket \Rightarrow \llbracket A \cup B \rrbracket)$

**definition** *hyperInt* :: *'a hyperset*  $\Rightarrow$  *'a hyperset*  $\Rightarrow$  *'a hyperset* (**infixl**  $\langle \sqcap \rangle$  70)

**where**

$A \sqcap B \equiv \text{case } A \text{ of } \text{None} \Rightarrow B$   
 $| \llbracket A \rrbracket \Rightarrow (\text{case } B \text{ of } \text{None} \Rightarrow \llbracket A \rrbracket | \llbracket B \rrbracket \Rightarrow \llbracket A \cap B \rrbracket)$

**definition** *hyperDiff1* :: *'a hyperset*  $\Rightarrow$  *'a*  $\Rightarrow$  *'a hyperset* (**infixl**  $\langle \ominus \rangle$  65)

**where**

$A \ominus a \equiv \text{case } A \text{ of } \text{None} \Rightarrow \text{None} | \llbracket A \rrbracket \Rightarrow \llbracket A - \{a\} \rrbracket$

**definition** *hyper-isin* :: *'a*  $\Rightarrow$  *'a hyperset*  $\Rightarrow$  *bool* (**infix**  $\langle \in \in \rangle$  50)

**where**

$a \in \in A \equiv \text{case } A \text{ of } \text{None} \Rightarrow \text{True} | \llbracket A \rrbracket \Rightarrow a \in A$

**definition** *hyper-subset* :: *'a hyperset*  $\Rightarrow$  *'a hyperset*  $\Rightarrow$  *bool* (**infix**  $\langle \sqsubseteq \rangle$  50)

**where**

$A \sqsubseteq B \equiv \text{case } B \text{ of } \text{None} \Rightarrow \text{True}$   
 $| \llbracket B \rrbracket \Rightarrow (\text{case } A \text{ of } \text{None} \Rightarrow \text{False} | \llbracket A \rrbracket \Rightarrow A \subseteq B)$

**lemmas** *hyperset-defs* =

*hyperUn-def hyperInt-def hyperDiff1-def hyper-isin-def hyper-subset-def*

**lemma** [*simp*]:  $\llbracket \{\} \rrbracket \sqcup A = A \wedge A \sqcup \llbracket \{\} \rrbracket = A \langle \text{proof} \rangle$

**lemma** [*simp*]:  $\llbracket A \rrbracket \sqcup \llbracket B \rrbracket = \llbracket A \cup B \rrbracket \wedge \llbracket A \rrbracket \ominus a = \llbracket A - \{a\} \rrbracket \langle \text{proof} \rangle$

**lemma** [*simp*]:  $\text{None} \sqcup A = \text{None} \wedge A \sqcup \text{None} = \text{None} \langle \text{proof} \rangle$

**lemma** [*simp*]:  $a \in \in \text{None} \wedge \text{None} \ominus a = \text{None} \langle \text{proof} \rangle$

**lemma** *hyper-isin-union*:  $x \in \in \llbracket A \rrbracket \Longrightarrow x \in \in \llbracket A \rrbracket \sqcup B$

$\langle \text{proof} \rangle$

**lemma** *hyperUn-assoc*:  $(A \sqcup B) \sqcup C = A \sqcup (B \sqcup C)$ *(proof)*

**lemma** *hyper-insert-comm*:  $A \sqcup [\{a\}] = [\{a\}] \sqcup A \wedge A \sqcup ([\{a\}] \sqcup B) = [\{a\}] \sqcup (A \sqcup B)$ *(proof)*

**lemma** *hyper-comm*:  $A \sqcup B = B \sqcup A \wedge A \sqcup B \sqcup C = B \sqcup A \sqcup C$ *(proof)*

### 1.16.2 Definite assignment

**primrec**

$\mathcal{A} :: 'a \text{ exp} \Rightarrow 'a \text{ hyperset}$

**and**  $\mathcal{A}s :: 'a \text{ exp list} \Rightarrow 'a \text{ hyperset}$

**where**

$\mathcal{A} (\text{new } C) = [\{\}]$

$\mathcal{A} (\text{Cast } C \ e) = \mathcal{A} \ e$

$\mathcal{A} (\text{Val } v) = [\{\}]$

$\mathcal{A} (e_1 \ll \text{bop} \gg e_2) = \mathcal{A} \ e_1 \sqcup \mathcal{A} \ e_2$

$\mathcal{A} (\text{Var } V) = [\{\}]$

$\mathcal{A} (\text{LAss } V \ e) = [\{V\}] \sqcup \mathcal{A} \ e$

$\mathcal{A} (e \cdot F\{D\}) = \mathcal{A} \ e$

$\mathcal{A} (C \cdot_s F\{D\}) = [\{\}]$

$\mathcal{A} (e_1 \cdot F\{D\} := e_2) = \mathcal{A} \ e_1 \sqcup \mathcal{A} \ e_2$

$\mathcal{A} (C \cdot_s F\{D\} := e_2) = \mathcal{A} \ e_2$

$\mathcal{A} (e \cdot M(es)) = \mathcal{A} \ e \sqcup \mathcal{A}s \ es$

$\mathcal{A} (C \cdot_s M(es)) = \mathcal{A}s \ es$

$\mathcal{A} (\{V:T; e\}) = \mathcal{A} \ e \ominus V$

$\mathcal{A} (e_1 ;; e_2) = \mathcal{A} \ e_1 \sqcup \mathcal{A} \ e_2$

$\mathcal{A} (\text{if } (e) \ e_1 \ \text{else } e_2) = \mathcal{A} \ e \sqcup (\mathcal{A} \ e_1 \sqcap \mathcal{A} \ e_2)$

$\mathcal{A} (\text{while } (b) \ e) = \mathcal{A} \ b$

$\mathcal{A} (\text{throw } e) = \text{None}$

$\mathcal{A} (\text{try } e_1 \ \text{catch}(C \ V) \ e_2) = \mathcal{A} \ e_1 \sqcap (\mathcal{A} \ e_2 \ominus V)$

$\mathcal{A} (\text{INIT } C \ (Cs, b) \leftarrow e) = [\{\}]$

$\mathcal{A} (\text{RI } (C, e); Cs \leftarrow e') = \mathcal{A} \ e$

$\mathcal{A}s ([\ ]) = [\{\}]$

$\mathcal{A}s (e \# es) = \mathcal{A} \ e \sqcup \mathcal{A}s \ es$

**primrec**

$\mathcal{D} :: 'a \text{ exp} \Rightarrow 'a \text{ hyperset} \Rightarrow \text{bool}$

**and**  $\mathcal{D}s :: 'a \text{ exp list} \Rightarrow 'a \text{ hyperset} \Rightarrow \text{bool}$

**where**

$\mathcal{D} (\text{new } C) \ A = \text{True}$

$\mathcal{D} (\text{Cast } C \ e) \ A = \mathcal{D} \ e \ A$

$\mathcal{D} (\text{Val } v) \ A = \text{True}$

$\mathcal{D} (e_1 \ll \text{bop} \gg e_2) \ A = (\mathcal{D} \ e_1 \ A \wedge \mathcal{D} \ e_2 \ (A \sqcup \mathcal{A} \ e_1))$

$\mathcal{D} (\text{Var } V) \ A = (V \in \in A)$

$\mathcal{D} (\text{LAss } V \ e) \ A = \mathcal{D} \ e \ A$

$\mathcal{D} (e \cdot F\{D\}) \ A = \mathcal{D} \ e \ A$

$\mathcal{D} (C \cdot_s F\{D\}) \ A = \text{True}$

$\mathcal{D} (e_1 \cdot F\{D\} := e_2) \ A = (\mathcal{D} \ e_1 \ A \wedge \mathcal{D} \ e_2 \ (A \sqcup \mathcal{A} \ e_1))$

$\mathcal{D} (C \cdot_s F\{D\} := e_2) \ A = \mathcal{D} \ e_2 \ A$

$\mathcal{D} (e \cdot M(es)) \ A = (\mathcal{D} \ e \ A \wedge \mathcal{D}s \ es \ (A \sqcup \mathcal{A} \ e))$

$\mathcal{D} (C \cdot_s M(es)) \ A = \mathcal{D}s \ es \ A$

$\mathcal{D} (\{V:T; e\}) \ A = \mathcal{D} \ e \ (A \ominus V)$

$\mathcal{D} (e_1 ;; e_2) \ A = (\mathcal{D} \ e_1 \ A \wedge \mathcal{D} \ e_2 \ (A \sqcup \mathcal{A} \ e_1))$

$\mathcal{D} (\text{if } (e) \ e_1 \ \text{else } e_2) \ A =$

$(\mathcal{D} \ e \ A \wedge \mathcal{D} \ e_1 \ (A \sqcup \mathcal{A} \ e) \wedge \mathcal{D} \ e_2 \ (A \sqcup \mathcal{A} \ e))$



|  $\mathcal{D} (\text{while } (e) \ c) \ A = (\mathcal{D} \ e \ A \wedge \mathcal{D} \ c \ (A \sqcup \mathcal{A} \ e))$   
|  $\mathcal{D} (\text{throw } e) \ A = \mathcal{D} \ e \ A$   
|  $\mathcal{D} (\text{try } e_1 \ \text{catch}(C \ V) \ e_2) \ A = (\mathcal{D} \ e_1 \ A \wedge \mathcal{D} \ e_2 \ (A \sqcup [\{V\}]))$   
|  $\mathcal{D} (\text{INIT } C \ (Cs, b) \leftarrow e) \ A = \mathcal{D} \ e \ A$   
|  $\mathcal{D} (\text{RI } (C, e); Cs \leftarrow e') \ A = (\mathcal{D} \ e \ A \wedge \mathcal{D} \ e' \ A)$

|  $\mathcal{D}s \ (\square) \ A = \text{True}$   
|  $\mathcal{D}s \ (e \# es) \ A = (\mathcal{D} \ e \ A \wedge \mathcal{D}s \ es \ (A \sqcup \mathcal{A} \ e))$

**lemma** *As-map-Val[simp]*:  $\mathcal{A}s \ (\text{map } \text{Val } vs) = [\{\}] \langle \text{proof} \rangle$

**lemma** *D-append[iff]*:  $\bigwedge A. \mathcal{D}s \ (es \ @ \ es') \ A = (\mathcal{D}s \ es \ A \wedge \mathcal{D}s \ es' \ (A \sqcup \mathcal{A}s \ es)) \langle \text{proof} \rangle$

**lemma** *A-fv*:  $\bigwedge A. \mathcal{A} \ e = [A] \implies A \subseteq \text{fv } e$   
**and**  $\bigwedge A. \mathcal{A}s \ es = [A] \implies A \subseteq \text{fvs } es \langle \text{proof} \rangle$

**lemma** *sqUn-lem*:  $A \sqsubseteq A' \implies A \sqcup B \sqsubseteq A' \sqcup B \langle \text{proof} \rangle$

**lemma** *diff-lem*:  $A \sqsubseteq A' \implies A \ominus b \sqsubseteq A' \ominus b \langle \text{proof} \rangle$

**lemma** *D-mono*:  $\bigwedge A \ A'. A \sqsubseteq A' \implies \mathcal{D} \ e \ A \implies \mathcal{D} \ (e :: 'a \ \text{exp}) \ A'$   
**and** *Ds-mono*:  $\bigwedge A \ A'. A \sqsubseteq A' \implies \mathcal{D}s \ es \ A \implies \mathcal{D}s \ (es :: 'a \ \text{exp list}) \ A' \langle \text{proof} \rangle$

**lemma** *D-mono'*:  $\mathcal{D} \ e \ A \implies A \sqsubseteq A' \implies \mathcal{D} \ e \ A'$   
**and** *Ds-mono'*:  $\mathcal{D}s \ es \ A \implies A \sqsubseteq A' \implies \mathcal{D}s \ es \ A' \langle \text{proof} \rangle$

**lemma** *Ds-Vals*:  $\mathcal{D}s \ (\text{map } \text{Val } vs) \ A \langle \text{proof} \rangle$

end

## 1.17 Conformance Relations for Type Soundness Proofs

**theory** *Conform*  
**imports** *Exceptions*  
**begin**

**definition** *conf* ::  $'m \ \text{prog} \Rightarrow \text{heap} \Rightarrow \text{val} \Rightarrow \text{ty} \Rightarrow \text{bool}$  ( $\langle -, - \vdash - : \leq \rangle$  [51,51,51,51] 50)

**where**

$P, h \vdash v : \leq T \equiv$   
 $\exists T'. \text{typeof}_h \ v = \text{Some } T' \wedge P \vdash T' \leq T$

**definition** *oconf* ::  $'m \ \text{prog} \Rightarrow \text{heap} \Rightarrow \text{obj} \Rightarrow \text{bool}$  ( $\langle -, - \vdash - \surd \rangle$  [51,51,51] 50)

**where**

$P, h \vdash \text{obj } \surd \equiv$   
 $\text{let } (C, fs) = \text{obj in } \forall F \ D \ T. P \vdash C \ \text{has } F, \text{NonStatic}:T \ \text{in } D \longrightarrow$   
 $(\exists v. fs(F, D) = \text{Some } v \wedge P, h \vdash v : \leq T)$

**definition** *soconf* ::  $'m \ \text{prog} \Rightarrow \text{heap} \Rightarrow \text{cname} \Rightarrow \text{sfields} \Rightarrow \text{bool}$  ( $\langle -, -, - \vdash_s - \surd \rangle$  [51,51,51,51] 50)

**where**

$P, h, C \vdash_s \text{sfs } \surd \equiv$   
 $\forall F \ T. P \vdash C \ \text{has } F, \text{Static}:T \ \text{in } C \longrightarrow$   
 $(\exists v. \text{sfs } F = \text{Some } v \wedge P, h \vdash v : \leq T)$

**definition** *hconf* ::  $'m \ \text{prog} \Rightarrow \text{heap} \Rightarrow \text{bool}$  ( $\langle - \vdash - \surd \rangle$  [51,51] 50)

**where**

$$P \vdash h \checkmark \equiv (\forall a \text{ obj. } h a = \text{Some obj} \longrightarrow P, h \vdash \text{obj } \checkmark) \wedge \text{preallocated } h$$

**definition**  $shconf :: 'm \text{ prog} \Rightarrow \text{heap} \Rightarrow \text{sheap} \Rightarrow \text{bool}$  ( $\langle \cdot, \cdot \vdash_s \cdot \checkmark \rangle$  [51,51,51] 50)

**where**

$$P, h \vdash_s sh \checkmark \equiv (\forall C \text{ sfs } i. sh C = \text{Some}(sfs, i) \longrightarrow P, h, C \vdash_s sfs \checkmark)$$

**definition**  $lconf :: 'm \text{ prog} \Rightarrow \text{heap} \Rightarrow (\text{vname} \rightarrow \text{val}) \Rightarrow (\text{vname} \rightarrow \text{ty}) \Rightarrow \text{bool}$  ( $\langle \cdot, \cdot \vdash \cdot'(\leq) \rangle$  [51,51,51,51] 50)

**where**

$$P, h \vdash l (\leq) E \equiv \forall V v. l V = \text{Some } v \longrightarrow (\exists T. E V = \text{Some } T \wedge P, h \vdash v \leq T)$$

**abbreviation**

$$\begin{aligned} \text{confs} &:: 'm \text{ prog} \Rightarrow \text{heap} \Rightarrow \text{val list} \Rightarrow \text{ty list} \Rightarrow \text{bool} \\ &(\langle \cdot, \cdot \vdash \cdot [\leq] \rangle \rightarrow [51,51,51,51] 50) \text{ where} \\ P, h \vdash vs [\leq] Ts &\equiv \text{list-all2 } (\text{conf } P h) \text{ vs } Ts \end{aligned}$$

### 1.17.1 Value conformance $\leq$

**lemma**  $\text{conf-Null}$  [simp]:  $P, h \vdash \text{Null} \leq T = P \vdash NT \leq T \langle \text{proof} \rangle$

**lemma**  $\text{typeof-conf}$  [simp]:  $\text{typeof}_h v = \text{Some } T \Longrightarrow P, h \vdash v \leq T \langle \text{proof} \rangle$

**lemma**  $\text{typeof-lit-conf}$  [simp]:  $\text{typeof } v = \text{Some } T \Longrightarrow P, h \vdash v \leq T \langle \text{proof} \rangle$

**lemma**  $\text{defval-conf}$  [simp]:  $P, h \vdash \text{default-val } T \leq T \langle \text{proof} \rangle$

**lemma**  $\text{conf-upd-obj}$ :  $h a = \text{Some}(C, fs) \Longrightarrow (P, h(a \mapsto (C, fs')) \vdash x \leq T) = (P, h \vdash x \leq T) \langle \text{proof} \rangle$

**lemma**  $\text{conf-widen}$ :  $P, h \vdash v \leq T \Longrightarrow P \vdash T \leq T' \Longrightarrow P, h \vdash v \leq T' \langle \text{proof} \rangle$

**lemma**  $\text{conf-hext}$ :  $h \leq h' \Longrightarrow P, h \vdash v \leq T \Longrightarrow P, h' \vdash v \leq T \langle \text{proof} \rangle$

**lemma**  $\text{conf-ClassD}$ :  $P, h \vdash v \leq \text{Class } C \Longrightarrow$

$$v = \text{Null} \vee (\exists a \text{ obj } T. v = \text{Addr } a \wedge h a = \text{Some obj} \wedge \text{obj-ty obj} = T \wedge P \vdash T \leq \text{Class } C) \langle \text{proof} \rangle$$

**lemma**  $\text{conf-NT}$  [iff]:  $P, h \vdash v \leq NT = (v = \text{Null}) \langle \text{proof} \rangle$

**lemma**  $\text{non-npD}$ :  $\llbracket v \neq \text{Null}; P, h \vdash v \leq \text{Class } C \rrbracket$

$$\Longrightarrow \exists a C' fs. v = \text{Addr } a \wedge h a = \text{Some}(C', fs) \wedge P \vdash C' \preceq^* C \langle \text{proof} \rangle$$

### 1.17.2 Value list conformance $[\leq]$

**lemma**  $\text{confs-widens}$  [trans]:  $\llbracket P, h \vdash vs [\leq] Ts; P \vdash Ts [\leq] Ts' \rrbracket \Longrightarrow P, h \vdash vs [\leq] Ts' \langle \text{proof} \rangle$

**lemma**  $\text{confs-rev}$ :  $P, h \vdash \text{rev } s [\leq] t = (P, h \vdash s [\leq] \text{rev } t) \langle \text{proof} \rangle$

**lemma**  $\text{confs-conv-map}$ :

$$\bigwedge Ts'. P, h \vdash vs [\leq] Ts' = (\exists Ts. \text{map } \text{typeof}_h vs = \text{map } \text{Some } Ts \wedge P \vdash Ts [\leq] Ts') \langle \text{proof} \rangle$$

**lemma**  $\text{confs-hext}$ :  $P, h \vdash vs [\leq] Ts \Longrightarrow h \leq h' \Longrightarrow P, h' \vdash vs [\leq] Ts \langle \text{proof} \rangle$

**lemma**  $\text{confs-Cons2}$ :  $P, h \vdash xs [\leq] y \# ys = (\exists z zs. xs = z \# zs \wedge P, h \vdash z \leq y \wedge P, h \vdash zs [\leq] ys) \langle \text{proof} \rangle$

### 1.17.3 Object conformance

**lemma**  $\text{oconf-hext}$ :  $P, h \vdash \text{obj } \checkmark \Longrightarrow h \leq h' \Longrightarrow P, h' \vdash \text{obj } \checkmark \langle \text{proof} \rangle$

**lemma**  $\text{oconf-blank}$ :

$$P \vdash C \text{ has-fields } FDTs \Longrightarrow P, h \vdash \text{blank } P C \checkmark \langle \text{proof} \rangle$$

**lemma**  $\text{oconf-fupd}$  [intro?]:

$$\begin{aligned} \llbracket P \vdash C \text{ has } F, \text{NonStatic}: T \text{ in } D; P, h \vdash v \leq T; P, h \vdash (C, fs) \checkmark \rrbracket \\ \Longrightarrow P, h \vdash (C, fs((F, D) \mapsto v)) \checkmark \langle \text{proof} \rangle \end{aligned}$$

### 1.17.4 Static object conformance

**lemma soconf-hest:**  $P, h, C \vdash_s \text{obj} \checkmark \implies h \leq h' \implies P, h', C \vdash_s \text{obj} \checkmark \langle \text{proof} \rangle$

**lemma soconf-sblank:**

$P \vdash C \text{ has-fields } FDTs \implies P, h, C \vdash_s \text{sblank } P \ C \checkmark \langle \text{proof} \rangle$

**lemma soconf-fupd** [intro?]:

$\llbracket P \vdash C \text{ has } F, \text{Static}: T \text{ in } C; P, h \vdash v : \leq T; P, h, C \vdash_s \text{sfs} \checkmark \rrbracket$   
 $\implies P, h, C \vdash_s \text{sfs}(F \mapsto v) \checkmark \langle \text{proof} \rangle$

### 1.17.5 Heap conformance

**lemma hconfD:**  $\llbracket P \vdash h \checkmark; h \ a = \text{Some } \text{obj} \rrbracket \implies P, h \vdash \text{obj} \checkmark \langle \text{proof} \rangle$

**lemma hconf-new:**  $\llbracket P \vdash h \checkmark; h \ a = \text{None}; P, h \vdash \text{obj} \checkmark \rrbracket \implies P \vdash h(a \mapsto \text{obj}) \checkmark \langle \text{proof} \rangle$

**lemma hconf-upd-obj:**  $\llbracket P \vdash h \checkmark; h \ a = \text{Some}(C, \text{fs}); P, h \vdash (C, \text{fs}') \checkmark \rrbracket \implies P \vdash h(a \mapsto (C, \text{fs}')) \checkmark \langle \text{proof} \rangle$

### 1.17.6 Class statics conformance

**lemma shconfD:**  $\llbracket P, h \vdash_s \text{sh} \checkmark; \text{sh } C = \text{Some}(\text{sfs}, i) \rrbracket \implies P, h, C \vdash_s \text{sfs} \checkmark \langle \text{proof} \rangle$

**lemma shconf-upd-obj:**  $\llbracket P, h \vdash_s \text{sh} \checkmark; P, h, C \vdash_s \text{sfs}' \checkmark \rrbracket$

$\implies P, h \vdash_s \text{sh}(C \mapsto (\text{sfs}', i')) \checkmark \langle \text{proof} \rangle$

**lemma shconf-hnew:**  $\llbracket P, h \vdash_s \text{sh} \checkmark; h \ a = \text{None} \rrbracket \implies P, h(a \mapsto \text{obj}) \vdash_s \text{sh} \checkmark \langle \text{proof} \rangle$

**lemma shconf-hupd-obj:**  $\llbracket P, h \vdash_s \text{sh} \checkmark; h \ a = \text{Some}(C, \text{fs}) \rrbracket \implies P, h(a \mapsto (C, \text{fs}')) \vdash_s \text{sh} \checkmark \langle \text{proof} \rangle$

### 1.17.7 Local variable conformance

**lemma lconf-hest:**  $\llbracket P, h \vdash l (: \leq) E; h \leq h' \rrbracket \implies P, h' \vdash l (: \leq) E \langle \text{proof} \rangle$

**lemma lconf-upd:**

$\llbracket P, h \vdash l (: \leq) E; P, h \vdash v : \leq T; E \ V = \text{Some } T \rrbracket \implies P, h \vdash l(V \mapsto v) (: \leq) E \langle \text{proof} \rangle$

**lemma lconf-empty**[iff]:  $P, h \vdash \text{Map.empty} (: \leq) E \langle \text{proof} \rangle$

**lemma lconf-upd2:**  $\llbracket P, h \vdash l (: \leq) E; P, h \vdash v : \leq T \rrbracket \implies P, h \vdash l(V \mapsto v) (: \leq) E(V \mapsto T) \langle \text{proof} \rangle$

end

## 1.18 Small Step Semantics

**theory SmallStep**

**imports Expr State WWellForm**

**begin**

**fun blocks** ::  $vname \ \text{list} * ty \ \text{list} * val \ \text{list} * expr \Rightarrow expr$

**where**

$\text{blocks}(V \# Vs, T \# Ts, v \# vs, e) = \{ V : T := \text{Val } v; \text{blocks}(Vs, Ts, vs, e) \}$

$|\text{blocks}([], [], [], e) = e$

**lemmas blocks-induct** =  $\text{blocks.induct}[\text{split-format } (\text{complete})]$

**lemma** [simp]:

$\llbracket \text{size } vs = \text{size } Vs; \text{size } Ts = \text{size } Vs \rrbracket \implies \text{fv}(\text{blocks}(Vs, Ts, vs, e)) = \text{fv } e - \text{set } Vs \langle \text{proof} \rangle$

**lemma sub-RI-blocks-body**[iff]:  $\text{length } vs = \text{length } pns \implies \text{length } Ts = \text{length } pns$

$\implies \text{sub-RI } (\text{blocks } (pns, Ts, vs, \text{body})) \longleftrightarrow \text{sub-RI } \text{body}$

$\langle \text{proof} \rangle$

**definition**  $assigned :: 'a \Rightarrow 'a \text{ exp} \Rightarrow \text{bool}$

**where**

$$assigned \ V \ e \equiv \exists v \ e'. e = (V := Val \ v;; e')$$

— expression is okay to go the right side of *INIT* or *RI*  $\leftarrow$  or to have indicator Boolean be True (in latter case, given that class is also verified initialized)

**fun**  $icheck :: 'm \ \text{prog} \Rightarrow \text{cname} \Rightarrow 'a \ \text{exp} \Rightarrow \text{bool}$  **where**

$$icheck \ P \ C' \ (\text{new } C) = (C' = C) \mid$$

$$icheck \ P \ D' \ (C \cdot_s F \{D\}) = ((D' = D) \wedge (\exists T. P \vdash C \text{ has } F, \text{Static}: T \text{ in } D)) \mid$$

$$icheck \ P \ D' \ (C \cdot_s F \{D\} := (Val \ v)) = ((D' = D) \wedge (\exists T. P \vdash C \text{ has } F, \text{Static}: T \text{ in } D)) \mid$$

$$icheck \ P \ D \ (C \cdot_s M(es)) = ((\exists vs. es = \text{map } Val \ vs) \wedge (\exists Ts \ T \ m. P \vdash C \text{ sees } M, \text{Static}: Ts \rightarrow T = m \text{ in } D)) \mid$$

$$icheck \ P \ - = \text{False}$$

**lemma**  $nichck-SFAss-nonVal: \text{val-of } e_2 = \text{None} \implies \neg icheck \ P \ C' \ (C \cdot_s F \{D\} := (e_2 :: 'a \ \text{exp}))$

*<proof>*

**inductive-set**

$$\text{red} :: J\text{-prog} \Rightarrow ((\text{expr} \times \text{state} \times \text{bool}) \times (\text{expr} \times \text{state} \times \text{bool})) \ \text{set}$$

$$\text{and } \text{reds} :: J\text{-prog} \Rightarrow ((\text{expr list} \times \text{state} \times \text{bool}) \times (\text{expr list} \times \text{state} \times \text{bool})) \ \text{set}$$

$$\text{and } \text{red}' :: J\text{-prog} \Rightarrow \text{expr} \Rightarrow \text{state} \Rightarrow \text{bool} \Rightarrow \text{expr} \Rightarrow \text{state} \Rightarrow \text{bool} \Rightarrow \text{bool}$$

$$(\leftarrow \vdash ((1 \langle -, /-, /- \rangle) \rightarrow / (1 \langle -, /-, /- \rangle))) \triangleright [51, 0, 0, 0, 0, 0, 0] \ 81$$

$$\text{and } \text{reds}' :: J\text{-prog} \Rightarrow \text{expr list} \Rightarrow \text{state} \Rightarrow \text{bool} \Rightarrow \text{expr list} \Rightarrow \text{state} \Rightarrow \text{bool} \Rightarrow \text{bool}$$

$$(\leftarrow \vdash ((1 \langle -, /-, /- \rangle) [\rightarrow] / (1 \langle -, /-, /- \rangle))) \triangleright [51, 0, 0, 0, 0, 0, 0] \ 81$$

$$\text{for } P :: J\text{-prog}$$

**where**

$$P \vdash \langle e, s, b \rangle \rightarrow \langle e', s', b' \rangle \equiv ((e, s, b), e', s', b') \in \text{red } P$$

$$\mid P \vdash \langle es, s, b \rangle [\rightarrow] \langle es', s', b' \rangle \equiv ((es, s, b), es', s', b') \in \text{reds } P$$

*RedNew:*

$$\llbracket \text{new-Addr } h = \text{Some } a; P \vdash C \text{ has-fields } FDTs; h' = h(a \mapsto \text{blank } P \ C) \rrbracket$$

$$\implies P \vdash \langle \text{new } C, (h, l, sh), \text{True} \rangle \rightarrow \langle \text{addr } a, (h', l, sh), \text{False} \rangle$$

*RedNewFail:*

$$\llbracket \text{new-Addr } h = \text{None}; \text{is-class } P \ C \rrbracket \implies$$

$$P \vdash \langle \text{new } C, (h, l, sh), \text{True} \rangle \rightarrow \langle \text{THROW OutOfMemory}, (h, l, sh), \text{False} \rangle$$

*NewInitDoneRed:*

$$sh \ C = \text{Some } (sfs, \text{Done}) \implies$$

$$P \vdash \langle \text{new } C, (h, l, sh), \text{False} \rangle \rightarrow \langle \text{new } C, (h, l, sh), \text{True} \rangle$$

*NewInitRed:*

$$\llbracket \nexists sfs. sh \ C = \text{Some } (sfs, \text{Done}); \text{is-class } P \ C \rrbracket$$

$$\implies P \vdash \langle \text{new } C, (h, l, sh), \text{False} \rangle \rightarrow \langle \text{INIT } C \ ([C], \text{False}) \leftarrow \text{new } C, (h, l, sh), \text{False} \rangle$$

*CastRed:*

$$P \vdash \langle e, s, b \rangle \rightarrow \langle e', s', b' \rangle \implies$$

$$P \vdash \langle \text{Cast } C \ e, s, b \rangle \rightarrow \langle \text{Cast } C \ e', s', b' \rangle$$

*RedCastNull:*

$$P \vdash \langle \text{Cast } C \ \text{null}, s, b \rangle \rightarrow \langle \text{null}, s, b \rangle$$

*RedCast:*

- $\llbracket h a = \text{Some}(D,fs); P \vdash D \preceq^* C \rrbracket$   
 $\implies P \vdash \langle \text{Cast } C \text{ (addr } a), (h,l,sh), b \rangle \rightarrow \langle \text{addr } a, (h,l,sh), b \rangle$
- RedCastFail:*  
 $\llbracket h a = \text{Some}(D,fs); \neg P \vdash D \preceq^* C \rrbracket$   
 $\implies P \vdash \langle \text{Cast } C \text{ (addr } a), (h,l,sh), b \rangle \rightarrow \langle \text{THROW ClassCast}, (h,l,sh), b \rangle$
- BinOpRed1:*  
 $P \vdash \langle e,s,b \rangle \rightarrow \langle e',s',b' \rangle \implies$   
 $P \vdash \langle e \text{ «bop» } e_2, s, b \rangle \rightarrow \langle e' \text{ «bop» } e_2, s', b' \rangle$
- BinOpRed2:*  
 $P \vdash \langle e,s,b \rangle \rightarrow \langle e',s',b' \rangle \implies$   
 $P \vdash \langle (\text{Val } v_1) \text{ «bop» } e, s, b \rangle \rightarrow \langle (\text{Val } v_1) \text{ «bop» } e', s', b' \rangle$
- RedBinOp:*  
 $\text{binop}(bop,v_1,v_2) = \text{Some } v \implies$   
 $P \vdash \langle (\text{Val } v_1) \text{ «bop» } (\text{Val } v_2), s, b \rangle \rightarrow \langle \text{Val } v, s, b \rangle$
- RedVar:*  
 $l V = \text{Some } v \implies$   
 $P \vdash \langle \text{Var } V, (h,l,sh), b \rangle \rightarrow \langle \text{Val } v, (h,l,sh), b \rangle$
- LAssRed:*  
 $P \vdash \langle e,s,b \rangle \rightarrow \langle e',s',b' \rangle \implies$   
 $P \vdash \langle V:=e,s,b \rangle \rightarrow \langle V:=e',s',b' \rangle$
- RedLAss:*  
 $P \vdash \langle V:= (\text{Val } v), (h,l,sh), b \rangle \rightarrow \langle \text{unit}, (h,l(V \mapsto v),sh), b \rangle$
- FAccRed:*  
 $P \vdash \langle e,s,b \rangle \rightarrow \langle e',s',b' \rangle \implies$   
 $P \vdash \langle e \cdot F\{D\}, s, b \rangle \rightarrow \langle e' \cdot F\{D\}, s', b' \rangle$
- RedFAcc:*  
 $\llbracket h a = \text{Some}(C,fs); fs(F,D) = \text{Some } v;$   
 $\quad P \vdash C \text{ has } F, \text{NonStatic}:t \text{ in } D \rrbracket$   
 $\implies P \vdash \langle (\text{addr } a) \cdot F\{D\}, (h,l,sh), b \rangle \rightarrow \langle \text{Val } v, (h,l,sh), b \rangle$
- RedFAccNull:*  
 $P \vdash \langle \text{null} \cdot F\{D\}, s, b \rangle \rightarrow \langle \text{THROW NullPointer}, s, b \rangle$
- RedFAccNone:*  
 $\llbracket h a = \text{Some}(C,fs); \neg(\exists b t. P \vdash C \text{ has } F, b:t \text{ in } D) \rrbracket$   
 $\implies P \vdash \langle (\text{addr } a) \cdot F\{D\}, (h,l,sh), b \rangle \rightarrow \langle \text{THROW NoSuchFieldError}, (h,l,sh), b \rangle$
- RedFAccStatic:*  
 $\llbracket h a = \text{Some}(C,fs); P \vdash C \text{ has } F, \text{Static}:t \text{ in } D \rrbracket$   
 $\implies P \vdash \langle (\text{addr } a) \cdot F\{D\}, (h,l,sh), b \rangle \rightarrow \langle \text{THROW IncompatibleClassChangeError}, (h,l,sh), b \rangle$
- RedSFAcc:*  
 $\llbracket P \vdash C \text{ has } F, \text{Static}:t \text{ in } D;$   
 $\quad sh D = \text{Some } (sfs,i);$   
 $\quad sfs F = \text{Some } v \rrbracket$

$$\implies P \vdash \langle C \cdot_s F\{D\}, (h, l, sh), True \rangle \rightarrow \langle Val\ v, (h, l, sh), False \rangle$$

| *SFAccInitDoneRed*:

$$\begin{aligned} & \llbracket P \vdash C \text{ has } F, \text{Static}:t \text{ in } D; \\ & \quad sh\ D = \text{Some}(sfs, Done) \rrbracket \\ \implies & P \vdash \langle C \cdot_s F\{D\}, (h, l, sh), False \rangle \rightarrow \langle C \cdot_s F\{D\}, (h, l, sh), True \rangle \end{aligned}$$

| *SFAccInitRed*:

$$\begin{aligned} & \llbracket P \vdash C \text{ has } F, \text{Static}:t \text{ in } D; \\ & \quad \nexists sfs. sh\ D = \text{Some}(sfs, Done) \rrbracket \\ \implies & P \vdash \langle C \cdot_s F\{D\}, (h, l, sh), False \rangle \rightarrow \langle \text{INIT } D\ ([D], False) \leftarrow C \cdot_s F\{D\}, (h, l, sh), False \rangle \end{aligned}$$

| *RedSFAccNone*:

$$\begin{aligned} & \neg(\exists b\ t. P \vdash C \text{ has } F, b:t \text{ in } D) \\ \implies & P \vdash \langle C \cdot_s F\{D\}, (h, l, sh), b \rangle \rightarrow \langle \text{THROW } \text{NoSuchFieldError}, (h, l, sh), False \rangle \end{aligned}$$

| *RedSFAccNonStatic*:

$$\begin{aligned} & P \vdash C \text{ has } F, \text{NonStatic}:t \text{ in } D \\ \implies & P \vdash \langle C \cdot_s F\{D\}, (h, l, sh), b \rangle \rightarrow \langle \text{THROW } \text{IncompatibleClassChangeError}, (h, l, sh), False \rangle \end{aligned}$$

| *FAssRed1*:

$$\begin{aligned} & P \vdash \langle e, s, b \rangle \rightarrow \langle e', s', b' \rangle \implies \\ & P \vdash \langle e \cdot F\{D\} := e_2, s, b \rangle \rightarrow \langle e' \cdot F\{D\} := e_2, s', b' \rangle \end{aligned}$$

| *FAssRed2*:

$$\begin{aligned} & P \vdash \langle e, s, b \rangle \rightarrow \langle e', s', b' \rangle \implies \\ & P \vdash \langle Val\ v \cdot F\{D\} := e, s, b \rangle \rightarrow \langle Val\ v \cdot F\{D\} := e', s', b' \rangle \end{aligned}$$

| *RedFAss*:

$$\begin{aligned} & \llbracket P \vdash C \text{ has } F, \text{NonStatic}:t \text{ in } D; h\ a = \text{Some}(C, fs) \rrbracket \implies \\ & P \vdash \langle (addr\ a) \cdot F\{D\} := (Val\ v), (h, l, sh), b \rangle \rightarrow \langle \text{unit}, (h(a \mapsto (C, fs((F, D) \mapsto v))), l, sh), b \rangle \end{aligned}$$

| *RedFAssNull*:

$$P \vdash \langle \text{null} \cdot F\{D\} := Val\ v, s, b \rangle \rightarrow \langle \text{THROW } \text{NullPointer}, s, b \rangle$$

| *RedFAssNone*:

$$\begin{aligned} & \llbracket h\ a = \text{Some}(C, fs); \neg(\exists b\ t. P \vdash C \text{ has } F, b:t \text{ in } D) \rrbracket \\ \implies & P \vdash \langle (addr\ a) \cdot F\{D\} := (Val\ v), (h, l, sh), b \rangle \rightarrow \langle \text{THROW } \text{NoSuchFieldError}, (h, l, sh), b \rangle \end{aligned}$$

| *RedFAssStatic*:

$$\begin{aligned} & \llbracket h\ a = \text{Some}(C, fs); P \vdash C \text{ has } F, \text{Static}:t \text{ in } D \rrbracket \\ \implies & P \vdash \langle (addr\ a) \cdot F\{D\} := (Val\ v), (h, l, sh), b \rangle \rightarrow \langle \text{THROW } \text{IncompatibleClassChangeError}, (h, l, sh), b \rangle \end{aligned}$$

| *SFAssRed*:

$$\begin{aligned} & P \vdash \langle e, s, b \rangle \rightarrow \langle e', s', b' \rangle \implies \\ & P \vdash \langle C \cdot_s F\{D\} := e, s, b \rangle \rightarrow \langle C \cdot_s F\{D\} := e', s', b' \rangle \end{aligned}$$

| *RedSFAss*:

$$\begin{aligned} & \llbracket P \vdash C \text{ has } F, \text{Static}:t \text{ in } D; \\ & \quad sh\ D = \text{Some}(sfs, i); \\ & \quad sfs' = sfs(F \mapsto v); sh' = sh(D \mapsto (sfs', i)) \rrbracket \\ \implies & P \vdash \langle C \cdot_s F\{D\} := (Val\ v), (h, l, sh), True \rangle \rightarrow \langle \text{unit}, (h, l, sh'), False \rangle \end{aligned}$$

| *SFAssInitDoneRed*:

$$\begin{aligned} & \llbracket P \vdash C \text{ has } F, \text{Static}:t \text{ in } D; \\ & \quad sh D = \text{Some}(sfs, \text{Done}) \rrbracket \\ & \implies P \vdash \langle C \cdot_s F\{D\} := (\text{Val } v), (h, l, sh), \text{False} \rangle \rightarrow \langle C \cdot_s F\{D\} := (\text{Val } v), (h, l, sh), \text{True} \rangle \end{aligned}$$

| *SFAssInitRed:*

$$\begin{aligned} & \llbracket P \vdash C \text{ has } F, \text{Static}:t \text{ in } D; \\ & \quad \nexists sfs. sh D = \text{Some}(sfs, \text{Done}) \rrbracket \\ & \implies P \vdash \langle C \cdot_s F\{D\} := (\text{Val } v), (h, l, sh), \text{False} \rangle \rightarrow \langle \text{INIT } D ([D], \text{False}) \leftarrow C \cdot_s F\{D\} := (\text{Val } v), (h, l, sh), \text{False} \rangle \end{aligned}$$

| *RedSFAssNone:*

$$\begin{aligned} & \neg(\exists b t. P \vdash C \text{ has } F, b:t \text{ in } D) \\ & \implies P \vdash \langle C \cdot_s F\{D\} := (\text{Val } v), s, b \rangle \rightarrow \langle \text{THROW } \text{NoSuchFieldError}, s, \text{False} \rangle \end{aligned}$$

| *RedSFAssNonStatic:*

$$\begin{aligned} & P \vdash C \text{ has } F, \text{NonStatic}:t \text{ in } D \\ & \implies P \vdash \langle C \cdot_s F\{D\} := (\text{Val } v), s, b \rangle \rightarrow \langle \text{THROW } \text{IncompatibleClassChangeError}, s, \text{False} \rangle \end{aligned}$$

| *CallObj:*

$$\begin{aligned} & P \vdash \langle e, s, b \rangle \rightarrow \langle e', s', b' \rangle \implies \\ & P \vdash \langle e \cdot M(es), s, b \rangle \rightarrow \langle e' \cdot M(es), s', b' \rangle \end{aligned}$$

| *CallParams:*

$$\begin{aligned} & P \vdash \langle es, s, b \rangle [\rightarrow] \langle es', s', b' \rangle \implies \\ & P \vdash \langle (\text{Val } v) \cdot M(es), s, b \rangle \rightarrow \langle (\text{Val } v) \cdot M(es'), s', b' \rangle \end{aligned}$$

| *RedCall:*

$$\begin{aligned} & \llbracket h a = \text{Some}(C, fs); P \vdash C \text{ sees } M, \text{NonStatic}:Ts \rightarrow T = (pns, body) \text{ in } D; \text{ size } vs = \text{size } pns; \text{ size } Ts \\ & = \text{size } pns \rrbracket \\ & \implies P \vdash \langle (addr a) \cdot M(\text{map } \text{Val } vs), (h, l, sh), b \rangle \rightarrow \langle \text{blocks}(\text{this}\#pns, \text{Class } D\#Ts, \text{Addr } a\#vs, body), \\ & (h, l, sh), b \rangle \end{aligned}$$

| *RedCallNull:*

$$P \vdash \langle \text{null} \cdot M(\text{map } \text{Val } vs), s, b \rangle \rightarrow \langle \text{THROW } \text{NullPointer}, s, b \rangle$$

| *RedCallNone:*

$$\begin{aligned} & \llbracket h a = \text{Some}(C, fs); \neg(\exists b Ts T m D. P \vdash C \text{ sees } M, b:Ts \rightarrow T = m \text{ in } D) \rrbracket \\ & \implies P \vdash \langle (addr a) \cdot M(\text{map } \text{Val } vs), (h, l, sh), b \rangle \rightarrow \langle \text{THROW } \text{NoSuchMethodError}, (h, l, sh), b \rangle \end{aligned}$$

| *RedCallStatic:*

$$\begin{aligned} & \llbracket h a = \text{Some}(C, fs); P \vdash C \text{ sees } M, \text{Static}:Ts \rightarrow T = m \text{ in } D \rrbracket \\ & \implies P \vdash \langle (addr a) \cdot M(\text{map } \text{Val } vs), (h, l, sh), b \rangle \rightarrow \langle \text{THROW } \text{IncompatibleClassChangeError}, (h, l, sh), b \rangle \end{aligned}$$

| *SCallParams:*

$$\begin{aligned} & P \vdash \langle es, s, b \rangle [\rightarrow] \langle es', s', b' \rangle \implies \\ & P \vdash \langle C \cdot_s M(es), s, b \rangle \rightarrow \langle C \cdot_s M(es'), s', b' \rangle \end{aligned}$$

| *RedSCall:*

$$\begin{aligned} & \llbracket P \vdash C \text{ sees } M, \text{Static}:Ts \rightarrow T = (pns, body) \text{ in } D; \\ & \quad \text{length } vs = \text{length } pns; \text{ size } Ts = \text{size } pns \rrbracket \\ & \implies P \vdash \langle C \cdot_s M(\text{map } \text{Val } vs), s, \text{True} \rangle \rightarrow \langle \text{blocks}(pns, Ts, vs, body), s, \text{False} \rangle \end{aligned}$$

| *SCallInitDoneRed:*

$$\begin{aligned} & \llbracket P \vdash C \text{ sees } M, \text{Static}:Ts \rightarrow T = (pns, body) \text{ in } D; \\ & \quad sh D = \text{Some}(sfs, \text{Done}) \vee (M = \text{clinit} \wedge sh D = \text{Some}(sfs, \text{Processing})) \rrbracket \end{aligned}$$

$$\implies P \vdash \langle C \cdot_s M(\text{map Val } vs), (h, l, sh), \text{False} \rangle \rightarrow \langle C \cdot_s M(\text{map Val } vs), (h, l, sh), \text{True} \rangle$$

| *SCallInitRed:*

$$\llbracket P \vdash C \text{ sees } M, \text{Static}: Ts \rightarrow T = (pns, body) \text{ in } D; \\ \nexists sfs. sh \ D = \text{Some}(sfs, Done); M \neq \text{clinit} \rrbracket$$

$$\implies P \vdash \langle C \cdot_s M(\text{map Val } vs), (h, l, sh), \text{False} \rangle \rightarrow \langle \text{INIT } D ([D], \text{False}) \leftarrow C \cdot_s M(\text{map Val } vs), (h, l, sh), \text{False} \rangle$$

| *RedSCallNone:*

$$\llbracket \neg(\exists b \ Ts \ T \ m \ D. P \vdash C \text{ sees } M, b: Ts \rightarrow T = m \text{ in } D) \rrbracket$$

$$\implies P \vdash \langle C \cdot_s M(\text{map Val } vs), s, b \rangle \rightarrow \langle \text{THROW } \text{NoSuchMethodError}, s, \text{False} \rangle$$

| *RedSCallNonStatic:*

$$\llbracket P \vdash C \text{ sees } M, \text{NonStatic}: Ts \rightarrow T = m \text{ in } D \rrbracket$$

$$\implies P \vdash \langle C \cdot_s M(\text{map Val } vs), s, b \rangle \rightarrow \langle \text{THROW } \text{IncompatibleClassChangeError}, s, \text{False} \rangle$$

| *BlockRedNone:*

$$\llbracket P \vdash \langle e, (h, l(V := \text{None}), sh), b \rangle \rightarrow \langle e', (h', l', sh'), b' \rangle; l' V = \text{None}; \neg \text{assigned } V \ e \rrbracket$$

$$\implies P \vdash \langle \{V:T; e\}, (h, l, sh), b \rangle \rightarrow \langle \{V:T; e'\}, (h', l'(V := l V), sh'), b' \rangle$$

| *BlockRedSome:*

$$\llbracket P \vdash \langle e, (h, l(V := \text{None}), sh), b \rangle \rightarrow \langle e', (h', l', sh'), b' \rangle; l' V = \text{Some } v; \neg \text{assigned } V \ e \rrbracket$$

$$\implies P \vdash \langle \{V:T; e\}, (h, l, sh), b \rangle \rightarrow \langle \{V:T := \text{Val } v; e'\}, (h', l'(V := l V), sh'), b' \rangle$$

| *InitBlockRed:*

$$\llbracket P \vdash \langle e, (h, l(V \mapsto v), sh), b \rangle \rightarrow \langle e', (h', l', sh'), b' \rangle; l' V = \text{Some } v' \rrbracket$$

$$\implies P \vdash \langle \{V:T := \text{Val } v; e\}, (h, l, sh), b \rangle \rightarrow \langle \{V:T := \text{Val } v'; e'\}, (h', l'(V := l V), sh'), b' \rangle$$

| *RedBlock:*

$$P \vdash \langle \{V:T; \text{Val } u\}, s, b \rangle \rightarrow \langle \text{Val } u, s, b \rangle$$

| *RedInitBlock:*

$$P \vdash \langle \{V:T := \text{Val } v; \text{Val } u\}, s, b \rangle \rightarrow \langle \text{Val } u, s, b \rangle$$

| *SeqRed:*

$$P \vdash \langle e, s, b \rangle \rightarrow \langle e', s', b' \rangle \implies$$

$$P \vdash \langle e;; e_2, s, b \rangle \rightarrow \langle e';; e_2, s', b' \rangle$$

| *RedSeq:*

$$P \vdash \langle (\text{Val } v);; e_2, s, b \rangle \rightarrow \langle e_2, s, b \rangle$$

| *CondRed:*

$$P \vdash \langle e, s, b \rangle \rightarrow \langle e', s', b' \rangle \implies$$

$$P \vdash \langle \text{if } (e) \ e_1 \ \text{else } e_2, s, b \rangle \rightarrow \langle \text{if } (e') \ e_1 \ \text{else } e_2, s', b' \rangle$$

| *RedCondT:*

$$P \vdash \langle \text{if } (\text{true}) \ e_1 \ \text{else } e_2, s, b \rangle \rightarrow \langle e_1, s, b \rangle$$

| *RedCondF:*

$$P \vdash \langle \text{if } (\text{false}) \ e_1 \ \text{else } e_2, s, b \rangle \rightarrow \langle e_2, s, b \rangle$$

| *RedWhile:*

$$P \vdash \langle \text{while}(b) \ c, s, b' \rangle \rightarrow \langle \text{if}(b) \ (c;; \text{while}(b) \ c) \ \text{else } \text{unit}, s, b' \rangle$$

| *ThrowRed:*



$$P \vdash \langle e, s, b \rangle \rightarrow \langle e', s', b' \rangle \implies \\ P \vdash \langle \text{throw } e, s, b \rangle \rightarrow \langle \text{throw } e', s', b' \rangle$$

$$| \text{RedThrowNull:} \\ P \vdash \langle \text{throw null}, s, b \rangle \rightarrow \langle \text{THROW NullPointer}, s, b \rangle$$

$$| \text{TryRed:} \\ P \vdash \langle e, s, b \rangle \rightarrow \langle e', s', b' \rangle \implies \\ P \vdash \langle \text{try } e \text{ catch}(C \ V) \ e_2, s, b \rangle \rightarrow \langle \text{try } e' \text{ catch}(C \ V) \ e_2, s', b' \rangle$$

$$| \text{RedTry:} \\ P \vdash \langle \text{try } (\text{Val } v) \text{ catch}(C \ V) \ e_2, s, b \rangle \rightarrow \langle \text{Val } v, s, b \rangle$$

$$| \text{RedTryCatch:} \\ \llbracket \text{hp } s \ a = \text{Some}(D, \text{fs}); P \vdash D \preceq^* C \rrbracket \\ \implies P \vdash \langle \text{try } (\text{Throw } a) \text{ catch}(C \ V) \ e_2, s, b \rangle \rightarrow \langle \{V: \text{Class } C := \text{addr } a; e_2\}, s, b \rangle$$

$$| \text{RedTryFail:} \\ \llbracket \text{hp } s \ a = \text{Some}(D, \text{fs}); \neg P \vdash D \preceq^* C \rrbracket \\ \implies P \vdash \langle \text{try } (\text{Throw } a) \text{ catch}(C \ V) \ e_2, s, b \rangle \rightarrow \langle \text{Throw } a, s, b \rangle$$

$$| \text{ListRed1:} \\ P \vdash \langle e, s, b \rangle \rightarrow \langle e', s', b' \rangle \implies \\ P \vdash \langle e \# es, s, b \rangle [\rightarrow] \langle e' \# es, s', b' \rangle$$

$$| \text{ListRed2:} \\ P \vdash \langle es, s, b \rangle [\rightarrow] \langle es', s', b' \rangle \implies \\ P \vdash \langle \text{Val } v \ \# \ es, s, b \rangle [\rightarrow] \langle \text{Val } v \ \# \ es', s', b' \rangle$$

— Initialization procedure

$$| \text{RedInit:} \\ \neg \text{sub-RI } e \implies P \vdash \langle \text{INIT } C \ (\text{Nil}, b) \leftarrow e, s, b' \rangle \rightarrow \langle e, s, \text{icheck } P \ C \ e \rangle$$

$$| \text{InitNoneRed:} \\ \text{sh } C = \text{None} \\ \implies P \vdash \langle \text{INIT } C' \ (C \# Cs, \text{False}) \leftarrow e, (h, l, \text{sh}), b \rangle \rightarrow \langle \text{INIT } C' \ (C \# Cs, \text{False}) \leftarrow e, (h, l, \text{sh}(C \mapsto (\text{sblank } P \ C, \text{Prepared}))), b \rangle$$

$$| \text{RedInitDone:} \\ \text{sh } C = \text{Some}(sfs, \text{Done}) \\ \implies P \vdash \langle \text{INIT } C' \ (C \# Cs, \text{False}) \leftarrow e, (h, l, \text{sh}), b \rangle \rightarrow \langle \text{INIT } C' \ (Cs, \text{True}) \leftarrow e, (h, l, \text{sh}), b \rangle$$

$$| \text{RedInitProcessing:} \\ \text{sh } C = \text{Some}(sfs, \text{Processing}) \\ \implies P \vdash \langle \text{INIT } C' \ (C \# Cs, \text{False}) \leftarrow e, (h, l, \text{sh}), b \rangle \rightarrow \langle \text{INIT } C' \ (Cs, \text{True}) \leftarrow e, (h, l, \text{sh}), b \rangle$$

$$| \text{RedInitError:} \\ \text{sh } C = \text{Some}(sfs, \text{Error}) \\ \implies P \vdash \langle \text{INIT } C' \ (C \# Cs, \text{False}) \leftarrow e, (h, l, \text{sh}), b \rangle \rightarrow \langle \text{RI } (C, \text{THROW NoClassDefFoundError}); Cs \leftarrow e, (h, l, \text{sh}), b \rangle$$

$$| \text{InitObjectRed:} \\ \llbracket \text{sh } C = \text{Some}(sfs, \text{Prepared});$$

$$\begin{aligned}
& C = \text{Object}; \\
& sh' = sh(C \mapsto (sfs, \text{Processing})) \ ] \\
\implies P \vdash \langle \text{INIT } C' (C \# Cs, \text{False}) \leftarrow e, (h, l, sh), b \rangle \rightarrow \langle \text{INIT } C' (C \# Cs, \text{True}) \leftarrow e, (h, l, sh'), b \rangle
\end{aligned}$$

| *InitNonObjectSuperRed*:

$$\begin{aligned}
& \llbracket sh \ C = \text{Some}(sfs, \text{Prepared}); \\
& \quad C \neq \text{Object}; \\
& \quad \text{class } P \ C = \text{Some} (D, r); \\
& \quad sh' = sh(C \mapsto (sfs, \text{Processing})) \ ] \\
\implies P \vdash \langle \text{INIT } C' (C \# Cs, \text{False}) \leftarrow e, (h, l, sh), b \rangle \rightarrow \langle \text{INIT } C' (D \# C \# Cs, \text{False}) \leftarrow e, (h, l, sh'), b \rangle
\end{aligned}$$

| *RedInitRInit*:

$$P \vdash \langle \text{INIT } C' (C \# Cs, \text{True}) \leftarrow e, (h, l, sh), b \rangle \rightarrow \langle \text{RI } (C, C \cdot_s \text{clinit}(\ [])); Cs \leftarrow e, (h, l, sh), b \rangle$$

| *RInitRed*:

$$\begin{aligned}
& P \vdash \langle e, s, b \rangle \rightarrow \langle e', s', b' \rangle \implies \\
& P \vdash \langle \text{RI } (C, e); Cs \leftarrow e_0, s, b \rangle \rightarrow \langle \text{RI } (C, e'); Cs \leftarrow e_0, s', b' \rangle
\end{aligned}$$

| *RedRInit*:

$$\begin{aligned}
& \llbracket sh \ C = \text{Some} (sfs, i); \\
& \quad sh' = sh(C \mapsto (sfs, \text{Done})); \\
& \quad C' = \text{last}(C \# Cs) \ ] \implies \\
& P \vdash \langle \text{RI } (C, \text{Val } v); Cs \leftarrow e, (h, l, sh), b \rangle \rightarrow \langle \text{INIT } C' (Cs, \text{True}) \leftarrow e, (h, l, sh'), b \rangle
\end{aligned}$$

— Exception propagation

$$\begin{aligned}
& | \text{CastThrow}: P \vdash \langle \text{Cast } C (\text{throw } e), s, b \rangle \rightarrow \langle \text{throw } e, s, b \rangle \\
& | \text{BinOpThrow1}: P \vdash \langle (\text{throw } e) \llbracket \text{bop} \rrbracket e_2, s, b \rangle \rightarrow \langle \text{throw } e, s, b \rangle \\
& | \text{BinOpThrow2}: P \vdash \langle (\text{Val } v_1) \llbracket \text{bop} \rrbracket (\text{throw } e), s, b \rangle \rightarrow \langle \text{throw } e, s, b \rangle \\
& | \text{LAssThrow}: P \vdash \langle V := (\text{throw } e), s, b \rangle \rightarrow \langle \text{throw } e, s, b \rangle \\
& | \text{FAccThrow}: P \vdash \langle (\text{throw } e) \cdot F\{D\}, s, b \rangle \rightarrow \langle \text{throw } e, s, b \rangle \\
& | \text{FAssThrow1}: P \vdash \langle (\text{throw } e) \cdot F\{D\} := e_2, s, b \rangle \rightarrow \langle \text{throw } e, s, b \rangle \\
& | \text{FAssThrow2}: P \vdash \langle \text{Val } v \cdot F\{D\} := (\text{throw } e), s, b \rangle \rightarrow \langle \text{throw } e, s, b \rangle \\
& | \text{SFAssThrow}: P \vdash \langle C \cdot_s F\{D\} := (\text{throw } e), s, b \rangle \rightarrow \langle \text{throw } e, s, b \rangle \\
& | \text{CallThrowObj}: P \vdash \langle (\text{throw } e) \cdot M(es), s, b \rangle \rightarrow \langle \text{throw } e, s, b \rangle \\
& | \text{CallThrowParams}: \llbracket es = \text{map } \text{Val } vs \ @ \ \text{throw } e \ \# \ es' \ ] \implies P \vdash \langle (\text{Val } v) \cdot M(es), s, b \rangle \rightarrow \langle \text{throw } e, s, b \rangle \\
& | \text{SCallThrowParams}: \llbracket es = \text{map } \text{Val } vs \ @ \ \text{throw } e \ \# \ es' \ ] \implies P \vdash \langle C \cdot_s M(es), s, b \rangle \rightarrow \langle \text{throw } e, s, b \rangle \\
& | \text{BlockThrow}: P \vdash \langle \{ V:T; \text{Throw } a \}, s, b \rangle \rightarrow \langle \text{Throw } a, s, b \rangle \\
& | \text{InitBlockThrow}: P \vdash \langle \{ V:T := \text{Val } v; \text{Throw } a \}, s, b \rangle \rightarrow \langle \text{Throw } a, s, b \rangle \\
& | \text{SeqThrow}: P \vdash \langle (\text{throw } e); e_2, s, b \rangle \rightarrow \langle \text{throw } e, s, b \rangle \\
& | \text{CondThrow}: P \vdash \langle \text{if } (\text{throw } e) \ e_1 \ \text{else } e_2, s, b \rangle \rightarrow \langle \text{throw } e, s, b \rangle \\
& | \text{ThrowThrow}: P \vdash \langle \text{throw}(\text{throw } e), s, b \rangle \rightarrow \langle \text{throw } e, s, b \rangle \\
& | \text{RInitInitThrow}: \llbracket sh \ C = \text{Some}(sfs, i); sh' = sh(C \mapsto (sfs, \text{Error})) \ ] \implies \\
& \quad P \vdash \langle \text{RI } (C, \text{Throw } a); D \# Cs \leftarrow e, (h, l, sh), b \rangle \rightarrow \langle \text{RI } (D, \text{Throw } a); Cs \leftarrow e, (h, l, sh'), b \rangle \\
& | \text{RInitThrow}: \llbracket sh \ C = \text{Some}(sfs, i); sh' = sh(C \mapsto (sfs, \text{Error})) \ ] \implies \\
& \quad P \vdash \langle \text{RI } (C, \text{Throw } a); \text{Nil} \leftarrow e, (h, l, sh), b \rangle \rightarrow \langle \text{Throw } a, (h, l, sh'), b \rangle
\end{aligned}$$

### 1.18.1 The reflexive transitive closure

abbreviation

$$\begin{aligned}
& \text{Step} :: J\text{-prog} \Rightarrow \text{expr} \Rightarrow \text{state} \Rightarrow \text{bool} \Rightarrow \text{expr} \Rightarrow \text{state} \Rightarrow \text{bool} \Rightarrow \text{bool} \\
& \quad (\leftarrow \vdash \ ((1 \langle -, /-, /- \rangle) \rightarrow^* / (1 \langle -, /-, /- \rangle))) \triangleright [51, 0, 0, 0, 0, 0, 0] \ 81)
\end{aligned}$$

where  $P \vdash \langle e, s, b \rangle \rightarrow^* \langle e', s', b' \rangle \equiv ((e, s, b), e', s', b') \in (\text{red } P)^*$

### abbreviation

$\text{Steps} :: J\text{-prog} \Rightarrow \text{expr list} \Rightarrow \text{state} \Rightarrow \text{bool} \Rightarrow \text{expr list} \Rightarrow \text{state} \Rightarrow \text{bool} \Rightarrow \text{bool}$   
 $(\vdash \vdash ((1 \langle -, /-, /- \rangle) [\rightarrow]^* / (1 \langle -, /-, /- \rangle)) \triangleright [51, 0, 0, 0, 0, 0, 0] 81)$

where  $P \vdash \langle es, s, b \rangle [\rightarrow]^* \langle es', s', b' \rangle \equiv ((es, s, b), es', s', b') \in (\text{reds } P)^*$

**lemmas** *converse-rtrancl-induct3* =

*converse-rtrancl-induct* [of  $(ax, ay, az)$   $(bx, by, bz)$ , *split-format* (complete),  
*consumes 1*, *case-names refl step*]

**lemma** *converse-rtrancl-induct-red*[*consumes 1*]:

**assumes**  $P \vdash \langle e, (h, l, sh), b \rangle \rightarrow^* \langle e', (h', l', sh'), b' \rangle$

**and**  $\bigwedge e h l sh b. R e h l sh b e h l sh b$

**and**  $\bigwedge e_0 h_0 l_0 sh_0 b_0 e_1 h_1 l_1 sh_1 b_1 e' h' l' sh' b'$ .

$\llbracket P \vdash \langle e_0, (h_0, l_0, sh_0), b_0 \rangle \rightarrow \langle e_1, (h_1, l_1, sh_1), b_1 \rangle; R e_1 h_1 l_1 sh_1 b_1 e' h' l' sh' b' \rrbracket$   
 $\implies R e_0 h_0 l_0 sh_0 b_0 e' h' l' sh' b'$

**shows**  $R e h l sh b e' h' l' sh' b'$ *<proof>*

## 1.18.2 Some easy lemmas

**lemma** [*iff*]:  $\neg P \vdash \langle [], s, b \rangle [\rightarrow] \langle es', s', b' \rangle$ *<proof>*

**lemma** [*iff*]:  $\neg P \vdash \langle \text{Val } v, s, b \rangle \rightarrow \langle e', s', b' \rangle$ *<proof>*

**lemma** *val-no-step*:  $\text{val-of } e = \lfloor v \rfloor \implies \neg P \vdash \langle e, s, b \rangle \rightarrow \langle e', s', b' \rangle$ *<proof>*

**lemma** [*iff*]:  $\neg P \vdash \langle \text{Throw } a, s, b \rangle \rightarrow \langle e', s', b' \rangle$ *<proof>*

**lemma** *map-Vals-no-step* [*iff*]:  $\neg P \vdash \langle \text{map Val } vs, s, b \rangle [\rightarrow] \langle es', s', b' \rangle$ *<proof>*

**lemma** *vals-no-step*:  $\text{map-vals-of } es = \lfloor vs \rfloor \implies \neg P \vdash \langle es, s, b \rangle [\rightarrow] \langle es', s', b' \rangle$ *<proof>*

**lemma** *vals-throw-no-step* [*iff*]:  $\neg P \vdash \langle \text{map Val } vs @ \text{Throw } a \# es, s, b \rangle [\rightarrow] \langle es', s', b' \rangle$ *<proof>*

**lemma** *lass-val-of-red*:

$\llbracket \text{lass-val-of } e = \lfloor a \rfloor; P \vdash \langle e, (h, l, sh), b \rangle \rightarrow \langle e', (h', l', sh'), b' \rangle \rrbracket$   
 $\implies e' = \text{unit} \wedge h' = h \wedge l' = l(\text{fst } a \rightarrow \text{snd } a) \wedge sh' = sh \wedge b = b'$ *<proof>*

**lemma** *final-no-step* [*iff*]:  $\text{final } e \implies \neg P \vdash \langle e, s, b \rangle \rightarrow \langle e', s', b' \rangle$ *<proof>*

**lemma** *finals-no-step* [*iff*]:  $\text{finals } es \implies \neg P \vdash \langle es, s, b \rangle [\rightarrow] \langle es', s', b' \rangle$ *<proof>*

**lemma** *reds-final-same*:

$P \vdash \langle e, s, b \rangle \rightarrow^* \langle e', s', b' \rangle \implies \text{final } e \implies e = e' \wedge s = s' \wedge b = b'$   
*<proof>*

**lemma** *reds-throw*:

$P \vdash \langle e, s, b \rangle \rightarrow^* \langle e', s', b' \rangle \implies (\bigwedge e_t. \text{throw-of } e = \lfloor e_t \rfloor \implies \exists e'_t. \text{throw-of } e' = \lfloor e'_t \rfloor)$   
*<proof>*

**lemma** *red-hext-incr*:  $P \vdash \langle e, (h, l, sh), b \rangle \rightarrow \langle e', (h', l', sh'), b' \rangle \implies h \trianglelefteq h'$

**and** *reds-hext-incr*:  $P \vdash \langle es, (h, l, sh), b \rangle [\rightarrow] \langle es', (h', l', sh'), b' \rangle \implies h \trianglelefteq h'$ *<proof>*

**lemma** *red-lcl-incr*:  $P \vdash \langle e, (h_0, l_0, sh_0), b \rangle \rightarrow \langle e', (h_1, l_1, sh_1), b' \rangle \implies \text{dom } l_0 \subseteq \text{dom } l_1$

**and** *reds-lcl-incr*:  $P \vdash \langle es, (h_0, l_0, sh_0), b \rangle [\rightarrow] \langle es', (h_1, l_1, sh_1), b' \rangle \implies \text{dom } l_0 \subseteq \text{dom } l_1$ *<proof>*

**lemma** *red-lcl-add*:  $P \vdash \langle e, (h, l, sh), b \rangle \rightarrow \langle e', (h', l', sh'), b' \rangle \implies (\bigwedge l_0. P \vdash \langle e, (h, l_0 ++ l, sh), b \rangle \rightarrow \langle e', (h', l_0 ++ l', sh'), b' \rangle)$

**and** *reds-lcl-add*:  $P \vdash \langle es, (h, l, sh), b \rangle [\rightarrow] \langle es', (h', l', sh'), b' \rangle \implies (\bigwedge l_0. P \vdash \langle es, (h, l_0 ++ l, sh), b \rangle [\rightarrow] \langle es', (h', l_0 ++ l', sh'), b' \rangle)$ *<proof>*

**lemma** *Red-lcl-add*:

**assumes**  $P \vdash \langle e, (h, l, sh), b \rangle \rightarrow^* \langle e', (h', l', sh'), b' \rangle$  **shows**  $P \vdash \langle e, (h, l_0 ++ l, sh), b \rangle \rightarrow^* \langle e', (h', l_0 ++ l', sh'), b' \rangle$  *<proof>*  
**lemma assumes** *wf: wwf-J-prog P*  
**shows** *red-proc-pres: P*  $\vdash \langle e, (h, l, sh), b \rangle \rightarrow \langle e', (h', l', sh'), b' \rangle$   
 $\implies$  *not-init C e*  $\implies$  *sh C*  $= \llbracket (sfs, Processing) \rrbracket \implies$  *not-init C e'*  $\wedge (\exists sfs'. sh' C = \llbracket (sfs', Processing) \rrbracket)$   
**and** *reds-proc-pres: P*  $\vdash \langle es, (h, l, sh), b \rangle [\rightarrow] \langle es', (h', l', sh'), b' \rangle$   
 $\implies$  *not-inits C es*  $\implies$  *sh C*  $= \llbracket (sfs, Processing) \rrbracket \implies$  *not-inits C es'*  $\wedge (\exists sfs'. sh' C = \llbracket (sfs', Processing) \rrbracket)$  *<proof>*

## 1.19 Expression conformance properties

**theory** *EConform*  
**imports** *SmallStep BigStep*  
**begin**

**lemma** *cons-to-append: list*  $\neq [] \longrightarrow (\exists ls. a \# list = ls @ [last\ list])$   
*<proof>*

### 1.19.1 Initialization conformance

**fun** *init-class*  $:: 'm\ prog \Rightarrow 'a\ exp \Rightarrow cname\ option$  **where**  
*init-class P* (*new C*)  $= Some\ C$  |  
*init-class P* ( $C \cdot_s F\{D\}$ )  $= Some\ D$  |  
*init-class P* ( $C \cdot_s F\{D\} := e_2$ )  $= Some\ D$  |  
*init-class P* ( $C \cdot_s M(es)$ )  $= seeing-class\ P\ C\ M$  |  
*init-class - -*  $= None$

**lemma** *icheck-init-class: icheck P C e*  $\implies$  *init-class P e*  $= \llbracket C \rrbracket$   
*<proof>*

**fun** *ss-exp*  $:: 'a\ exp \Rightarrow 'a\ exp$  **and** *ss-exps*  $:: 'a\ exp\ list \Rightarrow 'a\ exp\ option$  **where**  
*ss-exp* (*new C*)  $= new\ C$   
| *ss-exp* (*Cast C e*)  $= (case\ val-of\ e\ of\ Some\ v \Rightarrow Cast\ C\ e \mid - \Rightarrow ss-exp\ e)$   
| *ss-exp* (*Val v*)  $= Val\ v$   
| *ss-exp* ( $e_1 \llcorner bop \rceil e_2$ )  $= (case\ val-of\ e_1\ of\ Some\ v \Rightarrow (case\ val-of\ e_2\ of\ Some\ v \Rightarrow e_1 \llcorner bop \rceil e_2 \mid - \Rightarrow ss-exp\ e_2) \mid - \Rightarrow ss-exp\ e_1)$   
| *ss-exp* (*Var V*)  $= Var\ V$   
| *ss-exp* (*LAss V e*)  $= (case\ val-of\ e\ of\ Some\ v \Rightarrow LAss\ V\ e \mid - \Rightarrow ss-exp\ e)$   
| *ss-exp* ( $e \cdot F\{D\}$ )  $= (case\ val-of\ e\ of\ Some\ v \Rightarrow e \cdot F\{D\} \mid - \Rightarrow ss-exp\ e)$   
| *ss-exp* ( $C \cdot_s F\{D\}$ )  $= C \cdot_s F\{D\}$   
| *ss-exp* ( $e_1 \cdot F\{D\} := e_2$ )  $= (case\ val-of\ e_1\ of\ Some\ v \Rightarrow (case\ val-of\ e_2\ of\ Some\ v \Rightarrow e_1 \cdot F\{D\} := e_2 \mid - \Rightarrow ss-exp\ e_2) \mid - \Rightarrow ss-exp\ e_1)$   
| *ss-exp* ( $C \cdot_s F\{D\} := e_2$ )  $= (case\ val-of\ e_2\ of\ Some\ v \Rightarrow C \cdot_s F\{D\} := e_2 \mid - \Rightarrow ss-exp\ e_2)$   
| *ss-exp* ( $e \cdot M(es)$ )  $= (case\ val-of\ e\ of\ Some\ v \Rightarrow (case\ map-vals-of\ es\ of\ Some\ t \Rightarrow e \cdot M(es) \mid - \Rightarrow the(ss-exps\ es)) \mid - \Rightarrow ss-exp\ e)$   
| *ss-exp* ( $C \cdot_s M(es)$ )  $= (case\ map-vals-of\ es\ of\ Some\ t \Rightarrow C \cdot_s M(es) \mid - \Rightarrow the(ss-exps\ es))$   
| *ss-exp* ( $\{V:T; e\}$ )  $= ss-exp\ e$   
| *ss-exp* ( $e_1;; e_2$ )  $= (case\ val-of\ e_1\ of\ Some\ v \Rightarrow ss-exp\ e_2 \mid None \Rightarrow (case\ lass-val-of\ e_1\ of\ Some\ p \Rightarrow ss-exp\ e_2 \mid None \Rightarrow ss-exp\ e_1))$   
| *ss-exp* (*if* (*b*)  $e_1$  *else*  $e_2$ )  $= (case\ bool-of\ b\ of\ Some\ True \Rightarrow if\ (b)\ e_1\ else\ e_2 \mid Some\ False \Rightarrow if\ (b)\ e_1\ else\ e_2)$

$$\begin{aligned}
& | - \Rightarrow \text{ss-exp } b) \\
| \text{ss-exp } (\text{while } (b) e) &= \text{while } (b) e \\
| \text{ss-exp } (\text{throw } e) &= (\text{case val-of } e \text{ of Some } v \Rightarrow \text{throw } e \mid - \Rightarrow \text{ss-exp } e) \\
| \text{ss-exp } (\text{try } e_1 \text{ catch}(C V) e_2) &= (\text{case val-of } e_1 \text{ of Some } v \Rightarrow \text{try } e_1 \text{ catch}(C V) e_2 \\
& \quad | - \Rightarrow \text{ss-exp } e_1) \\
| \text{ss-exp } (\text{INIT } C (Cs, b) \leftarrow e) &= \text{INIT } C (Cs, b) \leftarrow e \\
| \text{ss-exp } (\text{RI } (C, e); Cs \leftarrow e') &= (\text{case val-of } e \text{ of Some } v \Rightarrow \text{RI } (C, e); Cs \leftarrow e \mid - \Rightarrow \text{ss-exp } e) \\
| \text{ss-exps}(\[]) &= \text{None} \\
| \text{ss-exps}(e\#es) &= (\text{case val-of } e \text{ of Some } v \Rightarrow \text{ss-exps } es \mid - \Rightarrow \text{Some } (\text{ss-exp } e))
\end{aligned}$$

**lemma** *icheck-ss-exp*:

**assumes** *icheck*  $P C e$  **shows**  $\text{ss-exp } e = e$

*<proof>*

**lemma** *ss-exps-Vals-None[simp]*:

$\text{ss-exps } (\text{map Val } vs) = \text{None}$

*<proof>*

**lemma** *ss-exps-Vals-NoneI*:

$\text{ss-exps } es = \text{None} \Longrightarrow \exists vs. es = \text{map Val } vs$

*<proof>*

**lemma** *ss-exps-throw-nVal*:

$\llbracket \text{val-of } e = \text{None}; \text{ss-exps } (\text{map Val } vs @ \text{throw } e \# es') = \llbracket e' \rrbracket \rrbracket$   
 $\Longrightarrow e' = \text{ss-exp } e$

*<proof>*

**lemma** *ss-exps-throw-Val*:

$\llbracket \text{val-of } e = \llbracket a \rrbracket; \text{ss-exps } (\text{map Val } vs @ \text{throw } e \# es') = \llbracket e' \rrbracket \rrbracket$   
 $\Longrightarrow e' = \text{throw } e$

*<proof>*

**abbreviation** *curr-init* :: 'm prog  $\Rightarrow$  'a exp  $\Rightarrow$  cname option **where**

*curr-init*  $P e \equiv \text{init-class } P (\text{ss-exp } e)$

**abbreviation** *curr-inits* :: 'm prog  $\Rightarrow$  'a exp list  $\Rightarrow$  cname option **where**

*curr-inits*  $P es \equiv \text{case ss-exps } es \text{ of Some } e \Rightarrow \text{init-class } P e \mid - \Rightarrow \text{None}$

**lemma** *icheck-curr-init'*:  $\bigwedge e'. \text{ss-exp } e = e' \Longrightarrow \text{icheck } P C e' \Longrightarrow \text{curr-init } P e = \llbracket C \rrbracket$

**and** *icheck-curr-inits'*:  $\bigwedge e. \text{ss-exps } es = \llbracket e \rrbracket \Longrightarrow \text{icheck } P C e \Longrightarrow \text{curr-inits } P es = \llbracket C \rrbracket$

*<proof>*

**lemma** *icheck-curr-init*:  $\text{icheck } P C e' \Longrightarrow \text{ss-exp } e = e' \Longrightarrow \text{curr-init } P e = \llbracket C \rrbracket$

*<proof>*

**lemma** *icheck-curr-inits*:  $\text{icheck } P C e \Longrightarrow \text{ss-exps } es = \llbracket e \rrbracket \Longrightarrow \text{curr-inits } P es = \llbracket C \rrbracket$

*<proof>*

**definition** *initPD* :: sheap  $\Rightarrow$  cname  $\Rightarrow$  bool **where**

*initPD*  $sh C \equiv \exists sfs i. sh C = \text{Some } (sfs, i) \wedge (i = \text{Done} \vee i = \text{Processing})$

— checks that *INIT* and *RI* conform and are only in the main computation

**fun** *iconf* :: sheap  $\Rightarrow$  'a exp  $\Rightarrow$  bool **and** *iconfs* :: sheap  $\Rightarrow$  'a exp list  $\Rightarrow$  bool **where**

*iconf*  $sh (\text{new } C) = \text{True}$

$| \text{iconf sh } (\text{Cast } C \ e) = \text{iconf sh } e$   
 $| \text{iconf sh } (\text{Val } v) = \text{True}$   
 $| \text{iconf sh } (e_1 \ll \text{bop} \gg e_2) = (\text{case val-of } e_1 \text{ of Some } v \Rightarrow \text{iconf sh } e_2 \mid - \Rightarrow \text{iconf sh } e_1 \wedge \neg \text{sub-RI } e_2)$   
 $| \text{iconf sh } (\text{Var } V) = \text{True}$   
 $| \text{iconf sh } (\text{LAss } V \ e) = \text{iconf sh } e$   
 $| \text{iconf sh } (e \cdot F\{D\}) = \text{iconf sh } e$   
 $| \text{iconf sh } (C \cdot_s F\{D\}) = \text{True}$   
 $| \text{iconf sh } (e_1 \cdot F\{D\} := e_2) = (\text{case val-of } e_1 \text{ of Some } v \Rightarrow \text{iconf sh } e_2 \mid - \Rightarrow \text{iconf sh } e_1 \wedge \neg \text{sub-RI } e_2)$   
 $| \text{iconf sh } (C \cdot_s F\{D\} := e_2) = \text{iconf sh } e_2$   
 $| \text{iconf sh } (e \cdot M(es)) = (\text{case val-of } e \text{ of Some } v \Rightarrow \text{iconfs sh } es \mid - \Rightarrow \text{iconf sh } e \wedge \neg \text{sub-RIs } es)$   
 $| \text{iconf sh } (C \cdot_s M(es)) = \text{iconfs sh } es$   
 $| \text{iconf sh } (\{V:T; e\}) = \text{iconf sh } e$   
 $| \text{iconf sh } (e_1;; e_2) = (\text{case val-of } e_1 \text{ of Some } v \Rightarrow \text{iconf sh } e_2$   
 $\quad \mid \text{None} \Rightarrow (\text{case lass-val-of } e_1 \text{ of Some } p \Rightarrow \text{iconf sh } e_2$   
 $\quad \quad \quad \mid \text{None} \Rightarrow \text{iconf sh } e_1 \wedge \neg \text{sub-RI } e_2))$   
 $| \text{iconf sh } (\text{if } (b) \ e_1 \ \text{else } e_2) = (\text{iconf sh } b \wedge \neg \text{sub-RI } e_1 \wedge \neg \text{sub-RI } e_2)$   
 $| \text{iconf sh } (\text{while } (b) \ e) = (\neg \text{sub-RI } b \wedge \neg \text{sub-RI } e)$   
 $| \text{iconf sh } (\text{throw } e) = \text{iconf sh } e$   
 $| \text{iconf sh } (\text{try } e_1 \ \text{catch}(C \ V) \ e_2) = (\text{iconf sh } e_1 \wedge \neg \text{sub-RI } e_2)$   
 $| \text{iconf sh } (\text{INIT } C \ (Cs, b) \leftarrow e) = ((\text{case } Cs \ \text{of Nil} \Rightarrow \text{initPD sh } C \mid C' \# Cs' \Rightarrow \text{last } Cs = C) \wedge \neg \text{sub-RI } e)$   
 $| \text{iconf sh } (\text{RI } (C, e); Cs \leftarrow e') = (\text{iconf sh } e \wedge \neg \text{sub-RI } e')$   
 $| \text{iconfs sh } (\square) = \text{True}$   
 $| \text{iconfs sh } (e \# es) = (\text{case val-of } e \ \text{of Some } v \Rightarrow \text{iconfs sh } es \mid - \Rightarrow \text{iconf sh } e \wedge \neg \text{sub-RIs } es)$

**lemma** *iconfs-map-throw*:  $\text{iconfs sh } (\text{map Val vs } @ \ \text{throw } e \ \# \ es') \Longrightarrow \text{iconf sh } e$   
 $\langle \text{proof} \rangle$

**lemma** *nsub-RI-iconf-aux*:

$(\neg \text{sub-RI } (e::'a \ \text{exp}) \longrightarrow (\forall e'. e' \in \text{subexp } e \longrightarrow \neg \text{sub-RI } e' \longrightarrow \text{iconf sh } e') \longrightarrow \text{iconf sh } e)$   
 $\wedge (\neg \text{sub-RIs } (es::'a \ \text{exp list}) \longrightarrow (\forall e'. e' \in \text{subexps } es \longrightarrow \neg \text{sub-RI } e' \longrightarrow \text{iconf sh } e') \longrightarrow \text{iconfs sh } es)$   
 $\langle \text{proof} \rangle$

**lemma** *nsub-RI-iconf-aux'*:

$(\bigwedge e'. \text{subexp-of } e' \ e \Longrightarrow \neg \text{sub-RI } e' \longrightarrow \text{iconf sh } e') \Longrightarrow (\neg \text{sub-RI } e \Longrightarrow \text{iconf sh } e)$   
 $\langle \text{proof} \rangle$

**lemma** *nsub-RI-iconf*:  $\neg \text{sub-RI } e \Longrightarrow \text{iconf sh } e$

**and** *nsub-RIs-iconfs*:  $\neg \text{sub-RIs } es \Longrightarrow \text{iconfs sh } es$

$\langle \text{proof} \rangle$

**lemma** *lass-val-of-iconf*:  $\text{lass-val-of } e = \lfloor a \rfloor \Longrightarrow \text{iconf sh } e$

$\langle \text{proof} \rangle$

**lemma** *icheck-iconf*:

**assumes** *icheck*  $P \ C \ e$  **shows**  $\text{iconf sh } e$

$\langle \text{proof} \rangle$

## 1.19.2 Indicator boolean conformance

**definition** *bconf* ::  $'m \ \text{prog} \Rightarrow \text{sheap} \Rightarrow 'a \ \text{exp} \Rightarrow \text{bool} \Rightarrow \text{bool}$  ( $\langle -, - \vdash_b '(-, -) \surd \rangle$  [51,51,0,0] 50)

**where**

$P, sh \vdash_b (e, b) \surd \equiv b \longrightarrow (\exists C. \text{icheck } P \ C \ (ss\text{-exp } e) \wedge \text{initPD sh } C)$

**definition**  $bconfs :: 'm \text{ prog} \Rightarrow \text{sheap} \Rightarrow 'a \text{ exp list} \Rightarrow \text{bool} \Rightarrow \text{bool} \langle \langle -, - \rangle \vdash_b '(-, -)' \checkmark \rangle [51, 51, 0, 0] 50)$

where

$$P, sh \vdash_b (es, b) \checkmark \equiv b \longrightarrow (\exists C. (\text{icheck } P \ C \ (\text{the}(ss\text{-exps } es)) \\ \wedge (\text{curr-inits } P \ es = \text{Some } C) \wedge \text{initPD } sh \ C))$$

— bconf helper lemmas

**lemma**  $bconf\text{-nonVal}[simp]:$

$$P, sh \vdash_b (e, \text{True}) \checkmark \Longrightarrow \text{val-of } e = \text{None} \\ \langle \text{proof} \rangle$$

**lemma**  $bconfs\text{-nonVals}[simp]:$

$$P, sh \vdash_b (es, \text{True}) \checkmark \Longrightarrow \text{map-vals-of } es = \text{None} \\ \langle \text{proof} \rangle$$

**lemma**  $bconf\text{-Cast}[iff]:$

$$P, sh \vdash_b (\text{Cast } C \ e, b) \checkmark \longleftrightarrow P, sh \vdash_b (e, b) \checkmark \\ \langle \text{proof} \rangle$$

**lemma**  $bconf\text{-BinOp}[iff]:$

$$P, sh \vdash_b (e1 \ \langle \text{bop} \rangle \ e2, b) \checkmark \\ \longleftrightarrow (\text{case val-of } e1 \ \text{of } \text{Some } v \Rightarrow P, sh \vdash_b (e2, b) \checkmark \mid - \Rightarrow P, sh \vdash_b (e1, b) \checkmark) \\ \langle \text{proof} \rangle$$

**lemma**  $bconf\text{-LAss}[iff]:$

$$P, sh \vdash_b (\text{LAss } V \ e, b) \checkmark \longleftrightarrow P, sh \vdash_b (e, b) \checkmark \\ \langle \text{proof} \rangle$$

**lemma**  $bconf\text{-FAcc}[iff]:$

$$P, sh \vdash_b (e \cdot F \{D\}, b) \checkmark \longleftrightarrow P, sh \vdash_b (e, b) \checkmark \\ \langle \text{proof} \rangle$$

**lemma**  $bconf\text{-FAss}[iff]:$

$$P, sh \vdash_b (\text{FAss } e1 \ F \ D \ e2, b) \checkmark \\ \longleftrightarrow (\text{case val-of } e1 \ \text{of } \text{Some } v \Rightarrow P, sh \vdash_b (e2, b) \checkmark \mid - \Rightarrow P, sh \vdash_b (e1, b) \checkmark) \\ \langle \text{proof} \rangle$$

**lemma**  $bconf\text{-SFAss}[iff]:$

$$\text{val-of } e2 = \text{None} \Longrightarrow P, sh \vdash_b (\text{SFAss } C \ F \ D \ e2, b) \checkmark \longleftrightarrow P, sh \vdash_b (e2, b) \checkmark \\ \langle \text{proof} \rangle$$

**lemma**  $bconfs\text{-Vals}[iff]:$

$$P, sh \vdash_b (\text{map Val } vs, b) \checkmark \longleftrightarrow \neg b \\ \langle \text{proof} \rangle$$

**lemma**  $bconf\text{-Call}[iff]:$

$$P, sh \vdash_b (e \cdot M(es), b) \checkmark \\ \longleftrightarrow (\text{case val-of } e \ \text{of } \text{Some } v \Rightarrow P, sh \vdash_b (es, b) \checkmark \mid - \Rightarrow P, sh \vdash_b (e, b) \checkmark) \\ \langle \text{proof} \rangle$$

**lemma**  $bconf\text{-SCall}[iff]:$

**assumes**  $mnv$ :  $\text{map-vals-of } es = \text{None}$

**shows**  $P, sh \vdash_b (C \cdot_s M(es), b) \checkmark \longleftrightarrow P, sh \vdash_b (es, b) \checkmark$   
 ⟨proof⟩

**lemma** *bconf-Cons*[iff]:  
 $P, sh \vdash_b (e \# es, b) \checkmark$   
 $\longleftrightarrow (case\ val\ of\ e\ of\ Some\ v \Rightarrow P, sh \vdash_b (es, b) \checkmark \mid - \Rightarrow P, sh \vdash_b (e, b) \checkmark)$   
 ⟨proof⟩

**lemma** *bconf-InitBlock*[iff]:  
 $P, sh \vdash_b (\{V:T; V:=Val\ v;; e_2\}, b) \checkmark \longleftrightarrow P, sh \vdash_b (e_2, b) \checkmark$   
 ⟨proof⟩

**lemma** *bconf-Block*[iff]:  
 $P, sh \vdash_b (\{V:T; e\}, b) \checkmark \longleftrightarrow P, sh \vdash_b (e, b) \checkmark$   
 ⟨proof⟩

**lemma** *bconf-Seq*[iff]:  
 $P, sh \vdash_b (e_1;; e_2, b) \checkmark$   
 $\longleftrightarrow (case\ val\ of\ e_1\ of\ Some\ v \Rightarrow P, sh \vdash_b (e_2, b) \checkmark$   
 $\mid - \Rightarrow (case\ lass\ val\ of\ e_1\ of\ Some\ p \Rightarrow P, sh \vdash_b (e_2, b) \checkmark$   
 $\mid None \Rightarrow P, sh \vdash_b (e_1, b) \checkmark))$   
 ⟨proof⟩

**lemma** *bconf-Cond*[iff]:  
 $P, sh \vdash_b (if\ (b)\ e_1\ else\ e_2, b') \checkmark \longleftrightarrow P, sh \vdash_b (b, b') \checkmark$   
 ⟨proof⟩

**lemma** *bconf-While*[iff]:  
 $P, sh \vdash_b (while\ (b)\ e, b') \checkmark \longleftrightarrow \neg b'$   
 ⟨proof⟩

**lemma** *bconf-Throw*[iff]:  
 $P, sh \vdash_b (throw\ e, b) \checkmark \longleftrightarrow P, sh \vdash_b (e, b) \checkmark$   
 ⟨proof⟩

**lemma** *bconf-Try*[iff]:  
 $P, sh \vdash_b (try\ e_1\ catch\ (C\ V)\ e_2, b) \checkmark \longleftrightarrow P, sh \vdash_b (e_1, b) \checkmark$   
 ⟨proof⟩

**lemma** *bconf-INIT*[iff]:  
 $P, sh \vdash_b (INIT\ C\ (Cs, b') \leftarrow e, b) \checkmark \longleftrightarrow \neg b$   
 ⟨proof⟩

**lemma** *bconf-RI*[iff]:  
 $P, sh \vdash_b (RI\ (C, e); Cs \leftarrow e', b) \checkmark \longleftrightarrow P, sh \vdash_b (e, b) \checkmark$   
 ⟨proof⟩

**lemma** *bconfs-map-throw*[iff]:  
 $P, sh \vdash_b (map\ Val\ vs\ @\ throw\ e\ \# es', b) \checkmark \longleftrightarrow P, sh \vdash_b (e, b) \checkmark$   
 ⟨proof⟩

**end**



## 1.20 Progress of Small Step Semantics

**theory** *Progress*

**imports** *WellTypeRT DefAss ../Common/Conform EConform*

**begin**

**lemma** *final-addrE*:

$$\begin{aligned} & \llbracket P, E, h, sh \vdash e : \text{Class } C; \text{ final } e; \\ & \quad \bigwedge a. e = \text{addr } a \implies R; \\ & \quad \bigwedge a. e = \text{Throw } a \implies R \rrbracket \implies R(\text{proof}) \end{aligned}$$

**lemma** *finalRefE*:

$$\begin{aligned} & \llbracket P, E, h, sh \vdash e : T; \text{ is-refT } T; \text{ final } e; \\ & \quad e = \text{null} \implies R; \\ & \quad \bigwedge a C. \llbracket e = \text{addr } a; T = \text{Class } C \rrbracket \implies R; \\ & \quad \bigwedge a. e = \text{Throw } a \implies R \rrbracket \implies R(\text{proof}) \end{aligned}$$

Derivation of new induction scheme for well typing:

**inductive**

$$\begin{aligned} & \text{WTrt}' :: [J\text{-prog}, \text{heap}, \text{sheap}, \text{env}, \text{expr}, \text{ty}] \Rightarrow \text{bool} \\ & \text{and WTrts}' :: [J\text{-prog}, \text{heap}, \text{sheap}, \text{env}, \text{expr list}, \text{ty list}] \Rightarrow \text{bool} \\ & \text{and WTrt2}' :: [J\text{-prog}, \text{env}, \text{heap}, \text{sheap}, \text{expr}, \text{ty}] \Rightarrow \text{bool} \\ & \quad (\langle -, -, -, - \vdash - : '' \rangle \rightarrow [51, 51, 51, 51] 50) \\ & \text{and WTrts2}' :: [J\text{-prog}, \text{env}, \text{heap}, \text{sheap}, \text{expr list}, \text{ty list}] \Rightarrow \text{bool} \\ & \quad (\langle -, -, -, - \vdash - : '' \rangle \rightarrow [51, 51, 51, 51] 50) \\ & \text{for } P :: J\text{-prog} \text{ and } h :: \text{heap} \text{ and } sh :: \text{sheap} \end{aligned}$$

**where**

$$\begin{aligned} & P, E, h, sh \vdash e : ' T \equiv \text{WTrt}' P h sh E e T \\ & | P, E, h, sh \vdash es [:' ] Ts \equiv \text{WTrts}' P h sh E es Ts \\ \\ & | \text{is-class } P C \implies P, E, h, sh \vdash \text{new } C : ' \text{Class } C \\ & | \llbracket P, E, h, sh \vdash e : ' T; \text{ is-refT } T; \text{ is-class } P C \rrbracket \\ & \quad \implies P, E, h, sh \vdash \text{Cast } C e : ' \text{Class } C \\ & | \text{typeof}_h v = \text{Some } T \implies P, E, h, sh \vdash \text{Val } v : ' T \\ & | E v = \text{Some } T \implies P, E, h, sh \vdash \text{Var } v : ' T \\ & | \llbracket P, E, h, sh \vdash e_1 : ' T_1; P, E, h, sh \vdash e_2 : ' T_2 \rrbracket \\ & \quad \implies P, E, h, sh \vdash e_1 \llbracket \text{Eq} \rrbracket e_2 : ' \text{Boolean} \\ & | \llbracket P, E, h, sh \vdash e_1 : ' \text{Integer}; P, E, h, sh \vdash e_2 : ' \text{Integer} \rrbracket \\ & \quad \implies P, E, h, sh \vdash e_1 \llbracket \text{Add} \rrbracket e_2 : ' \text{Integer} \\ & | \llbracket P, E, h, sh \vdash \text{Var } V : ' T; P, E, h, sh \vdash e : ' T'; P \vdash T' \leq T \rrbracket \\ & \quad \implies P, E, h, sh \vdash V := e : ' \text{Void} \\ & | \llbracket P, E, h, sh \vdash e : ' \text{Class } C; P \vdash C \text{ has } F, \text{NonStatic}: T \text{ in } D \rrbracket \implies P, E, h, sh \vdash e \cdot F\{D\} : ' T \\ & | P, E, h, sh \vdash e : ' NT \implies P, E, h, sh \vdash e \cdot F\{D\} : ' T \\ & | \llbracket P \vdash C \text{ has } F, \text{Static}: T \text{ in } D \rrbracket \implies P, E, h, sh \vdash C \cdot_s F\{D\} : ' T \\ & | \llbracket P, E, h, sh \vdash e_1 : ' \text{Class } C; P \vdash C \text{ has } F, \text{NonStatic}: T \text{ in } D; \\ & \quad P, E, h, sh \vdash e_2 : ' T_2; P \vdash T_2 \leq T \rrbracket \\ & \quad \implies P, E, h, sh \vdash e_1 \cdot F\{D\} := e_2 : ' \text{Void} \\ & | \llbracket P, E, h, sh \vdash e_1 : ' NT; P, E, h, sh \vdash e_2 : ' T_2 \rrbracket \implies P, E, h, sh \vdash e_1 \cdot F\{D\} := e_2 : ' \text{Void} \\ & | \llbracket P \vdash C \text{ has } F, \text{Static}: T \text{ in } D; \\ & \quad P, E, h, sh \vdash e_2 : ' T_2; P \vdash T_2 \leq T \rrbracket \\ & \quad \implies P, E, h, sh \vdash C \cdot_s F\{D\} := e_2 : ' \text{Void} \\ & | \llbracket P, E, h, sh \vdash e : ' \text{Class } C; P \vdash C \text{ sees } M, \text{NonStatic}: Ts \rightarrow T = (\text{pns}, \text{body}) \text{ in } D; \\ & \quad P, E, h, sh \vdash es [:' ] Ts'; P \vdash Ts' \llbracket \leq \rrbracket Ts \rrbracket \\ & \quad \implies P, E, h, sh \vdash e \cdot M(es) : ' T \end{aligned}$$

$\llbracket P, E, h, sh \vdash e : ' NT; P, E, h, sh \vdash es [:\uparrow] Ts \rrbracket \Longrightarrow P, E, h, sh \vdash e \cdot M(es) : ' T$   
 $\llbracket P \vdash C \text{ sees } M, \text{Static: } Ts \rightarrow T = (pns, body) \text{ in } D;$   
 $P, E, h, sh \vdash es [:\uparrow] Ts'; P \vdash Ts' [\leq] Ts;$   
 $M = \text{clinit} \rightarrow sh D = \llbracket (sfs, Processing) \rrbracket \wedge es = \text{map Val } vs \rrbracket$   
 $\Longrightarrow P, E, h, sh \vdash C \cdot_s M(es) : ' T$   
 $P, E, h, sh \vdash [] [:\uparrow] []$   
 $\llbracket P, E, h, sh \vdash e : ' T; P, E, h, sh \vdash es [:\uparrow] Ts \rrbracket \Longrightarrow P, E, h, sh \vdash e \# es [:\uparrow] T \# Ts$   
 $\llbracket \text{typeof}_h v = \text{Some } T_1; P \vdash T_1 \leq T; P, E(V \mapsto T), h, sh \vdash e_2 : ' T_2 \rrbracket$   
 $\Longrightarrow P, E, h, sh \vdash \{V:T := \text{Val } v; e_2\} : ' T_2$   
 $\llbracket P, E(V \mapsto T), h, sh \vdash e : ' T'; \neg \text{assigned } V e \rrbracket \Longrightarrow P, E, h, sh \vdash \{V:T; e\} : ' T'$   
 $\llbracket P, E, h, sh \vdash e_1 : ' T_1; P, E, h, sh \vdash e_2 : ' T_2 \rrbracket \Longrightarrow P, E, h, sh \vdash e_1; e_2 : ' T_2$   
 $\llbracket P, E, h, sh \vdash e : ' \text{Boolean}; P, E, h, sh \vdash e_1 : ' T_1; P, E, h, sh \vdash e_2 : ' T_2;$   
 $P \vdash T_1 \leq T_2 \vee P \vdash T_2 \leq T_1;$   
 $P \vdash T_1 \leq T_2 \rightarrow T = T_2; P \vdash T_2 \leq T_1 \rightarrow T = T_1 \rrbracket$   
 $\Longrightarrow P, E, h, sh \vdash \text{if } (e) e_1 \text{ else } e_2 : ' T$   
 $\llbracket P, E, h, sh \vdash e : ' \text{Boolean}; P, E, h, sh \vdash c : ' T \rrbracket$   
 $\Longrightarrow P, E, h, sh \vdash \text{while}(e) c : ' \text{Void}$   
 $\llbracket P, E, h, sh \vdash e : ' T_r; \text{is-ref } T_r \rrbracket \Longrightarrow P, E, h, sh \vdash \text{throw } e : ' T$   
 $\llbracket P, E, h, sh \vdash e_1 : ' T_1; P, E(V \mapsto \text{Class } C), h, sh \vdash e_2 : ' T_2; P \vdash T_1 \leq T_2 \rrbracket$   
 $\Longrightarrow P, E, h, sh \vdash \text{try } e_1 \text{ catch}(C V) e_2 : ' T_2$   
 $\llbracket P, E, h, sh \vdash e : ' T; \forall C' \in \text{set } (C \# Cs). \text{is-class } P C'; \neg \text{sub-RI } e;$   
 $\forall C' \in \text{set } (tl \ Cs). \exists \text{sfs. } sh C' = \llbracket (sfs, Processing) \rrbracket;$   
 $b \rightarrow (\forall C' \in \text{set } Cs. \exists \text{sfs. } sh C' = \llbracket (sfs, Processing) \rrbracket);$   
 $\text{distinct } Cs; \text{supercls-lst } P Cs \rrbracket \Longrightarrow P, E, h, sh \vdash \text{INIT } C (Cs, b) \leftarrow e : ' T$   
 $\llbracket P, E, h, sh \vdash e : ' T; P, E, h, sh \vdash e' : ' T'; \forall C' \in \text{set } (C \# Cs). \text{is-class } P C'; \neg \text{sub-RI } e';$   
 $\forall C' \in \text{set } (C \# Cs). \text{not-init } C' e';$   
 $\forall C' \in \text{set } Cs. \exists \text{sfs. } sh C' = \llbracket (sfs, Processing) \rrbracket;$   
 $\exists \text{sfs. } sh C = \llbracket (sfs, Processing) \rrbracket \vee (sh C = \llbracket (sfs, Error) \rrbracket) \wedge e = \text{THROW NoClassDefFoundError};$   
 $\text{distinct } (C \# Cs); \text{supercls-lst } P (C \# Cs) \rrbracket$   
 $\Longrightarrow P, E, h, sh \vdash \text{RI}(C, e); Cs \leftarrow e' : ' T'$

**lemma** [iff]:  $P, E, h, sh \vdash e_1; e_2 : ' T_2 = (\exists T_1. P, E, h, sh \vdash e_1 : ' T_1 \wedge P, E, h, sh \vdash e_2 : ' T_2)$ ⟨proof⟩

**lemma** [iff]:  $P, E, h, sh \vdash \text{Val } v : ' T = (\text{typeof}_h v = \text{Some } T)$ ⟨proof⟩

**lemma** [iff]:  $P, E, h, sh \vdash \text{Var } v : ' T = (E v = \text{Some } T)$ ⟨proof⟩

**lemma** *wt-wt'*:  $P, E, h, sh \vdash e : T \Longrightarrow P, E, h, sh \vdash e : ' T$

**and** *wts-wts'*:  $P, E, h, sh \vdash es [:\uparrow] Ts \Longrightarrow P, E, h, sh \vdash es [:\uparrow] Ts$ ⟨proof⟩

**lemma** *wt'-wt*:  $P, E, h, sh \vdash e : ' T \Longrightarrow P, E, h, sh \vdash e : T$

**and** *wts'-wts*:  $P, E, h, sh \vdash es [:\uparrow] Ts \Longrightarrow P, E, h, sh \vdash es [:\uparrow] Ts$ ⟨proof⟩

**corollary** *wt'-iff-wt*:  $(P, E, h, sh \vdash e : ' T) = (P, E, h, sh \vdash e : T)$ ⟨proof⟩

**corollary** *wts'-iff-wts*:  $(P, E, h, sh \vdash es [:\uparrow] Ts) = (P, E, h, sh \vdash es [:\uparrow] Ts)$ ⟨proof⟩

**theorem** *assumes wf: wuf-J-prog P and hconf: P ⊢ h √ and shconf: P, h ⊢<sub>s</sub> sh √*

**shows** *progress: P, E, h, sh ⊢ e : T ⟹*

$(\bigwedge l. \llbracket D e [dom l]; P, sh \vdash_b (e, b) \checkmark; \neg \text{final } e \rrbracket \Longrightarrow \exists e' s' b'. P \vdash \langle e, (h, l, sh), b \rangle \rightarrow \langle e', s', b' \rangle)$

**and**  $P, E, h, sh \vdash es [:\uparrow] Ts \Longrightarrow$

$(\bigwedge l. \llbracket Ds es [dom l]; P, sh \vdash_b (es, b) \checkmark; \neg \text{finals } es \rrbracket \Longrightarrow \exists es' s' b'. P \vdash \langle es, (h, l, sh), b \rangle \rightarrow \langle es', s', b' \rangle)$ ⟨proof⟩

**end**

## 1.21 Well-formedness Constraints

**theory** *JWellForm*

**imports** *../Common/WellForm WWellForm WellType DefAss*

**begin**

**definition** *wf-J-mdecl* :: *J-prog*  $\Rightarrow$  *cname*  $\Rightarrow$  *J-mb mdecl*  $\Rightarrow$  *bool*

**where**

*wf-J-mdecl* *P C*  $\equiv$   $\lambda(M, b, Ts, T, (pns, body)).$

*length* *Ts* = *length* *pns*  $\wedge$

*distinct* *pns*  $\wedge$

$\neg$ *sub-RI* *body*  $\wedge$

(*case* *b* *of*

*NonStatic*  $\Rightarrow$  *this*  $\notin$  *set* *pns*  $\wedge$

$(\exists T'. P, [this \mapsto \text{Class } C, pns \mapsto Ts] \vdash body :: T' \wedge P \vdash T' \leq T) \wedge$

$\mathcal{D} \text{ body } [\{this\} \cup \text{set } pns]$

| *Static*  $\Rightarrow (\exists T'. P, [pns \mapsto Ts] \vdash body :: T' \wedge P \vdash T' \leq T) \wedge$

$\mathcal{D} \text{ body } [set \text{ } pns]$ )

**lemma** *wf-J-mdecl-NonStatic[simp]*:

*wf-J-mdecl* *P C* (*M, NonStatic, Ts, T, pns, body*)  $\equiv$

(*length* *Ts* = *length* *pns*  $\wedge$

*distinct* *pns*  $\wedge$

$\neg$ *sub-RI* *body*  $\wedge$

*this*  $\notin$  *set* *pns*  $\wedge$

$(\exists T'. P, [this \mapsto \text{Class } C, pns \mapsto Ts] \vdash body :: T' \wedge P \vdash T' \leq T) \wedge$

$\mathcal{D} \text{ body } [\{this\} \cup \text{set } pns]$ )*<proof>*

**lemma** *wf-J-mdecl-Static[simp]*:

*wf-J-mdecl* *P C* (*M, Static, Ts, T, pns, body*)  $\equiv$

(*length* *Ts* = *length* *pns*  $\wedge$

*distinct* *pns*  $\wedge$

$\neg$ *sub-RI* *body*  $\wedge$

$(\exists T'. P, [pns \mapsto Ts] \vdash body :: T' \wedge P \vdash T' \leq T) \wedge$

$\mathcal{D} \text{ body } [set \text{ } pns]$ )*<proof>*

**abbreviation**

*wf-J-prog* :: *J-prog*  $\Rightarrow$  *bool* **where**

*wf-J-prog* == *wf-prog wf-J-mdecl*

**lemma** *wf-J-prog-wf-J-mdecl*:

$\llbracket wf\text{-}J\text{-prog } P; (C, D, fds, mths) \in \text{set } P; jmdcl \in \text{set } mths \rrbracket$

$\Longrightarrow wf\text{-}J\text{-mdecl } P C jmdcl$ *<proof>*

**lemma** *wf-mdecl-wwf-mdecl*: *wf-J-mdecl* *P C Md*  $\Longrightarrow wwf\text{-}J\text{-mdecl } P C Md$ *<proof>*

**lemma** *wf-prog-wwf-prog*: *wf-J-prog* *P*  $\Longrightarrow wwf\text{-}J\text{-prog } P$ *<proof>*

**end**

## 1.22 Type Safety Proof

**theory** *TypeSafe*

**imports** *Progress BigStep SmallStep JWellForm*

**begin**

**lemma red-shext-incr:**  $P \vdash \langle e, (h, l, sh), b \rangle \rightarrow \langle e', (h', l', sh'), b' \rangle$   
 $\implies (\bigwedge E T. P, E, h, sh \vdash e : T \implies sh \leq_s sh')$

**and reds-shext-incr:**  $P \vdash \langle es, (h, l, sh), b \rangle [\rightarrow] \langle es', (h', l', sh'), b' \rangle$   
 $\implies (\bigwedge E Ts. P, E, h, sh \vdash es [:] Ts \implies sh \leq_s sh') \langle proof \rangle$

**lemma wf-types-clinit:**

**assumes**  $wf: wf\text{-prog } wf\text{-md } P$  **and**  $ex: class P C = Some a$  **and**  $proc: sh C = [(sfs, Processing)]$   
**shows**  $P, E, h, sh \vdash C \cdot_s clinit([]) : Void$   
 $\langle proof \rangle$

### 1.22.1 Basic preservation lemmas

First some easy preservation lemmas.

**theorem red-preserves-hconf:**

$P \vdash \langle e, (h, l, sh), b \rangle \rightarrow \langle e', (h', l', sh'), b' \rangle \implies (\bigwedge T E. \llbracket P, E, h, sh \vdash e : T; P \vdash h \checkmark \rrbracket \implies P \vdash h' \checkmark)$

**and reds-preserves-hconf:**

$P \vdash \langle es, (h, l, sh), b \rangle [\rightarrow] \langle es', (h', l', sh'), b' \rangle \implies (\bigwedge Ts E. \llbracket P, E, h, sh \vdash es [:] Ts; P \vdash h \checkmark \rrbracket \implies P \vdash h' \checkmark) \langle proof \rangle$

**theorem red-preserves-lconf:**

$P \vdash \langle e, (h, l, sh), b \rangle \rightarrow \langle e', (h', l', sh'), b' \rangle \implies$   
 $(\bigwedge T E. \llbracket P, E, h, sh \vdash e : T; P, h \vdash l (: \leq) E \rrbracket \implies P, h' \vdash l' (: \leq) E)$

**and reds-preserves-lconf:**

$P \vdash \langle es, (h, l, sh), b \rangle [\rightarrow] \langle es', (h', l', sh'), b' \rangle \implies$   
 $(\bigwedge Ts E. \llbracket P, E, h, sh \vdash es [:] Ts; P, h \vdash l (: \leq) E \rrbracket \implies P, h' \vdash l' (: \leq) E) \langle proof \rangle$

**theorem red-preserves-shconf:**

$P \vdash \langle e, (h, l, sh), b \rangle \rightarrow \langle e', (h', l', sh'), b' \rangle \implies (\bigwedge T E. \llbracket P, E, h, sh \vdash e : T; P, h \vdash_s sh \checkmark \rrbracket \implies P, h' \vdash_s sh' \checkmark)$

**and reds-preserves-shconf:**

$P \vdash \langle es, (h, l, sh), b \rangle [\rightarrow] \langle es', (h', l', sh'), b' \rangle \implies (\bigwedge Ts E. \llbracket P, E, h, sh \vdash es [:] Ts; P, h \vdash_s sh \checkmark \rrbracket \implies P, h' \vdash_s sh' \checkmark) \langle proof \rangle$

**theorem assumes**  $wf: wwf\text{-J-prog } P$

**shows red-preserves-icnf:**

$P \vdash \langle e, (h, l, sh), b \rangle \rightarrow \langle e', (h', l', sh'), b' \rangle \implies icnf sh e \implies icnf sh' e'$

**and reds-preserves-icnf:**

$P \vdash \langle es, (h, l, sh), b \rangle [\rightarrow] \langle es', (h', l', sh'), b' \rangle \implies iconfs sh es \implies iconfs sh' es' \langle proof \rangle$

**lemma Seq-bconf-preserve-aux:**

**assumes**  $P \vdash \langle e, (h, l, sh), b \rangle \rightarrow \langle e', (h', l', sh'), b' \rangle$  **and**  $P, sh \vdash_b (e;; e_2, b) \checkmark$

**and**  $P, sh \vdash_b (e::expr, b) \checkmark \longrightarrow P, sh' \vdash_b (e'::expr, b') \checkmark$

**shows**  $P, sh' \vdash_b (e';; e_2, b') \checkmark$

$\langle proof \rangle$

**theorem red-preserves-bconf:**

$P \vdash \langle e, (h, l, sh), b \rangle \rightarrow \langle e', (h', l', sh'), b' \rangle \implies icnf sh e \implies P, sh \vdash_b (e, b) \checkmark \implies P, sh' \vdash_b (e', b') \checkmark$

**and reds-preserves-bconf:**

$P \vdash \langle es, (h, l, sh), b \rangle [\rightarrow] \langle es', (h', l', sh'), b' \rangle \implies iconfs sh es \implies P, sh \vdash_b (es, b) \checkmark \implies P, sh' \vdash_b (es', b') \checkmark \langle proof \rangle$

Preservation of definite assignment more complex and requires a few lemmas first.

**lemma [iff]:**  $\bigwedge A. \llbracket length Vs = length Ts; length vs = length Ts \rrbracket \implies$

$\mathcal{D} (blocks (Vs, Ts, vs, e)) A = \mathcal{D} e (A \sqcup [set Vs]) \langle proof \rangle$

**lemma red-lA-incr:**  $P \vdash \langle e, (h, l, sh), b \rangle \rightarrow \langle e', (h', l', sh'), b' \rangle$

$\implies \llbracket \text{dom } l \rrbracket \sqcup \mathcal{A} e \sqsubseteq \llbracket \text{dom } l' \rrbracket \sqcup \mathcal{A} e'$   
**and** *reds-lA-incr*:  $P \vdash \langle es, (h, l, sh), b \rangle \mapsto \langle es', (h', l', sh'), b' \rangle$   
 $\implies \llbracket \text{dom } l \rrbracket \sqcup \mathcal{A} s es \sqsubseteq \llbracket \text{dom } l' \rrbracket \sqcup \mathcal{A} s es' \langle \text{proof} \rangle$

Now preservation of definite assignment.

**lemma** *assumes* *wf*: *wf-J-prog* *P*

**shows** *red-preserves-defass*:

$P \vdash \langle e, (h, l, sh), b \rangle \mapsto \langle e', (h', l', sh'), b' \rangle \implies \mathcal{D} e \llbracket \text{dom } l \rrbracket \implies \mathcal{D} e' \llbracket \text{dom } l' \rrbracket$   
**and**  $P \vdash \langle es, (h, l, sh), b \rangle \mapsto \langle es', (h', l', sh'), b' \rangle \implies \mathcal{D} s es \llbracket \text{dom } l \rrbracket \implies \mathcal{D} s es' \llbracket \text{dom } l' \rrbracket \langle \text{proof} \rangle$

Combining conformance of heap, static heap, and local variables:

**definition** *sconf* :: *J-prog*  $\Rightarrow$  *env*  $\Rightarrow$  *state*  $\Rightarrow$  *bool* ( $\langle -, - \vdash - \sqrt{\phantom{x}} \rangle$  [51,51,51]50)

**where**

$P, E \vdash s \sqrt{\phantom{x}} \equiv \text{let } (h, l, sh) = s \text{ in } P \vdash h \sqrt{\phantom{x}} \wedge P, h \vdash l (\leq) E \wedge P, h \vdash_s sh \sqrt{\phantom{x}}$

**lemma** *red-preserves-sconf*:

$\llbracket P \vdash \langle e, s, b \rangle \mapsto \langle e', s', b' \rangle; P, E, hp s, shp s \vdash e : T; P, E \vdash s \sqrt{\phantom{x}} \rrbracket \implies P, E \vdash s' \sqrt{\phantom{x}} \langle \text{proof} \rangle$

**lemma** *reds-preserves-sconf*:

$\llbracket P \vdash \langle es, s, b \rangle \mapsto \langle es', s', b' \rangle; P, E, hp s, shp s \vdash es [:] Ts; P, E \vdash s \sqrt{\phantom{x}} \rrbracket \implies P, E \vdash s' \sqrt{\phantom{x}} \langle \text{proof} \rangle$

## 1.22.2 Subject reduction

**lemma** *wt-blocks*:

$\bigwedge E. \llbracket \text{length } Vs = \text{length } Ts; \text{length } vs = \text{length } Ts \rrbracket \implies$   
 $(P, E, h, sh \vdash \text{blocks}(Vs, Ts, vs, e) : T) =$   
 $(P, E(Vs \mapsto Ts), h, sh \vdash e : T \wedge (\exists Ts'. \text{map } (\text{typeof}_h) vs = \text{map } \text{Some } Ts' \wedge P \vdash Ts' \leq Ts)) \langle \text{proof} \rangle$

**theorem** *assumes* *wf*: *wf-J-prog* *P*

**shows** *subject-reduction2*:  $P \vdash \langle e, (h, l, sh), b \rangle \mapsto \langle e', (h', l', sh'), b' \rangle \implies$

$(\bigwedge E T. \llbracket P, E \vdash (h, l, sh) \sqrt{\phantom{x}}; \text{iconf } sh e; P, E, h, sh \vdash e : T \rrbracket$   
 $\implies \exists T'. P, E, h', sh' \vdash e' : T' \wedge P \vdash T' \leq T)$

**and** *subjects-reduction2*:  $P \vdash \langle es, (h, l, sh), b \rangle \mapsto \langle es', (h', l', sh'), b' \rangle \implies$

$(\bigwedge E Ts. \llbracket P, E \vdash (h, l, sh) \sqrt{\phantom{x}}; \text{iconfs } sh es; P, E, h, sh \vdash es [:] Ts \rrbracket$   
 $\implies \exists Ts'. P, E, h', sh' \vdash es' [:] Ts' \wedge P \vdash Ts' \leq Ts \rangle \langle \text{proof} \rangle$

**corollary** *subject-reduction*:

$\llbracket \text{wf-J-prog } P; P \vdash \langle e, s, b \rangle \mapsto \langle e', s', b' \rangle; P, E \vdash s \sqrt{\phantom{x}}; \text{iconf } (shp s) e; P, E, hp s, shp s \vdash e : T \rrbracket$   
 $\implies \exists T'. P, E, hp s', shp s' \vdash e' : T' \wedge P \vdash T' \leq T \langle \text{proof} \rangle$

**corollary** *subjects-reduction*:

$\llbracket \text{wf-J-prog } P; P \vdash \langle es, s, b \rangle \mapsto \langle es', s', b' \rangle; P, E \vdash s \sqrt{\phantom{x}}; \text{iconfs } (shp s) es; P, E, hp s, shp s \vdash es [:] Ts \rrbracket$   
 $\implies \exists Ts'. P, E, hp s', shp s' \vdash es' [:] Ts' \wedge P \vdash Ts' \leq Ts \rangle \langle \text{proof} \rangle$

## 1.22.3 Lifting to $\rightarrow^*$

Now all these preservation lemmas are first lifted to the transitive closure ...

**lemma** *Red-preserves-sconf*:

**assumes** *wf*: *wf-J-prog* *P* **and** *Red*:  $P \vdash \langle e, s, b \rangle \rightarrow^* \langle e', s', b' \rangle$

**shows**  $\bigwedge T. \llbracket P, E, hp s, shp s \vdash e : T; \text{iconf } (shp s) e; P, E \vdash s \sqrt{\phantom{x}} \rrbracket \implies P, E \vdash s' \sqrt{\phantom{x}} \langle \text{proof} \rangle$

**lemma** *Red-preserves-iconf*:

**assumes** *wf*: *wf-J-prog* *P* **and** *Red*:  $P \vdash \langle e, s, b \rangle \rightarrow^* \langle e', s', b' \rangle$

**shows**  $\text{iconf } (shp s) e \implies \text{iconf } (shp s') e' \langle \text{proof} \rangle$

**lemma** *Reds-preserves-iconf*:

**assumes** *wf*: *wf-J-prog* *P* **and** *Red*:  $P \vdash \langle es, s, b \rangle \mapsto^* \langle es', s', b' \rangle$

**shows**  $\text{iconfs } (shp s) es \implies \text{iconfs } (shp s') es' \langle \text{proof} \rangle$

**lemma** *Red-preserves-bconf*:

**assumes**  $wf: w\text{-}J\text{-}prog\ P$  **and**  $Red: P \vdash \langle e, s, b \rangle \rightarrow^* \langle e', s', b' \rangle$   
**shows**  $iconf\ (shp\ s)\ e \implies P, (shp\ s) \vdash_b (e, b) \checkmark \implies P, (shp\ s') \vdash_b (e'::\text{expr}, b') \checkmark \langle \text{proof} \rangle$   
**lemma**  $Reds\text{-}preserves\text{-}bconf$ :  
**assumes**  $wf: w\text{-}J\text{-}prog\ P$  **and**  $Red: P \vdash \langle es, s, b \rangle [\rightarrow]^* \langle es', s', b' \rangle$   
**shows**  $iconfs\ (shp\ s)\ es \implies P, (shp\ s) \vdash_b (es, b) \checkmark \implies P, (shp\ s') \vdash_b (es'::\text{expr}\ list, b') \checkmark \langle \text{proof} \rangle$   
**lemma**  $Red\text{-}preserves\text{-}defass$ :  
**assumes**  $wf: w\text{-}J\text{-}prog\ P$  **and**  $reds: P \vdash \langle e, s, b \rangle \rightarrow^* \langle e', s', b' \rangle$   
**shows**  $\mathcal{D}\ e\ [dom(lcl\ s)] \implies \mathcal{D}\ e'\ [dom(lcl\ s')]$   
 $\langle \text{proof} \rangle$

**lemma**  $Red\text{-}preserves\text{-}type$ :

**assumes**  $wf: w\text{-}J\text{-}prog\ P$  **and**  $Red: P \vdash \langle e, s, b \rangle \rightarrow^* \langle e', s', b' \rangle$   
**shows**  $!!T. \llbracket P, E \vdash s \checkmark; iconf\ (shp\ s)\ e; P, E, hp\ s, shp\ s \vdash e : T \rrbracket$   
 $\implies \exists T'. P \vdash T' \leq T \wedge P, E, hp\ s', shp\ s' \vdash e' : T' \langle \text{proof} \rangle$

### 1.22.4 The final polish

The above preservation lemmas are now combined and packed nicely.

**definition**  $wf\text{-}config :: J\text{-}prog \Rightarrow env \Rightarrow state \Rightarrow expr \Rightarrow ty \Rightarrow bool$   $\langle \langle -, -, - \vdash - : - \checkmark \rangle [51, 0, 0, 0, 0] 50 \rangle$   
**where**

$P, E, s \vdash e : T \checkmark \equiv P, E \vdash s \checkmark \wedge iconf\ (shp\ s)\ e \wedge P, E, hp\ s, shp\ s \vdash e : T$

**theorem**  $Subject\text{-}reduction$ : **assumes**  $wf: w\text{-}J\text{-}prog\ P$

**shows**  $P \vdash \langle e, s, b \rangle \rightarrow \langle e', s', b' \rangle \implies P, E, s \vdash e : T \checkmark$   
 $\implies \exists T'. P, E, s' \vdash e' : T' \checkmark \wedge P \vdash T' \leq T \langle \text{proof} \rangle$

**theorem**  $Subject\text{-}reductions$ :

**assumes**  $wf: w\text{-}J\text{-}prog\ P$  **and**  $reds: P \vdash \langle e, s, b \rangle \rightarrow^* \langle e', s', b' \rangle$   
**shows**  $\bigwedge T. P, E, s \vdash e : T \checkmark \implies \exists T'. P, E, s' \vdash e' : T' \checkmark \wedge P \vdash T' \leq T \langle \text{proof} \rangle$

**corollary**  $Progress$ : **assumes**  $wf: w\text{-}J\text{-}prog\ P$

**shows**  $\llbracket P, E, s \vdash e : T \checkmark; \mathcal{D}\ e\ [dom(lcl\ s)]; P, shp\ s \vdash_b (e, b) \checkmark; \neg\ final\ e \rrbracket$   
 $\implies \exists e' s' b'. P \vdash \langle e, s, b \rangle \rightarrow \langle e', s', b' \rangle \langle \text{proof} \rangle$

**corollary**  $TypeSafety$ :

**fixes**  $s::state$  **and**  $e::expr$

**assumes**  $wf: w\text{-}J\text{-}prog\ P$  **and**  $sconf: P, E \vdash s \checkmark$  **and**  $wt: P, E \vdash e::T$

**and**  $\mathcal{D}: \mathcal{D}\ e\ [dom(lcl\ s)]$

**and**  $iconf: iconf\ (shp\ s)\ e$  **and**  $bconf: P, (shp\ s) \vdash_b (e, b) \checkmark$

**and**  $steps: P \vdash \langle e, s, b \rangle \rightarrow^* \langle e', s', b' \rangle$

**and**  $nstep: \neg(\exists e'' s'' b''. P \vdash \langle e', s', b' \rangle \rightarrow \langle e'', s'', b'' \rangle)$

**shows**  $(\exists v. e' = Val\ v \wedge P, hp\ s' \vdash v : \leq T) \vee$   
 $(\exists a. e' = Throw\ a \wedge a \in dom(hp\ s')) \langle \text{proof} \rangle$

**end**

## 1.23 Equivalence of Big Step and Small Step Semantics

**theory**  $Equivalence$  **imports**  $TypeSafe\ WWellForm$  **begin**

### 1.23.1 Small steps simulate big step

#### Init

The reduction of initialization expressions doesn't touch or use their on-hold expressions (the subexpression to the right of  $\leftarrow$ ) until the initialization operation completes. This function is used to prove this and related properties. It is then used for general reduction of initialization expressions separately from their on-hold expressions by using the on-hold expression *unit*, then putting the real on-hold expression back in at the end.

```
fun init-switch :: 'a exp  $\Rightarrow$  'a exp  $\Rightarrow$  'a exp where
init-switch (INIT C (Cs,b)  $\leftarrow$  ei) e = (INIT C (Cs,b)  $\leftarrow$  e) |
init-switch (RI(C,e');Cs  $\leftarrow$  ei) e = (RI(C,e');Cs  $\leftarrow$  e) |
init-switch e' e = e'
```

```
fun INIT-class :: 'a exp  $\Rightarrow$  cname option where
INIT-class (INIT C (Cs,b)  $\leftarrow$  e) = (if C = last (C#Cs) then Some C else None) |
INIT-class (RI(C,e0);Cs  $\leftarrow$  e) = Some (last (C#Cs)) |
INIT-class - = None
```

**lemma** *init-red-init*:

```
[[ init-exp-of e0 = [e]; P  $\vdash$  <e0,s0,b0>  $\rightarrow$  <e1,s1,b1> ]]
 $\implies$  (init-exp-of e1 = [e]  $\wedge$  (INIT-class e0 = [C]  $\longrightarrow$  INIT-class e1 = [C]))
 $\vee$  (e1 = e  $\wedge$  b1 = icheck P (the(INIT-class e0)) e)  $\vee$  ( $\exists$  a. e1 = throw a)
<proof>
```

**lemma** *init-exp-switch*[*simp*]:

```
init-exp-of e0 = [e]  $\implies$  init-exp-of (init-switch e0 e') = [e']
<proof>
```

**lemma** *init-red-sync*:

```
[[ P  $\vdash$  <e0,s0,b0>  $\rightarrow$  <e1,s1,b1>; init-exp-of e0 = [e]; e1  $\neq$  e ]]
 $\implies$  ( $\bigwedge$  e'. P  $\vdash$  <init-switch e0 e',s0,b0>  $\rightarrow$  <init-switch e1 e',s1,b1>))
<proof>
```

**lemma** *init-red-sync-end*:

```
[[ P  $\vdash$  <e0,s0,b0>  $\rightarrow$  <e1,s1,b1>; init-exp-of e0 = [e]; e1 = e; throw-of e = None ]]
 $\implies$  ( $\bigwedge$  e'.  $\neg$ sub-RI e'  $\implies$  P  $\vdash$  <init-switch e0 e',s0,b0>  $\rightarrow$  <e',s1,icheck P (the(INIT-class e0)) e'>))
<proof>
```

**lemma** *init-reds-sync-unit'*:

```
[[ P  $\vdash$  <e0,s0,b0>  $\rightarrow^*$  <Val v',s1,b1>; init-exp-of e0 = [unit]; INIT-class e0 = [C] ]]
 $\implies$  ( $\bigwedge$  e'.  $\neg$ sub-RI e'  $\implies$  P  $\vdash$  <init-switch e0 e',s0,b0>  $\rightarrow^*$  <e',s1,icheck P (the(INIT-class e0)) e'>))
<proof>
```

**lemma** *init-reds-sync-unit-throw'*:

```
[[ P  $\vdash$  <e0,s0,b0>  $\rightarrow^*$  <throw a,s1,b1>; init-exp-of e0 = [unit] ]]
 $\implies$  ( $\bigwedge$  e'. P  $\vdash$  <init-switch e0 e',s0,b0>  $\rightarrow^*$  <throw a,s1,b1>))
<proof>
```

**lemma** *init-reds-sync-unit*:

```
assumes P  $\vdash$  <e0,s0,b0>  $\rightarrow^*$  <Val v',s1,b1> and init-exp-of e0 = [unit] and INIT-class e0 = [C]
and  $\neg$ sub-RI e'
shows P  $\vdash$  <init-switch e0 e',s0,b0>  $\rightarrow^*$  <e',s1,icheck P (the(INIT-class e0)) e'>
<proof>
```

**lemma** *init-reds-sync-unit-throw*:

**assumes**  $P \vdash \langle e_0, s_0, b_0 \rangle \rightarrow^* \langle \text{throw } a, s_1, b_1 \rangle$  **and** *init-exp-of*  $e_0 = [\text{unit}]$   
**shows**  $P \vdash \langle \text{init-switch } e_0 \ e', s_0, b_0 \rangle \rightarrow^* \langle \text{throw } a, s_1, b_1 \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *InitSeqReds*:

**assumes**  $P \vdash \langle \text{INIT } C \ ([C], b) \leftarrow \text{unit}, s_0, b_0 \rangle \rightarrow^* \langle \text{Val } v', s_1, b_1 \rangle$   
**and**  $P \vdash \langle e, s_1, \text{icheck } P \ C \ e \rangle \rightarrow^* \langle e_2, s_2, b_2 \rangle$  **and**  $\neg \text{sub-RI } e$   
**shows**  $P \vdash \langle \text{INIT } C \ ([C], b) \leftarrow e, s_0, b_0 \rangle \rightarrow^* \langle e_2, s_2, b_2 \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *InitSeqThrowReds*:

**assumes**  $P \vdash \langle \text{INIT } C \ ([C], b) \leftarrow \text{unit}, s_0, b_0 \rangle \rightarrow^* \langle \text{throw } a, s_1, b_1 \rangle$   
**shows**  $P \vdash \langle \text{INIT } C \ ([C], b) \leftarrow e, s_0, b_0 \rangle \rightarrow^* \langle \text{throw } a, s_1, b_1 \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *InitNoneReds*:

$\llbracket \text{sh } C = \text{None};$   
 $P \vdash \langle \text{INIT } C' \ (C \# Cs, \text{False}) \leftarrow e, (h, l, \text{sh}(C \mapsto (\text{sblank } P \ C, \text{Prepared}))), b \rangle \rightarrow^* \langle e', s', b' \rangle \rrbracket$   
 $\implies P \vdash \langle \text{INIT } C' \ (C \# Cs, \text{False}) \leftarrow e, (h, l, \text{sh}), b \rangle \rightarrow^* \langle e', s', b' \rangle \langle \text{proof} \rangle$

**lemma** *InitDoneReds*:

$\llbracket \text{sh } C = \text{Some}(sfs, \text{Done}); P \vdash \langle \text{INIT } C' \ (Cs, \text{True}) \leftarrow e, (h, l, \text{sh}), b \rangle \rightarrow^* \langle e', s', b' \rangle \rrbracket$   
 $\implies P \vdash \langle \text{INIT } C' \ (C \# Cs, \text{False}) \leftarrow e, (h, l, \text{sh}), b \rangle \rightarrow^* \langle e', s', b' \rangle \langle \text{proof} \rangle$

**lemma** *InitProcessingReds*:

$\llbracket \text{sh } C = \text{Some}(sfs, \text{Processing}); P \vdash \langle \text{INIT } C' \ (Cs, \text{True}) \leftarrow e, (h, l, \text{sh}), b \rangle \rightarrow^* \langle e', s', b' \rangle \rrbracket$   
 $\implies P \vdash \langle \text{INIT } C' \ (C \# Cs, \text{False}) \leftarrow e, (h, l, \text{sh}), b \rangle \rightarrow^* \langle e', s', b' \rangle \langle \text{proof} \rangle$

**lemma** *InitErrorReds*:

$\llbracket \text{sh } C = \text{Some}(sfs, \text{Error}); P \vdash \langle \text{RI } (C, \text{THROW } \text{NoClassDefNotFoundError}); Cs \leftarrow e, (h, l, \text{sh}), b \rangle \rightarrow^* \langle e', s', b' \rangle \rrbracket$   
 $\implies P \vdash \langle \text{INIT } C' \ (C \# Cs, \text{False}) \leftarrow e, (h, l, \text{sh}), b \rangle \rightarrow^* \langle e', s', b' \rangle \langle \text{proof} \rangle$

**lemma** *InitObjectReds*:

$\llbracket \text{sh } C = \text{Some}(sfs, \text{Prepared}); C = \text{Object}; \text{sh}' = \text{sh}(C \mapsto (\text{sfs}, \text{Processing}));$   
 $P \vdash \langle \text{INIT } C' \ (C \# Cs, \text{True}) \leftarrow e, (h, l, \text{sh}'), b \rangle \rightarrow^* \langle e', s', b' \rangle \rrbracket$   
 $\implies P \vdash \langle \text{INIT } C' \ (C \# Cs, \text{False}) \leftarrow e, (h, l, \text{sh}), b \rangle \rightarrow^* \langle e', s', b' \rangle \langle \text{proof} \rangle$

**lemma** *InitNonObjectReds*:

$\llbracket \text{sh } C = \text{Some}(sfs, \text{Prepared}); C \neq \text{Object}; \text{class } P \ C = \text{Some } (D, r);$   
 $\text{sh}' = \text{sh}(C \mapsto (\text{sfs}, \text{Processing}));$   
 $P \vdash \langle \text{INIT } C' \ (D \# C \# Cs, \text{False}) \leftarrow e, (h, l, \text{sh}'), b \rangle \rightarrow^* \langle e', s', b' \rangle \rrbracket$   
 $\implies P \vdash \langle \text{INIT } C' \ (C \# Cs, \text{False}) \leftarrow e, (h, l, \text{sh}), b \rangle \rightarrow^* \langle e', s', b' \rangle \langle \text{proof} \rangle$

**lemma** *RedsInitRInit*:

$P \vdash \langle \text{RI } (C, C \cdot_s \text{clinit}(\square)); Cs \leftarrow e, (h, l, \text{sh}), b \rangle \rightarrow^* \langle e', s', b' \rangle$   
 $\implies P \vdash \langle \text{INIT } C' \ (C \# Cs, \text{True}) \leftarrow e, (h, l, \text{sh}), b \rangle \rightarrow^* \langle e', s', b' \rangle \langle \text{proof} \rangle$

**lemmas** *rtrancl-induct3* =

*rtrancl-induct*[of  $(ax, ay, az) \ (bx, by, bz), \text{split-format } (\text{complete}), \text{consumes } 1, \text{case-names refl step}$ ]

**lemma** *RInitReds*:

$P \vdash \langle e, s, b \rangle \rightarrow^* \langle e', s', b' \rangle$   
 $\implies P \vdash \langle \text{RI } (C, e); Cs \leftarrow e_0, s, b \rangle \rightarrow^* \langle \text{RI } (C, e'); Cs \leftarrow e_0, s', b' \rangle \langle \text{proof} \rangle$

**lemma** *RedsRInit*:

$\llbracket P \vdash \langle e_0, s_0, b_0 \rangle \rightarrow^* \langle \text{Val } v, (h_1, l_1, \text{sh}_1), b_1 \rangle;$   
 $\text{sh}_1 \ C = \text{Some } (sfs, i); \text{sh}_2 = \text{sh}_1(C \mapsto (\text{sfs}, \text{Done})); C' = \text{last}(C \# Cs);$   
 $P \vdash \langle \text{INIT } C' \ (Cs, \text{True}) \leftarrow e, (h_1, l_1, \text{sh}_2), b_1 \rangle \rightarrow^* \langle e', s', b' \rangle \rrbracket$   
 $\implies P \vdash \langle \text{RI } (C, e_0); Cs \leftarrow e, s_0, b_0 \rangle \rightarrow^* \langle e', s', b' \rangle \langle \text{proof} \rangle$



**lemma** *RInitInitThrowReds*:

$$\begin{aligned} & \llbracket P \vdash \langle e, s, b \rangle \rightarrow^* \langle \text{Throw } a, (h', l', sh'), b \rangle; \\ & \quad sh' C = \text{Some } (sfs, i); sh'' = sh'(C \mapsto (sfs, \text{Error})); \\ & \quad P \vdash \langle \text{RI } (D, \text{Throw } a); Cs \leftarrow e_0, (h', l', sh'), b \rangle \rightarrow^* \langle e_1, s_1, b_1 \rangle \rrbracket \\ & \implies P \vdash \langle \text{RI } (C, e); D \# Cs \leftarrow e_0, s, b \rangle \rightarrow^* \langle e_1, s_1, b_1 \rangle \langle \text{proof} \rangle \end{aligned}$$

**lemma** *RInitThrowReds*:

$$\begin{aligned} & \llbracket P \vdash \langle e, s, b \rangle \rightarrow^* \langle \text{Throw } a, (h', l', sh'), b \rangle; \\ & \quad sh' C = \text{Some}(sfs, i); sh'' = sh'(C \mapsto (sfs, \text{Error})) \rrbracket \\ & \implies P \vdash \langle \text{RI } (C, e); \text{Nil} \leftarrow e_0, s, b \rangle \rightarrow^* \langle \text{Throw } a, (h', l', sh'), b \rangle \langle \text{proof} \rangle \end{aligned}$$

## New

**lemma** *NewInitDoneReds*:

$$\begin{aligned} & \llbracket sh C = \text{Some } (sfs, \text{Done}); \text{new-Addr } h = \text{Some } a; \\ & \quad P \vdash C \text{ has-fields } \text{FDTs}; h' = h(a \mapsto \text{blank } P C) \rrbracket \\ & \implies P \vdash \langle \text{new } C, (h, l, sh), \text{False} \rangle \rightarrow^* \langle \text{addr } a, (h', l, sh), \text{False} \rangle \langle \text{proof} \rangle \end{aligned}$$

**lemma** *NewInitDoneReds2*:

$$\begin{aligned} & \llbracket sh C = \text{Some } (sfs, \text{Done}); \text{new-Addr } h = \text{None}; \text{is-class } P C \rrbracket \\ & \implies P \vdash \langle \text{new } C, (h, l, sh), \text{False} \rangle \rightarrow^* \langle \text{THROW OutOfMemory}, (h, l, sh), \text{False} \rangle \langle \text{proof} \rangle \end{aligned}$$

**lemma** *NewInitReds*:

**assumes**  $n\text{Done}$ :  $\nexists sfs. \text{shp } s C = \text{Some } (sfs, \text{Done})$   
**and** *INIT-steps*:  $P \vdash \langle \text{INIT } C ([C], \text{False}) \leftarrow \text{unit}, s, \text{False} \rangle \rightarrow^* \langle \text{Val } v', (h', l', sh'), b \rangle$   
**and** *Addr*:  $\text{new-Addr } h' = \text{Some } a$  **and** *has*:  $P \vdash C \text{ has-fields } \text{FDTs}$   
**and**  $h'$ :  $h' = h'(a \mapsto \text{blank } P C)$  **and** *cls-C*:  $\text{is-class } P C$   
**shows**  $P \vdash \langle \text{new } C, s, \text{False} \rangle \rightarrow^* \langle \text{addr } a, (h', l', sh'), \text{False} \rangle \langle \text{proof} \rangle$

**lemma** *NewInitOOMReds*:

**assumes**  $n\text{Done}$ :  $\nexists sfs. \text{shp } s C = \text{Some } (sfs, \text{Done})$   
**and** *INIT-steps*:  $P \vdash \langle \text{INIT } C ([C], \text{False}) \leftarrow \text{unit}, s, \text{False} \rangle \rightarrow^* \langle \text{Val } v', (h', l', sh'), b \rangle$   
**and** *Addr*:  $\text{new-Addr } h' = \text{None}$  **and** *cls-C*:  $\text{is-class } P C$   
**shows**  $P \vdash \langle \text{new } C, s, \text{False} \rangle \rightarrow^* \langle \text{THROW OutOfMemory}, (h', l', sh'), \text{False} \rangle \langle \text{proof} \rangle$

**lemma** *NewInitThrowReds*:

**assumes**  $n\text{Done}$ :  $\nexists sfs. \text{shp } s C = \text{Some } (sfs, \text{Done})$   
**and** *cls-C*:  $\text{is-class } P C$   
**and** *INIT-steps*:  $P \vdash \langle \text{INIT } C ([C], \text{False}) \leftarrow \text{unit}, s, \text{False} \rangle \rightarrow^* \langle \text{throw } a, s', b \rangle$   
**shows**  $P \vdash \langle \text{new } C, s, \text{False} \rangle \rightarrow^* \langle \text{throw } a, s', b \rangle \langle \text{proof} \rangle$

## Cast

**lemma** *CastReds*:

$$P \vdash \langle e, s, b \rangle \rightarrow^* \langle e', s', b' \rangle \implies P \vdash \langle \text{Cast } C e, s, b \rangle \rightarrow^* \langle \text{Cast } C e', s', b' \rangle \langle \text{proof} \rangle$$

**lemma** *CastRedsNull*:

$$P \vdash \langle e, s, b \rangle \rightarrow^* \langle \text{null}, s', b' \rangle \implies P \vdash \langle \text{Cast } C e, s, b \rangle \rightarrow^* \langle \text{null}, s', b' \rangle \langle \text{proof} \rangle$$

**lemma** *CastRedsAddr*:

$$\begin{aligned} & \llbracket P \vdash \langle e, s, b \rangle \rightarrow^* \langle \text{addr } a, s', b' \rangle; \text{hp } s' a = \text{Some}(D, fs); P \vdash D \preceq^* C \rrbracket \implies \\ & P \vdash \langle \text{Cast } C e, s, b \rangle \rightarrow^* \langle \text{addr } a, s', b' \rangle \langle \text{proof} \rangle \end{aligned}$$

**lemma** *CastRedsFail*:

$$\begin{aligned} & \llbracket P \vdash \langle e, s, b \rangle \rightarrow^* \langle \text{addr } a, s', b' \rangle; \text{hp } s' a = \text{Some}(D, fs); \neg P \vdash D \preceq^* C \rrbracket \implies \\ & P \vdash \langle \text{Cast } C e, s, b \rangle \rightarrow^* \langle \text{THROW ClassCast}, s', b' \rangle \langle \text{proof} \rangle \end{aligned}$$

**lemma** *CastRedsThrow*:

$$\llbracket P \vdash \langle e, s, b \rangle \rightarrow^* \langle \text{throw } a, s', b' \rangle \rrbracket \implies P \vdash \langle \text{Cast } C e, s, b \rangle \rightarrow^* \langle \text{throw } a, s', b' \rangle \langle \text{proof} \rangle$$

## LAss

**lemma** *LAssReds*:

$$P \vdash \langle e, s, b \rangle \rightarrow^* \langle e', s', b' \rangle \implies P \vdash \langle V := e, s, b \rangle \rightarrow^* \langle V := e', s', b' \rangle \langle \text{proof} \rangle$$

**lemma** *LAssRedsVal*:

$$\llbracket P \vdash \langle e, s, b \rangle \rightarrow^* \langle \text{Val } v, (h', l', sh'), b' \rangle \rrbracket \implies P \vdash \langle V := e, s, b \rangle \rightarrow^* \langle \text{unit}, (h', l' (V \mapsto v), sh'), b' \rangle \langle \text{proof} \rangle$$

**lemma** *LAssRedsThrow*:

$$\llbracket P \vdash \langle e, s, b \rangle \rightarrow^* \langle \text{throw } a, s', b' \rangle \rrbracket \implies P \vdash \langle V := e, s, b \rangle \rightarrow^* \langle \text{throw } a, s', b' \rangle \langle \text{proof} \rangle$$

## BinOp

**lemma** *BinOp1Reds*:

$$P \vdash \langle e, s, b \rangle \rightarrow^* \langle e', s', b' \rangle \implies P \vdash \langle e \ll \text{bop} \gg e_2, s, b \rangle \rightarrow^* \langle e' \ll \text{bop} \gg e_2, s', b' \rangle \langle \text{proof} \rangle$$

**lemma** *BinOp2Reds*:

$$P \vdash \langle e, s, b \rangle \rightarrow^* \langle e', s', b' \rangle \implies P \vdash \langle (\text{Val } v) \ll \text{bop} \gg e, s, b \rangle \rightarrow^* \langle (\text{Val } v) \ll \text{bop} \gg e', s', b' \rangle \langle \text{proof} \rangle$$

**lemma** *BinOpRedsVal*:

**assumes** *e1-steps*:  $P \vdash \langle e_1, s_0, b_0 \rangle \rightarrow^* \langle \text{Val } v_1, s_1, b_1 \rangle$

**and** *e2-steps*:  $P \vdash \langle e_2, s_1, b_1 \rangle \rightarrow^* \langle \text{Val } v_2, s_2, b_2 \rangle$

**and** *op*:  $\text{binop}(\text{bop}, v_1, v_2) = \text{Some } v$

**shows**  $P \vdash \langle e_1 \ll \text{bop} \gg e_2, s_0, b_0 \rangle \rightarrow^* \langle \text{Val } v, s_2, b_2 \rangle \langle \text{proof} \rangle$

**lemma** *BinOpRedsThrow1*:

$$P \vdash \langle e, s, b \rangle \rightarrow^* \langle \text{throw } e', s', b' \rangle \implies P \vdash \langle e \ll \text{bop} \gg e_2, s, b \rangle \rightarrow^* \langle \text{throw } e', s', b' \rangle \langle \text{proof} \rangle$$

**lemma** *BinOpRedsThrow2*:

**assumes** *e1-steps*:  $P \vdash \langle e_1, s_0, b_0 \rangle \rightarrow^* \langle \text{Val } v_1, s_1, b_1 \rangle$

**and** *e2-steps*:  $P \vdash \langle e_2, s_1, b_1 \rangle \rightarrow^* \langle \text{throw } e, s_2, b_2 \rangle$

**shows**  $P \vdash \langle e_1 \ll \text{bop} \gg e_2, s_0, b_0 \rangle \rightarrow^* \langle \text{throw } e, s_2, b_2 \rangle \langle \text{proof} \rangle$

## FAcc

**lemma** *FAccReds*:

$$P \vdash \langle e, s, b \rangle \rightarrow^* \langle e', s', b' \rangle \implies P \vdash \langle e \cdot F\{D\}, s, b \rangle \rightarrow^* \langle e' \cdot F\{D\}, s', b' \rangle \langle \text{proof} \rangle$$

**lemma** *FAccRedsVal*:

$$\llbracket P \vdash \langle e, s, b \rangle \rightarrow^* \langle \text{addr } a, s', b' \rangle; \text{hp } s' a = \text{Some}(C, fs); fs(F, D) = \text{Some } v;$$

$$P \vdash C \text{ has } F, \text{NonStatic}:t \text{ in } D \rrbracket$$

$$\implies P \vdash \langle e \cdot F\{D\}, s, b \rangle \rightarrow^* \langle \text{Val } v, s', b' \rangle \langle \text{proof} \rangle$$

**lemma** *FAccRedsNull*:

$$P \vdash \langle e, s, b \rangle \rightarrow^* \langle \text{null}, s', b' \rangle \implies P \vdash \langle e \cdot F\{D\}, s, b \rangle \rightarrow^* \langle \text{THROW NullPointer}, s', b' \rangle \langle \text{proof} \rangle$$

**lemma** *FAccRedsNone*:

$$\llbracket P \vdash \langle e, s, b \rangle \rightarrow^* \langle \text{addr } a, s', b' \rangle;$$

$$\text{hp } s' a = \text{Some}(C, fs);$$

$$\neg(\exists b t. P \vdash C \text{ has } F, b:t \text{ in } D) \rrbracket$$

$$\implies P \vdash \langle e \cdot F\{D\}, s, b \rangle \rightarrow^* \langle \text{THROW NoSuchFieldError}, s', b' \rangle \langle \text{proof} \rangle$$

**lemma** *FAccRedsStatic*:

$$\llbracket P \vdash \langle e, s, b \rangle \rightarrow^* \langle \text{addr } a, s', b' \rangle;$$

$$\text{hp } s' a = \text{Some}(C, fs);$$

$$P \vdash C \text{ has } F, \text{Static}:t \text{ in } D \rrbracket$$

$$\implies P \vdash \langle e \cdot F\{D\}, s, b \rangle \rightarrow^* \langle \text{THROW IncompatibleClassChangeError}, s', b' \rangle \langle \text{proof} \rangle$$

**lemma** *FAccRedsThrow*:

$$P \vdash \langle e, s, b \rangle \rightarrow^* \langle \text{throw } a, s', b' \rangle \implies P \vdash \langle e \cdot F\{D\}, s, b \rangle \rightarrow^* \langle \text{throw } a, s', b' \rangle \langle \text{proof} \rangle$$

## SFAcc

**lemma** *SFAccReds*:

$$\llbracket P \vdash C \text{ has } F, \text{Static}:t \text{ in } D;$$

$$\text{shp } s D = \text{Some}(sfs, Done); sfs F = \text{Some } v \rrbracket$$

$$\implies P \vdash \langle C \cdot_s F\{D\}, s, True \rangle \rightarrow^* \langle \text{Val } v, s, False \rangle \langle \text{proof} \rangle$$

**lemma** *SFAccRedsNone*:

$$\neg(\exists b t. P \vdash C \text{ has } F, b:t \text{ in } D)$$

$\implies P \vdash \langle C \cdot_s F\{D\}, s, b \rangle \rightarrow^* \langle \text{THROW NoSuchFieldError}, s, \text{False} \rangle \langle \text{proof} \rangle$   
**lemma SFAccRedsNonStatic:**  
 $P \vdash C$  has  $F, \text{NonStatic}:t$  in  $D$   
 $\implies P \vdash \langle C \cdot_s F\{D\}, s, b \rangle \rightarrow^* \langle \text{THROW IncompatibleClassChangeError}, s, \text{False} \rangle \langle \text{proof} \rangle$   
**lemma SFAccInitDoneReds:**  
**assumes**  $cF: P \vdash C$  has  $F, \text{Static}:t$  in  $D$   
**and**  $shp: shp\ s\ D = \text{Some}(sfs, \text{Done})$  **and**  $sfs: sfs\ F = \text{Some}\ v$   
**shows**  $P \vdash \langle C \cdot_s F\{D\}, s, b \rangle \rightarrow^* \langle \text{Val}\ v, s, \text{False} \rangle \langle \text{proof} \rangle$   
**lemma SFAccInitReds:**  
**assumes**  $cF: P \vdash C$  has  $F, \text{Static}:t$  in  $D$   
**and**  $nDone: \nexists sfs. shp\ s\ D = \text{Some}(sfs, \text{Done})$   
**and**  $\text{INIT-steps}: P \vdash \langle \text{INIT}\ D\ ([D], \text{False}) \leftarrow \text{unit}, s, \text{False} \rangle \rightarrow^* \langle \text{Val}\ v', s', b' \rangle$   
**and**  $shp': shp\ s' D = \text{Some}(sfs, i)$  **and**  $sfs: sfs\ F = \text{Some}\ v$   
**shows**  $P \vdash \langle C \cdot_s F\{D\}, s, \text{False} \rangle \rightarrow^* \langle \text{Val}\ v, s', \text{False} \rangle \langle \text{proof} \rangle$   
**lemma SFAccInitThrowReds:**  
 $\llbracket P \vdash C$  has  $F, \text{Static}:t$  in  $D$ ;  
 $\nexists sfs. shp\ s\ D = \text{Some}(sfs, \text{Done});$   
 $P \vdash \langle \text{INIT}\ D\ ([D], \text{False}) \leftarrow \text{unit}, s, \text{False} \rangle \rightarrow^* \langle \text{throw}\ a, s', b' \rangle \rrbracket$   
 $\implies P \vdash \langle C \cdot_s F\{D\}, s, \text{False} \rangle \rightarrow^* \langle \text{throw}\ a, s', b' \rangle \langle \text{proof} \rangle$

## FAss

**lemma FAssReds1:**  
 $P \vdash \langle e, s, b \rangle \rightarrow^* \langle e', s', b' \rangle \implies P \vdash \langle e \cdot F\{D\} := e_2, s, b \rangle \rightarrow^* \langle e' \cdot F\{D\} := e_2, s', b' \rangle \langle \text{proof} \rangle$   
**lemma FAssReds2:**  
 $P \vdash \langle e, s, b \rangle \rightarrow^* \langle e', s', b' \rangle \implies P \vdash \langle \text{Val}\ v \cdot F\{D\} := e, s, b \rangle \rightarrow^* \langle \text{Val}\ v \cdot F\{D\} := e', s', b' \rangle \langle \text{proof} \rangle$   
**lemma FAssRedsVal:**  
**assumes**  $e_1\text{-steps}: P \vdash \langle e_1, s_0, b_0 \rangle \rightarrow^* \langle \text{addr}\ a, s_1, b_1 \rangle$   
**and**  $e_2\text{-steps}: P \vdash \langle e_2, s_1, b_1 \rangle \rightarrow^* \langle \text{Val}\ v, (h_2, l_2, sh_2), b_2 \rangle$   
**and**  $cF: P \vdash C$  has  $F, \text{NonStatic}:t$  in  $D$  **and**  $hC: \text{Some}(C, fs) = h_2\ a$   
**shows**  $P \vdash \langle e_1 \cdot F\{D\} := e_2, s_0, b_0 \rangle \rightarrow^* \langle \text{unit}, (h_2(a \mapsto (C, fs((F, D) \mapsto v))), l_2, sh_2), b_2 \rangle \langle \text{proof} \rangle$   
**lemma FAssRedsNull:**  
**assumes**  $e_1\text{-steps}: P \vdash \langle e_1, s_0, b_0 \rangle \rightarrow^* \langle \text{null}, s_1, b_1 \rangle$   
**and**  $e_2\text{-steps}: P \vdash \langle e_2, s_1, b_1 \rangle \rightarrow^* \langle \text{Val}\ v, s_2, b_2 \rangle$   
**shows**  $P \vdash \langle e_1 \cdot F\{D\} := e_2, s_0, b_0 \rangle \rightarrow^* \langle \text{THROW NullPointer}, s_2, b_2 \rangle \langle \text{proof} \rangle$   
**lemma FAssRedsThrow1:**  
 $P \vdash \langle e, s, b \rangle \rightarrow^* \langle \text{throw}\ e', s', b' \rangle \implies P \vdash \langle e \cdot F\{D\} := e_2, s, b \rangle \rightarrow^* \langle \text{throw}\ e', s', b' \rangle \langle \text{proof} \rangle$   
**lemma FAssRedsThrow2:**  
**assumes**  $e_1\text{-steps}: P \vdash \langle e_1, s_0, b_0 \rangle \rightarrow^* \langle \text{Val}\ v, s_1, b_1 \rangle$   
**and**  $e_2\text{-steps}: P \vdash \langle e_2, s_1, b_1 \rangle \rightarrow^* \langle \text{throw}\ e, s_2, b_2 \rangle$   
**shows**  $P \vdash \langle e_1 \cdot F\{D\} := e_2, s_0, b_0 \rangle \rightarrow^* \langle \text{throw}\ e, s_2, b_2 \rangle \langle \text{proof} \rangle$   
**lemma FAssRedsNone:**  
**assumes**  $e_1\text{-steps}: P \vdash \langle e_1, s_0, b_0 \rangle \rightarrow^* \langle \text{addr}\ a, s_1, b_1 \rangle$   
**and**  $e_2\text{-steps}: P \vdash \langle e_2, s_1, b_1 \rangle \rightarrow^* \langle \text{Val}\ v, (h_2, l_2, sh_2), b_2 \rangle$   
**and**  $hC: h_2\ a = \text{Some}(C, fs)$  **and**  $ncF: \neg(\exists b\ t. P \vdash C$  has  $F, b:t$  in  $D)$   
**shows**  $P \vdash \langle e_1 \cdot F\{D\} := e_2, s_0, b_0 \rangle \rightarrow^* \langle \text{THROW NoSuchFieldError}, (h_2, l_2, sh_2), b_2 \rangle \langle \text{proof} \rangle$   
**lemma FAssRedsStatic:**  
**assumes**  $e_1\text{-steps}: P \vdash \langle e_1, s_0, b_0 \rangle \rightarrow^* \langle \text{addr}\ a, s_1, b_1 \rangle$   
**and**  $e_2\text{-steps}: P \vdash \langle e_2, s_1, b_1 \rangle \rightarrow^* \langle \text{Val}\ v, (h_2, l_2, sh_2), b_2 \rangle$   
**and**  $hC: h_2\ a = \text{Some}(C, fs)$  **and**  $cF\text{-Static}: P \vdash C$  has  $F, \text{Static}:t$  in  $D$   
**shows**  $P \vdash \langle e_1 \cdot F\{D\} := e_2, s_0, b_0 \rangle \rightarrow^* \langle \text{THROW IncompatibleClassChangeError}, (h_2, l_2, sh_2), b_2 \rangle \langle \text{proof} \rangle$

**SFAss****lemma** *SFAssReds*:

$$P \vdash \langle e, s, b \rangle \rightarrow^* \langle e', s', b' \rangle \implies P \vdash \langle C \cdot_s F \{D\} := e, s, b \rangle \rightarrow^* \langle C \cdot_s F \{D\} := e', s', b' \rangle \langle \text{proof} \rangle$$

**lemma** *SFAssRedsVal*:**assumes** *e2-steps*:  $P \vdash \langle e_2, s_0, b_0 \rangle \rightarrow^* \langle \text{Val } v, (h_2, l_2, sh_2), b_2 \rangle$ **and** *cF*:  $P \vdash C \text{ has } F, \text{Static}:t \text{ in } D$ **and** *shD*:  $sh_2 D = \lfloor (sfs, Done) \rfloor$ **shows**  $P \vdash \langle C \cdot_s F \{D\} := e_2, s_0, b_0 \rangle \rightarrow^* \langle \text{unit}, (h_2, l_2, sh_2(D \mapsto (sfs(F \mapsto v), Done))), False \rangle \langle \text{proof} \rangle$ **lemma** *SFAssRedsThrow*:

$$\llbracket P \vdash \langle e_2, s_0, b_0 \rangle \rightarrow^* \langle \text{throw } e, s_2, b_2 \rangle \rrbracket$$

$$\implies P \vdash \langle C \cdot_s F \{D\} := e_2, s_0, b_0 \rangle \rightarrow^* \langle \text{throw } e, s_2, b_2 \rangle \langle \text{proof} \rangle$$

**lemma** *SFAssRedsNone*:

$$\llbracket P \vdash \langle e_2, s_0, b_0 \rangle \rightarrow^* \langle \text{Val } v, (h_2, l_2, sh_2), b_2 \rangle; \neg(\exists b t. P \vdash C \text{ has } F, b:t \text{ in } D) \rrbracket \implies$$

$$P \vdash \langle C \cdot_s F \{D\} := e_2, s_0, b_0 \rangle \rightarrow^* \langle \text{THROW NoSuchFieldError}, (h_2, l_2, sh_2), False \rangle \langle \text{proof} \rangle$$

**lemma** *SFAssRedsNonStatic*:

$$\llbracket P \vdash \langle e_2, s_0, b_0 \rangle \rightarrow^* \langle \text{Val } v, (h_2, l_2, sh_2), b_2 \rangle; P \vdash C \text{ has } F, \text{NonStatic}:t \text{ in } D \rrbracket \implies$$

$$P \vdash \langle C \cdot_s F \{D\} := e_2, s_0, b_0 \rangle \rightarrow^* \langle \text{THROW IncompatibleClassChangeError}, (h_2, l_2, sh_2), False \rangle \langle \text{proof} \rangle$$

**lemma** *SFAssInitReds*:**assumes** *e2-steps*:  $P \vdash \langle e_2, s_0, b_0 \rangle \rightarrow^* \langle \text{Val } v, (h_2, l_2, sh_2), False \rangle$ **and** *cF*:  $P \vdash C \text{ has } F, \text{Static}:t \text{ in } D$ **and** *nDone*:  $\nexists sfs. sh_2 D = \text{Some}(sfs, Done)$ **and** *INIT-steps*:  $P \vdash \langle \text{INIT } D \ ([D], False) \leftarrow \text{unit}, (h_2, l_2, sh_2), False \rangle \rightarrow^* \langle \text{Val } v', (h', l', sh'), b' \rangle$ **and** *sh'D*:  $sh' D = \text{Some}(sfs, i)$ **and** *sfs'*:  $sfs' = sfs(F \mapsto v)$  **and** *sh''*:  $sh'' = sh'(D \mapsto (sfs', i))$ **shows**  $P \vdash \langle C \cdot_s F \{D\} := e_2, s_0, b_0 \rangle \rightarrow^* \langle \text{unit}, (h', l', sh''), False \rangle \langle \text{proof} \rangle$ **lemma** *SFAssInitThrowReds*:**assumes** *e2-steps*:  $P \vdash \langle e_2, s_0, b_0 \rangle \rightarrow^* \langle \text{Val } v, (h_2, l_2, sh_2), False \rangle$ **and** *cF*:  $P \vdash C \text{ has } F, \text{Static}:t \text{ in } D$ **and** *nDone*:  $\nexists sfs. sh_2 D = \text{Some}(sfs, Done)$ **and** *INIT-steps*:  $P \vdash \langle \text{INIT } D \ ([D], False) \leftarrow \text{unit}, (h_2, l_2, sh_2), False \rangle \rightarrow^* \langle \text{throw } a, s', b' \rangle$ **shows**  $P \vdash \langle C \cdot_s F \{D\} := e_2, s_0, b_0 \rangle \rightarrow^* \langle \text{throw } a, s', b' \rangle \langle \text{proof} \rangle$ 

;;

**lemma** *SeqReds*:

$$P \vdash \langle e, s, b \rangle \rightarrow^* \langle e', s', b' \rangle \implies P \vdash \langle e;; e_2, s, b \rangle \rightarrow^* \langle e'; e_2, s', b' \rangle \langle \text{proof} \rangle$$

**lemma** *SeqRedsThrow*:

$$P \vdash \langle e, s, b \rangle \rightarrow^* \langle \text{throw } e', s', b' \rangle \implies P \vdash \langle e;; e_2, s, b \rangle \rightarrow^* \langle \text{throw } e', s', b' \rangle \langle \text{proof} \rangle$$

**lemma** *SeqReds2*:**assumes** *e1-steps*:  $P \vdash \langle e_1, s_0, b_0 \rangle \rightarrow^* \langle \text{Val } v_1, s_1, b_1 \rangle$ **and** *e2-steps*:  $P \vdash \langle e_2, s_1, b_1 \rangle \rightarrow^* \langle e_2', s_2, b_2 \rangle$ **shows**  $P \vdash \langle e_1;; e_2, s_0, b_0 \rangle \rightarrow^* \langle e_2', s_2, b_2 \rangle \langle \text{proof} \rangle$ **If****lemma** *CondReds*:

$$P \vdash \langle e, s, b \rangle \rightarrow^* \langle e', s', b' \rangle \implies P \vdash \langle \text{if } (e) e_1 \text{ else } e_2, s, b \rangle \rightarrow^* \langle \text{if } (e') e_1 \text{ else } e_2, s', b' \rangle \langle \text{proof} \rangle$$

**lemma** *CondRedsThrow*:

$$P \vdash \langle e, s, b \rangle \rightarrow^* \langle \text{throw } a, s', b' \rangle \implies P \vdash \langle \text{if } (e) e_1 \text{ else } e_2, s, b \rangle \rightarrow^* \langle \text{throw } a, s', b' \rangle \langle \text{proof} \rangle$$

**lemma** *CondReds2T*:**assumes** *e-steps*:  $P \vdash \langle e, s_0, b_0 \rangle \rightarrow^* \langle \text{true}, s_1, b_1 \rangle$ **and** *e1-steps*:  $P \vdash \langle e_1, s_1, b_1 \rangle \rightarrow^* \langle e', s_2, b_2 \rangle$

shows  $P \vdash \langle \text{if } (e) e_1 \text{ else } e_2, s_0, b_0 \rangle \rightarrow^* \langle e', s_2, b_2 \rangle \langle \text{proof} \rangle$

lemma *CondReds2F*:

assumes  $e$ -steps:  $P \vdash \langle e, s_0, b_0 \rangle \rightarrow^* \langle \text{false}, s_1, b_1 \rangle$

and  $e_2$ -steps:  $P \vdash \langle e_2, s_1, b_1 \rangle \rightarrow^* \langle e', s_2, b_2 \rangle$

shows  $P \vdash \langle \text{if } (e) e_1 \text{ else } e_2, s_0, b_0 \rangle \rightarrow^* \langle e', s_2, b_2 \rangle \langle \text{proof} \rangle$

## While

lemma *WhileFReds*:

assumes  $b$ -steps:  $P \vdash \langle b, s, b_0 \rangle \rightarrow^* \langle \text{false}, s', b' \rangle$

shows  $P \vdash \langle \text{while } (b) c, s, b_0 \rangle \rightarrow^* \langle \text{unit}, s', b' \rangle \langle \text{proof} \rangle$

lemma *WhileRedsThrow*:

assumes  $b$ -steps:  $P \vdash \langle b, s, b_0 \rangle \rightarrow^* \langle \text{throw } e, s', b' \rangle$

shows  $P \vdash \langle \text{while } (b) c, s, b_0 \rangle \rightarrow^* \langle \text{throw } e, s', b' \rangle \langle \text{proof} \rangle$

lemma *WhileTReds*:

assumes  $b$ -steps:  $P \vdash \langle b, s_0, b_0 \rangle \rightarrow^* \langle \text{true}, s_1, b_1 \rangle$

and  $c$ -steps:  $P \vdash \langle c, s_1, b_1 \rangle \rightarrow^* \langle \text{Val } v_1, s_2, b_2 \rangle$

and  $\text{while}$ -steps:  $P \vdash \langle \text{while } (b) c, s_2, b_2 \rangle \rightarrow^* \langle e, s_3, b_3 \rangle$

shows  $P \vdash \langle \text{while } (b) c, s_0, b_0 \rangle \rightarrow^* \langle e, s_3, b_3 \rangle \langle \text{proof} \rangle$

lemma *WhileTRedsThrow*:

assumes  $b$ -steps:  $P \vdash \langle b, s_0, b_0 \rangle \rightarrow^* \langle \text{true}, s_1, b_1 \rangle$

and  $c$ -steps:  $P \vdash \langle c, s_1, b_1 \rangle \rightarrow^* \langle \text{throw } e, s_2, b_2 \rangle$

shows  $P \vdash \langle \text{while } (b) c, s_0, b_0 \rangle \rightarrow^* \langle \text{throw } e, s_2, b_2 \rangle \langle \text{proof} \rangle$

## Throw

lemma *ThrowReds*:

$P \vdash \langle e, s, b \rangle \rightarrow^* \langle e', s', b' \rangle \implies P \vdash \langle \text{throw } e, s, b \rangle \rightarrow^* \langle \text{throw } e', s', b' \rangle \langle \text{proof} \rangle$

lemma *ThrowRedsNull*:

$P \vdash \langle e, s, b \rangle \rightarrow^* \langle \text{null}, s', b' \rangle \implies P \vdash \langle \text{throw } e, s, b \rangle \rightarrow^* \langle \text{THROW NullPointer}, s', b' \rangle \langle \text{proof} \rangle$

lemma *ThrowRedsThrow*:

$P \vdash \langle e, s, b \rangle \rightarrow^* \langle \text{throw } a, s', b' \rangle \implies P \vdash \langle \text{throw } e, s, b \rangle \rightarrow^* \langle \text{throw } a, s', b' \rangle \langle \text{proof} \rangle$

## InitBlock

lemma *InitBlockReds-aux*:

$P \vdash \langle e, s, b \rangle \rightarrow^* \langle e', s', b' \rangle \implies$

$\forall h \ l \ sh \ h' \ l' \ sh' \ v. s = (h, l(V \mapsto v), sh) \longrightarrow s' = (h', l', sh') \longrightarrow$

$P \vdash \langle \{ V:T := \text{Val } v; e \}, (h, l, sh), b \rangle \rightarrow^* \langle \{ V:T := \text{Val}(\text{the}(l' V)); e' \}, (h', l'(V := (l V)), sh'), b' \rangle \langle \text{proof} \rangle$

lemma *InitBlockReds*:

$P \vdash \langle e, (h, l(V \mapsto v), sh), b \rangle \rightarrow^* \langle e', (h', l', sh'), b' \rangle \implies$

$P \vdash \langle \{ V:T := \text{Val } v; e \}, (h, l, sh), b \rangle \rightarrow^* \langle \{ V:T := \text{Val}(\text{the}(l' V)); e' \}, (h', l'(V := (l V)), sh'), b' \rangle \langle \text{proof} \rangle$

lemma *InitBlockRedsFinal*:

$\llbracket P \vdash \langle e, (h, l(V \mapsto v), sh), b \rangle \rightarrow^* \langle e', (h', l', sh'), b' \rangle; \text{final } e' \rrbracket \implies$

$P \vdash \langle \{ V:T := \text{Val } v; e \}, (h, l, sh), b \rangle \rightarrow^* \langle e', (h', l'(V := l V), sh'), b' \rangle \langle \text{proof} \rangle$

## Block

lemmas *converse-rtranclE3 = converse-rtranclE* [of  $(xa, xb, xc)$   $(za, zb, zc)$ , *split-rule*]

lemma *BlockRedsFinal*:

assumes  $\text{reds}$ :  $P \vdash \langle e_0, s_0, b_0 \rangle \rightarrow^* \langle e_2, (h_2, l_2, sh_2), b_2 \rangle$  and  $\text{fin}$ :  $\text{final } e_2$

shows  $\bigwedge h_0 \ l_0 \ sh_0. s_0 = (h_0, l_0(V := \text{None}), sh_0) \implies P \vdash \langle \{ V:T; e_0 \}, (h_0, l_0, sh_0), b_0 \rangle \rightarrow^* \langle e_2, (h_2, l_2(V := l_0 V), sh_2), b_2 \rangle \langle \text{proof} \rangle$

### try-catch

**lemma TryReds:**

$$P \vdash \langle e, s, b \rangle \rightarrow^* \langle e', s', b' \rangle \implies P \vdash \langle \text{try } e \text{ catch}(C \ V) \ e_2, s, b \rangle \rightarrow^* \langle \text{try } e' \text{ catch}(C \ V) \ e_2, s', b' \rangle \langle \text{proof} \rangle$$

**lemma TryRedsVal:**

$$P \vdash \langle e, s, b \rangle \rightarrow^* \langle \text{Val } v, s', b' \rangle \implies P \vdash \langle \text{try } e \text{ catch}(C \ V) \ e_2, s, b \rangle \rightarrow^* \langle \text{Val } v, s', b' \rangle \langle \text{proof} \rangle$$

**lemma TryCatchRedsFinal:**

**assumes**  $e_1$ -steps:  $P \vdash \langle e_1, s_0, b_0 \rangle \rightarrow^* \langle \text{Throw } a, (h_1, l_1, sh_1), b_1 \rangle$

**and**  $h_1 a$ :  $h_1 a = \text{Some}(D, fs)$  **and**  $sub$ :  $P \vdash D \preceq^* C$

**and**  $e_2$ -steps:  $P \vdash \langle e_2, (h_1, l_1(V \mapsto \text{Addr } a), sh_1), b_1 \rangle \rightarrow^* \langle e_2', (h_2, l_2, sh_2), b_2 \rangle$

**and**  $final$ :  $final \ e_2'$

**shows**  $P \vdash \langle \text{try } e_1 \text{ catch}(C \ V) \ e_2, s_0, b_0 \rangle \rightarrow^* \langle e_2', (h_2, (l_2::\text{locals})(V := l_1 \ V), sh_2), b_2 \rangle \langle \text{proof} \rangle$

**lemma TryRedsFail:**

$$\llbracket P \vdash \langle e_1, s, b \rangle \rightarrow^* \langle \text{Throw } a, (h, l, sh), b' \rangle; h a = \text{Some}(D, fs); \neg P \vdash D \preceq^* C \rrbracket$$

$$\implies P \vdash \langle \text{try } e_1 \text{ catch}(C \ V) \ e_2, s, b \rangle \rightarrow^* \langle \text{Throw } a, (h, l, sh), b' \rangle \langle \text{proof} \rangle$$

### List

**lemma ListReds1:**

$$P \vdash \langle e, s, b \rangle \rightarrow^* \langle e', s', b' \rangle \implies P \vdash \langle e \# es, s, b \rangle [\rightarrow]^* \langle e' \# es, s', b' \rangle \langle \text{proof} \rangle$$

**lemma ListReds2:**

$$P \vdash \langle es, s, b \rangle [\rightarrow]^* \langle es', s', b' \rangle \implies P \vdash \langle \text{Val } v \# es, s, b \rangle [\rightarrow]^* \langle \text{Val } v \# es', s', b' \rangle \langle \text{proof} \rangle$$

**lemma ListRedsVal:**

$$\llbracket P \vdash \langle e, s_0, b_0 \rangle \rightarrow^* \langle \text{Val } v, s_1, b_1 \rangle; P \vdash \langle es, s_1, b_1 \rangle [\rightarrow]^* \langle es', s_2, b_2 \rangle \rrbracket$$

$$\implies P \vdash \langle e \# es, s_0, b_0 \rangle [\rightarrow]^* \langle \text{Val } v \# es', s_2, b_2 \rangle \langle \text{proof} \rangle$$

### Call

First a few lemmas on what happens to free variables during redction.

**lemma assumes**  $wf$ :  $wuf\text{-}J\text{-prog } P$

**shows**  $Red\text{-}fv$ :  $P \vdash \langle e, (h, l, sh), b \rangle \rightarrow \langle e', (h', l', sh'), b' \rangle \implies fv \ e' \subseteq fv \ e$

**and**  $P \vdash \langle es, (h, l, sh), b \rangle [\rightarrow] \langle es', (h', l', sh'), b' \rangle \implies fvs \ es' \subseteq fvs \ es \langle \text{proof} \rangle$

**lemma Red-dom-lcl:**

$P \vdash \langle e, (h, l, sh), b \rangle \rightarrow \langle e', (h', l', sh'), b' \rangle \implies dom \ l' \subseteq dom \ l \cup fv \ e$  **and**

$P \vdash \langle es, (h, l, sh), b \rangle [\rightarrow] \langle es', (h', l', sh'), b' \rangle \implies dom \ l' \subseteq dom \ l \cup fvs \ es \langle \text{proof} \rangle$

**lemma Reds-dom-lcl:**

**assumes**  $wf$ :  $wuf\text{-}J\text{-prog } P$

**shows**  $P \vdash \langle e, (h, l, sh), b \rangle \rightarrow^* \langle e', (h', l', sh'), b' \rangle \implies dom \ l' \subseteq dom \ l \cup fv \ e \langle \text{proof} \rangle$

Now a few lemmas on the behaviour of blocks during reduction.

**lemma override-on-upd-lemma:**

$$(\text{override-on } f \ (g(a \mapsto b)) \ A)(a := g \ a) = \text{override-on } f \ g \ (\text{insert } a \ A) \langle \text{proof} \rangle$$

**lemma blocksReds:**

$\bigwedge l. \llbracket \text{length } Vs = \text{length } Ts; \text{length } vs = \text{length } Ts; \text{distinct } Vs;$

$$P \vdash \langle e, (h, l(Vs [\rightarrow] vs), sh), b \rangle \rightarrow^* \langle e', (h', l', sh'), b' \rangle \rrbracket$$

$$\implies P \vdash \langle \text{blocks}(Vs, Ts, vs, e), (h, l, sh), b \rangle \rightarrow^* \langle \text{blocks}(Vs, Ts, \text{map } (the \circ l') \ Vs, e'), (h', \text{override-on } l' \ l \ (\text{set } Vs), sh'), b' \rangle \langle \text{proof} \rangle$$

**lemma blocksFinal:**

$$\bigwedge l. \llbracket \text{length } Vs = \text{length } Ts; \text{length } vs = \text{length } Ts; \text{final } e \rrbracket \implies$$

$$P \vdash \langle \text{blocks}(Vs, Ts, vs, e), (h, l, sh), b \rangle \rightarrow^* \langle e, (h, l, sh), b \rangle \langle \text{proof} \rangle$$

**lemma** *blocksRedsFinal*:

**assumes** *wf*:  $\text{length } Vs = \text{length } Ts$   $\text{length } vs = \text{length } Ts$  *distinct*  $Vs$   
**and** *reds*:  $P \vdash \langle e, (h, l(Vs \mapsto vs), sh), b \rangle \rightarrow^* \langle e', (h', l', sh'), b' \rangle$   
**and** *fin*: *final*  $e'$  **and**  $l''$ :  $l'' = \text{override-on } l' \ l$  (*set*  $Vs$ )  
**shows**  $P \vdash \langle \text{blocks}(Vs, Ts, vs, e), (h, l, sh), b \rangle \rightarrow^* \langle e', (h', l'', sh'), b' \rangle \langle \text{proof} \rangle$

An now the actual method call reduction lemmas.

**lemma** *CallRedsObj*:

$P \vdash \langle e, s, b \rangle \rightarrow^* \langle e', s', b' \rangle \implies P \vdash \langle e \cdot M(es), s, b \rangle \rightarrow^* \langle e' \cdot M(es), s', b' \rangle \langle \text{proof} \rangle$

**lemma** *CallRedsParams*:

$P \vdash \langle es, s, b \rangle \mapsto^* \langle es', s', b' \rangle \implies P \vdash \langle (Val \ v) \cdot M(es), s, b \rangle \rightarrow^* \langle (Val \ v) \cdot M(es'), s', b' \rangle \langle \text{proof} \rangle$

**lemma** *CallRedsFinal*:

**assumes** *wwf*: *wwf-J-prog*  $P$   
**and**  $P \vdash \langle e, s_0, b_0 \rangle \rightarrow^* \langle \text{addr } a, s_1, b_1 \rangle$   
 $P \vdash \langle es, s_1, b_1 \rangle \mapsto^* \langle \text{map } Val \ vs, (h_2, l_2, sh_2), b_2 \rangle$   
 $h_2 \ a = \text{Some}(C.fs)$   $P \vdash C \text{ sees } M, \text{NonStatic}: Ts \rightarrow T = (pns, body)$  *in*  $D$   
 $\text{size } vs = \text{size } pns$   
**and**  $l_2'$ :  $l_2' = [this \mapsto \text{Addr } a, pns \mapsto vs]$   
**and** *body*:  $P \vdash \langle \text{body}, (h_2, l_2', sh_2), b_2 \rangle \rightarrow^* \langle ef, (h_3, l_3, sh_3), b_3 \rangle$   
**and** *final*  $ef$   
**shows**  $P \vdash \langle e \cdot M(es), s_0, b_0 \rangle \rightarrow^* \langle ef, (h_3, l_2, sh_3), b_3 \rangle \langle \text{proof} \rangle$

**lemma** *CallRedsThrowParams*:

**assumes** *e-steps*:  $P \vdash \langle e, s_0, b_0 \rangle \rightarrow^* \langle Val \ v, s_1, b_1 \rangle$   
**and** *es-steps*:  $P \vdash \langle es, s_1, b_1 \rangle \mapsto^* \langle \text{map } Val \ vs_1 \ @ \ \text{throw } a \ \#, \ es_2, s_2, b_2 \rangle$   
**shows**  $P \vdash \langle e \cdot M(es), s_0, b_0 \rangle \rightarrow^* \langle \text{throw } a, s_2, b_2 \rangle \langle \text{proof} \rangle$

**lemma** *CallRedsThrowObj*:

$P \vdash \langle e, s_0, b_0 \rangle \rightarrow^* \langle \text{throw } a, s_1, b_1 \rangle \implies P \vdash \langle e \cdot M(es), s_0, b_0 \rangle \rightarrow^* \langle \text{throw } a, s_1, b_1 \rangle \langle \text{proof} \rangle$

**lemma** *CallRedsNull*:

**assumes** *e-steps*:  $P \vdash \langle e, s_0, b_0 \rangle \rightarrow^* \langle null, s_1, b_1 \rangle$   
**and** *es-steps*:  $P \vdash \langle es, s_1, b_1 \rangle \mapsto^* \langle \text{map } Val \ vs, s_2, b_2 \rangle$   
**shows**  $P \vdash \langle e \cdot M(es), s_0, b_0 \rangle \rightarrow^* \langle \text{THROW } \text{NullPointer}, s_2, b_2 \rangle \langle \text{proof} \rangle$

**lemma** *CallRedsNone*:

**assumes** *e-steps*:  $P \vdash \langle e, s, b \rangle \rightarrow^* \langle \text{addr } a, s_1, b_1 \rangle$   
**and** *es-steps*:  $P \vdash \langle es, s_1, b_1 \rangle \mapsto^* \langle \text{map } Val \ vs, s_2, b_2 \rangle$   
**and**  $hp_2 a$ :  $hp \ s_2 \ a = \text{Some}(C.fs)$   
**and**  $ncM$ :  $\neg(\exists b \ Ts \ T \ m \ D. P \vdash C \text{ sees } M, b: Ts \rightarrow T = m \ \text{in } D)$   
**shows**  $P \vdash \langle e \cdot M(es), s, b \rangle \rightarrow^* \langle \text{THROW } \text{NoSuchMethodError}, s_2, b_2 \rangle \langle \text{proof} \rangle$

**lemma** *CallRedsStatic*:

**assumes** *e-steps*:  $P \vdash \langle e, s, b \rangle \rightarrow^* \langle \text{addr } a, s_1, b_1 \rangle$   
**and** *es-steps*:  $P \vdash \langle es, s_1, b_1 \rangle \mapsto^* \langle \text{map } Val \ vs, s_2, b_2 \rangle$   
**and**  $hp_2 a$ :  $hp \ s_2 \ a = \text{Some}(C.fs)$   
**and** *cM-Static*:  $P \vdash C \text{ sees } M, \text{Static}: Ts \rightarrow T = m \ \text{in } D$   
**shows**  $P \vdash \langle e \cdot M(es), s, b \rangle \rightarrow^* \langle \text{THROW } \text{IncompatibleClassChangeError}, s_2, b_2 \rangle \langle \text{proof} \rangle$

### 1.23.2 SCall

**lemma** *SCallRedsParams*:

$P \vdash \langle es, s, b \rangle \mapsto^* \langle es', s', b' \rangle \implies P \vdash \langle C \cdot_s M(es), s, b \rangle \rightarrow^* \langle C \cdot_s M(es'), s', b' \rangle \langle \text{proof} \rangle$

**lemma** *SCallRedsFinal*:

**assumes**  $wuf: wuf\text{-}J\text{-}prog\ P$   
**and**  $P \vdash \langle es, s_0, b_0 \rangle [\rightarrow]^* \langle map\ Val\ vs, (h_2, l_2, sh_2), b_2 \rangle$   
 $P \vdash C\ sees\ M, Static: Ts \rightarrow T = (pns, body)\ in\ D$   
 $sh_2\ D = Some(sfs, Done) \vee (M = clinit \wedge sh_2\ D = [(sfs, Processing)])$   
 $size\ vs = size\ pns$   
**and**  $l_2': l_2' = [pns[\mapsto]vs]$   
**and**  $body: P \vdash \langle body, (h_2, l_2', sh_2), False \rangle \rightarrow^* \langle ef, (h_3, l_3, sh_3), b_3 \rangle$   
**and** *final*  $ef$   
**shows**  $P \vdash \langle C_s M(es), s_0, b_0 \rangle \rightarrow^* \langle ef, (h_3, l_2, sh_3), b_3 \rangle \langle proof \rangle$   
**lemma** *SCallRedsThrowParams*:  
 $\llbracket P \vdash \langle es, s_0, b_0 \rangle [\rightarrow]^* \langle map\ Val\ vs_1\ @\ throw\ a\ \# \ es_2, s_2, b_2 \rangle \rrbracket$   
 $\implies P \vdash \langle C_s M(es), s_0, b_0 \rangle \rightarrow^* \langle throw\ a, s_2, b_2 \rangle \langle proof \rangle$   
**lemma** *SCallRedsNone*:  
 $\llbracket P \vdash \langle es, s, b \rangle [\rightarrow]^* \langle map\ Val\ vs, s_2, False \rangle;$   
 $\neg(\exists b\ Ts\ T\ m\ D. P \vdash C\ sees\ M, b: Ts \rightarrow T = m\ in\ D) \rrbracket$   
 $\implies P \vdash \langle C_s M(es), s, b \rangle \rightarrow^* \langle THROW\ NoSuchMethodError, s_2, False \rangle \langle proof \rangle$   
**lemma** *SCallRedsNonStatic*:  
 $\llbracket P \vdash \langle es, s, b \rangle [\rightarrow]^* \langle map\ Val\ vs, s_2, False \rangle;$   
 $P \vdash C\ sees\ M, NonStatic: Ts \rightarrow T = m\ in\ D \rrbracket$   
 $\implies P \vdash \langle C_s M(es), s, b \rangle \rightarrow^* \langle THROW\ IncompatibleClassChangeError, s_2, False \rangle \langle proof \rangle$   
**lemma** *SCallInitThrowReds*:  
**assumes**  $wuf: wuf\text{-}J\text{-}prog\ P$   
**and**  $P \vdash \langle es, s_0, b_0 \rangle [\rightarrow]^* \langle map\ Val\ vs, (h_1, l_1, sh_1), False \rangle$   
 $P \vdash C\ sees\ M, Static: Ts \rightarrow T = (pns, body)\ in\ D$   
 $\nexists sfs. sh_1\ D = Some(sfs, Done)$   
 $M \neq clinit$   
**and**  $P \vdash \langle INIT\ D\ ([D], False) \leftarrow\ unit, (h_1, l_1, sh_1), False \rangle \rightarrow^* \langle throw\ a, (h_2, l_2, sh_2), b_2 \rangle$   
**shows**  $P \vdash \langle C_s M(es), s_0, b_0 \rangle \rightarrow^* \langle throw\ a, (h_2, l_2, sh_2), b_2 \rangle \langle proof \rangle$   
**lemma** *SCallInitReds*:  
**assumes**  $wuf: wuf\text{-}J\text{-}prog\ P$   
**and**  $P \vdash \langle es, s_0, b_0 \rangle [\rightarrow]^* \langle map\ Val\ vs, (h_1, l_1, sh_1), False \rangle$   
 $P \vdash C\ sees\ M, Static: Ts \rightarrow T = (pns, body)\ in\ D$   
 $\nexists sfs. sh_1\ D = Some(sfs, Done)$   
 $M \neq clinit$   
**and**  $P \vdash \langle INIT\ D\ ([D], False) \leftarrow\ unit, (h_1, l_1, sh_1), False \rangle \rightarrow^* \langle Val\ v', (h_2, l_2, sh_2), b_2 \rangle$   
**and**  $size\ vs = size\ pns$   
**and**  $l_2': l_2' = [pns[\mapsto]vs]$   
**and**  $body: P \vdash \langle body, (h_2, l_2', sh_2), False \rangle \rightarrow^* \langle ef, (h_3, l_3, sh_3), b_3 \rangle$   
**and** *final*  $ef$   
**shows**  $P \vdash \langle C_s M(es), s_0, b_0 \rangle \rightarrow^* \langle ef, (h_3, l_2, sh_3), b_3 \rangle \langle proof \rangle$   
**lemma** *SCallInitProcessingReds*:  
**assumes**  $wuf: wuf\text{-}J\text{-}prog\ P$   
**and**  $P \vdash \langle es, s_0, b_0 \rangle [\rightarrow]^* \langle map\ Val\ vs, (h_2, l_2, sh_2), b_2 \rangle$   
 $P \vdash C\ sees\ M, Static: Ts \rightarrow T = (pns, body)\ in\ D$   
 $sh_2\ D = Some(sfs, Processing)$   
**and**  $size\ vs = size\ pns$   
**and**  $l_2': l_2' = [pns[\mapsto]vs]$   
**and**  $body: P \vdash \langle body, (h_2, l_2', sh_2), False \rangle \rightarrow^* \langle ef, (h_3, l_3, sh_3), b_3 \rangle$   
**and** *final*  $ef$   
**shows**  $P \vdash \langle C_s M(es), s_0, b_0 \rangle \rightarrow^* \langle ef, (h_3, l_2, sh_3), b_3 \rangle \langle proof \rangle$

## The main Theorem

**lemma** *assumes*  $wuf: wuf\text{-}J\text{-}prog\ P$



**shows** *big-by-small*:  $P \vdash \langle e, s \rangle \Rightarrow \langle e', s' \rangle$   
 $\implies (\bigwedge b. \text{iconf } (\text{shp } s) e \implies P, \text{shp } s \vdash_b (e, b) \checkmark \implies P \vdash \langle e, s, b \rangle \rightarrow^* \langle e', s', \text{False} \rangle)$   
**and** *big-by-small*s:  $P \vdash \langle es, s \rangle [\Rightarrow] \langle es', s' \rangle$   
 $\implies (\bigwedge b. \text{iconfs } (\text{shp } s) es \implies P, \text{shp } s \vdash_b (es, b) \checkmark \implies P \vdash \langle es, s, b \rangle [\rightarrow]^* \langle es', s', \text{False} \rangle) \langle \text{proof} \rangle$

### 1.23.3 Big steps simulates small step

This direction was carried out by Norbert Schirmer and Daniel Wasserrab (and modified to include statics and DCI by Susannah Mansky).

The big step equivalent of *RedWhile*:

**lemma** *unfold-while*:

$$P \vdash \langle \text{while}(b) c, s \rangle \Rightarrow \langle e', s' \rangle = P \vdash \langle \text{if}(b) (c;; \text{while}(b) c) \text{ else } (\text{unit}), s \rangle \Rightarrow \langle e', s' \rangle \langle \text{proof} \rangle$$

**lemma** *blocksEval*:

$$\bigwedge Ts \text{ vs } l \ l'. \llbracket \text{size } ps = \text{size } Ts; \text{ size } ps = \text{size } vs; P \vdash \langle \text{blocks}(ps, Ts, vs, e), (h, l, sh) \rangle \Rightarrow \langle e', (h', l', sh') \rangle \rrbracket \\ \implies \exists l''. P \vdash \langle e, (h, l(ps \mapsto vs), sh) \rangle \Rightarrow \langle e', (h', l'', sh') \rangle \langle \text{proof} \rangle$$

**lemma**

**assumes** *wf*: *wuf-J-prog*  $P$

**shows** *eval-restrict-lcl*:

$$P \vdash \langle e, (h, l, sh) \rangle \Rightarrow \langle e', (h', l', sh') \rangle \implies (\bigwedge W. \text{fv } e \subseteq W \implies P \vdash \langle e, (h, l |' W, sh) \rangle \Rightarrow \langle e', (h', l' |' W, sh') \rangle) \\ \text{and } P \vdash \langle es, (h, l, sh) \rangle [\Rightarrow] \langle es', (h', l', sh') \rangle \implies (\bigwedge W. \text{fvs } es \subseteq W \implies P \vdash \langle es, (h, l |' W, sh) \rangle [\Rightarrow] \\ \langle es', (h', l' |' W, sh') \rangle) \langle \text{proof} \rangle$$

**lemma** *eval-notfree-unchanged*:

$$P \vdash \langle e, (h, l, sh) \rangle \Rightarrow \langle e', (h', l', sh') \rangle \implies (\bigwedge V. V \notin \text{fv } e \implies l' V = l V) \\ \text{and } P \vdash \langle es, (h, l, sh) \rangle [\Rightarrow] \langle es', (h', l', sh') \rangle \implies (\bigwedge V. V \notin \text{fvs } es \implies l' V = l V) \langle \text{proof} \rangle$$

**lemma** *eval-closed-lcl-unchanged*:

$$\llbracket P \vdash \langle e, (h, l, sh) \rangle \Rightarrow \langle e', (h', l', sh') \rangle; \text{fv } e = \{\} \rrbracket \implies l' = l \langle \text{proof} \rangle$$

**lemma** *list-eval-Throw*:

$$\text{assumes } \text{eval-e}: P \vdash \langle \text{throw } x, s \rangle \Rightarrow \langle e', s' \rangle \\ \text{shows } P \vdash \langle \text{map } \text{Val } vs \ @ \ \text{throw } x \ \# \ es', s \rangle [\Rightarrow] \langle \text{map } \text{Val } vs \ @ \ e' \ \# \ es', s' \rangle \langle \text{proof} \rangle$$

**lemma** *seq-ext*:

$$\text{assumes } \text{IH}: \bigwedge e' s'. P \vdash \langle e'', s'' \rangle \Rightarrow \langle e', s' \rangle \implies P \vdash \langle e, s \rangle \Rightarrow \langle e', s' \rangle \\ \text{and } \text{seq}: P \vdash \langle e'' ;; e_0, s'' \rangle \Rightarrow \langle e', s' \rangle \\ \text{shows } P \vdash \langle e ;; e_0, s \rangle \Rightarrow \langle e', s' \rangle \\ \langle \text{proof} \rangle$$

**lemma** *rinit-Val-ext*:

$$\text{assumes } \text{ri}: P \vdash \langle \text{RI } (C, e''); Cs \leftarrow e_0, s'' \rangle \Rightarrow \langle \text{Val } v', s_1 \rangle \\ \text{and } \text{IH}: \bigwedge e' s'. P \vdash \langle e'', s'' \rangle \Rightarrow \langle e', s' \rangle \implies P \vdash \langle e, s \rangle \Rightarrow \langle e', s' \rangle \\ \text{shows } P \vdash \langle \text{RI } (C, e); Cs \leftarrow e_0, s \rangle \Rightarrow \langle \text{Val } v', s_1 \rangle \\ \langle \text{proof} \rangle$$

**lemma** *rinit-throw-ext*:

$$\text{assumes } \text{ri}: P \vdash \langle \text{RI } (C, e''); Cs \leftarrow e_0, s'' \rangle \Rightarrow \langle \text{throw } e_t, s' \rangle \\ \text{and } \text{IH}: \bigwedge e' s'. P \vdash \langle e'', s'' \rangle \Rightarrow \langle e', s' \rangle \implies P \vdash \langle e, s \rangle \Rightarrow \langle e', s' \rangle \\ \text{shows } P \vdash \langle \text{RI } (C, e); Cs \leftarrow e_0, s \rangle \Rightarrow \langle \text{throw } e_t, s' \rangle \\ \langle \text{proof} \rangle$$

**lemma** *rinit-ext*:

$$\text{assumes } \text{IH}: \bigwedge e' s'. P \vdash \langle e'', s'' \rangle \Rightarrow \langle e', s' \rangle \implies P \vdash \langle e, s \rangle \Rightarrow \langle e', s' \rangle \\ \text{shows } \bigwedge e' s'. P \vdash \langle \text{RI } (C, e''); Cs \leftarrow e_0, s'' \rangle \Rightarrow \langle e', s' \rangle \\ \implies P \vdash \langle \text{RI } (C, e); Cs \leftarrow e_0, s \rangle \Rightarrow \langle e', s' \rangle \\ \langle \text{proof} \rangle$$

**lemma****shows** *eval-init-return*:  $P \vdash \langle e, s \rangle \Rightarrow \langle e', s' \rangle$  $\Rightarrow \text{iconf } (\text{shp } s) e$  $\Rightarrow (\exists Cs b. e = \text{INIT } C' (Cs, b) \leftarrow \text{unit}) \vee (\exists C e_0 Cs e_i. e = \text{RI}(C, e_0); Cs @ [C'] \leftarrow \text{unit})$   
 $\vee (\exists e_0. e = \text{RI}(C', e_0); \text{Nil} \leftarrow \text{unit})$  $\Rightarrow (\text{val-of } e' = \text{Some } v \longrightarrow (\exists sfs i. \text{shp } s' C' = [(sfs, i)] \wedge (i = \text{Done} \vee i = \text{Processing})))$   
 $\wedge (\text{throw-of } e' = \text{Some } a \longrightarrow (\exists sfs i. \text{shp } s' C' = [(sfs, \text{Error})]))$ **and**  $P \vdash \langle es, s \rangle [\Rightarrow] \langle es', s' \rangle \Longrightarrow \text{True}$ *<proof>***lemma** *init-Val-PD*:  $P \vdash \langle \text{INIT } C' (Cs, b) \leftarrow \text{unit}, s \rangle \Rightarrow \langle \text{Val } v, s' \rangle$  $\Rightarrow \text{iconf } (\text{shp } s) (\text{INIT } C' (Cs, b) \leftarrow \text{unit})$  $\Rightarrow \exists sfs i. \text{shp } s' C' = [(sfs, i)] \wedge (i = \text{Done} \vee i = \text{Processing})$ *<proof>***lemma** *init-throw-PD*:  $P \vdash \langle \text{INIT } C' (Cs, b) \leftarrow \text{unit}, s \rangle \Rightarrow \langle \text{throw } a, s' \rangle$  $\Rightarrow \text{iconf } (\text{shp } s) (\text{INIT } C' (Cs, b) \leftarrow \text{unit})$  $\Rightarrow \exists sfs i. \text{shp } s' C' = [(sfs, \text{Error})]$ *<proof>***lemma** *rinit-Val-PD*:  $P \vdash \langle \text{RI}(C, e_0); Cs \leftarrow \text{unit}, s \rangle \Rightarrow \langle \text{Val } v, s' \rangle$  $\Rightarrow \text{iconf } (\text{shp } s) (\text{RI}(C, e_0); Cs \leftarrow \text{unit}) \Longrightarrow \text{last}(C \# Cs) = C'$  $\Rightarrow \exists sfs i. \text{shp } s' C' = [(sfs, i)] \wedge (i = \text{Done} \vee i = \text{Processing})$ *<proof>***lemma** *rinit-throw-PD*:  $P \vdash \langle \text{RI}(C, e_0); Cs \leftarrow \text{unit}, s \rangle \Rightarrow \langle \text{throw } a, s' \rangle$  $\Rightarrow \text{iconf } (\text{shp } s) (\text{RI}(C, e_0); Cs \leftarrow \text{unit}) \Longrightarrow \text{last}(C \# Cs) = C'$  $\Rightarrow \exists sfs i. \text{shp } s' C' = [(sfs, \text{Error})]$ *<proof>***lemma** *eval-init-seq'*:  $P \vdash \langle e, s \rangle \Rightarrow \langle e', s' \rangle$  $\Rightarrow (\exists C Cs b e_i. e = \text{INIT } C (Cs, b) \leftarrow e_i) \vee (\exists C e_0 Cs e_i. e = \text{RI}(C, e_0); Cs \leftarrow e_i)$  $\Rightarrow (\exists C Cs b e_i. e = \text{INIT } C (Cs, b) \leftarrow e_i \wedge P \vdash \langle (\text{INIT } C (Cs, b) \leftarrow \text{unit});; e_i, s \rangle \Rightarrow \langle e', s' \rangle)$   
 $\vee (\exists C e_0 Cs e_i. e = \text{RI}(C, e_0); Cs \leftarrow e_i \wedge P \vdash \langle (\text{RI}(C, e_0); Cs \leftarrow \text{unit});; e_i, s \rangle \Rightarrow \langle e', s' \rangle)$ **and**  $P \vdash \langle es, s \rangle [\Rightarrow] \langle es', s' \rangle \Longrightarrow \text{True}$ *<proof>***lemma** *eval-init-seq*:  $P \vdash \langle \text{INIT } C (Cs, b) \leftarrow e, (h, l, sh) \rangle \Rightarrow \langle e', s' \rangle$  $\Rightarrow P \vdash \langle (\text{INIT } C (Cs, b) \leftarrow \text{unit});; e, (h, l, sh) \rangle \Rightarrow \langle e', s' \rangle$ *<proof>*

The key lemma:

**lemma****assumes** *wf*: *wf-J-prog*  $P$ **shows** *extend-1-eval*:  $P \vdash \langle e, s, b \rangle \rightarrow \langle e'', s'', b'' \rangle \Longrightarrow P, \text{shp } s \vdash_b (e, b) \checkmark$  $\Longrightarrow (\bigwedge s' e'. P \vdash \langle e'', s'' \rangle \Rightarrow \langle e', s' \rangle \Longrightarrow P \vdash \langle e, s \rangle \Rightarrow \langle e', s' \rangle)$ **and** *extend-1-evals*:  $P \vdash \langle es, s, b \rangle [\rightarrow] \langle es'', s'', b'' \rangle \Longrightarrow P, \text{shp } s \vdash_b (es, b) \checkmark$  $\Longrightarrow (\bigwedge s' es'. P \vdash \langle es'', s'' \rangle [\Rightarrow] \langle es', s' \rangle \Longrightarrow P \vdash \langle es, s \rangle [\Rightarrow] \langle es', s' \rangle)$ *<proof>*Its extension to  $\rightarrow^*$ :**lemma** *extend-eval*:**assumes** *wf*: *wf-J-prog*  $P$ **shows**  $\llbracket P \vdash \langle e, s, b \rangle \rightarrow^* \langle e'', s'', b'' \rangle; P \vdash \langle e'', s'' \rangle \Rightarrow \langle e', s' \rangle;$  $\text{iconf } (\text{shp } s) e; P, \text{shp } s \vdash_b (e::\text{expr}, b) \checkmark \rrbracket$

$$\Longrightarrow P \vdash \langle e, s \rangle \Rightarrow \langle e', s' \rangle \langle \text{proof} \rangle$$

**lemma** *extend-vals*:

**assumes** *wf*: *wf-J-prog P*

**shows**  $\llbracket P \vdash \langle es, s, b \rangle \rightarrow^* \langle es'', s'', b'' \rangle; P \vdash \langle es', s' \rangle \Rightarrow \langle es', s' \rangle;$   
 $\text{iconfs } (\text{shp } s) \text{ } es; P, \text{shp } s \vdash_b (es::\text{expr list}, b) \checkmark \rrbracket$

$$\Longrightarrow P \vdash \langle es, s \rangle \Rightarrow \langle es', s' \rangle \langle \text{proof} \rangle$$

Finally, small step semantics can be simulated by big step semantics:

**theorem**

**assumes** *wf*: *wf-J-prog P*

**shows** *small-by-big*:

$\llbracket P \vdash \langle e, s, b \rangle \rightarrow^* \langle e', s', b' \rangle; \text{iconf } (\text{shp } s) \text{ } e; P, \text{shp } s \vdash_b (e, b) \checkmark; \text{final } e' \rrbracket$

$$\Longrightarrow P \vdash \langle e, s \rangle \Rightarrow \langle e', s' \rangle$$

**and**  $\llbracket P \vdash \langle es, s, b \rangle \rightarrow^* \langle es', s', b' \rangle; \text{iconfs } (\text{shp } s) \text{ } es; P, \text{shp } s \vdash_b (es, b) \checkmark; \text{finals } es' \rrbracket$

$$\Longrightarrow P \vdash \langle es, s \rangle \Rightarrow \langle es', s' \rangle \langle \text{proof} \rangle$$

### 1.23.4 Equivalence

And now, the crowning achievement:

**corollary** *big-iff-small*:

$\llbracket \text{wf-J-prog } P; \text{iconf } (\text{shp } s) \text{ } e; P, \text{shp } s \vdash_b (e::\text{expr}, b) \checkmark \rrbracket$

$$\Longrightarrow P \vdash \langle e, s \rangle \Rightarrow \langle e', s' \rangle = (P \vdash \langle e, s, b \rangle \rightarrow^* \langle e', s', \text{False} \rangle \wedge \text{final } e') \langle \text{proof} \rangle$$

**corollary** *big-iff-small-WT*:

$$\text{wf-J-prog } P \Longrightarrow P, E \vdash e::T \Longrightarrow P, \text{shp } s \vdash_b (e, b) \checkmark \Longrightarrow$$

$$P \vdash \langle e, s \rangle \Rightarrow \langle e', s' \rangle = (P \vdash \langle e, s, b \rangle \rightarrow^* \langle e', s', \text{False} \rangle \wedge \text{final } e') \langle \text{proof} \rangle$$

### 1.23.5 Lifting type safety to $\Rightarrow$

... and now to the big step semantics, just for fun.

**lemma** *eval-preserves-sconf*:

**fixes** *s::state* **and** *s'::state*

**assumes** *wf-J-prog P* **and**  $P \vdash \langle e, s \rangle \Rightarrow \langle e', s' \rangle$  **and**  $\text{iconf } (\text{shp } s) \text{ } e$

**and**  $P, E \vdash e::T$  **and**  $P, E \vdash s \checkmark$

**shows**  $P, E \vdash s' \checkmark \langle \text{proof} \rangle$

**lemma** *eval-preserves-type*:

**fixes** *s::state*

**assumes** *wf*: *wf-J-prog P*

**and**  $P \vdash \langle e, s \rangle \Rightarrow \langle e', s' \rangle$  **and**  $P, E \vdash s \checkmark$  **and**  $\text{iconf } (\text{shp } s) \text{ } e$  **and**  $P, E \vdash e::T$

**shows**  $\exists T'. P \vdash T' \leq T \wedge P, E, \text{shp } s', \text{shp } s' \vdash e':T' \langle \text{proof} \rangle$

end

## 1.24 Program annotation

**theory** *Annotate* **imports** *WellType* **begin**

**inductive**

*Anno* ::  $[J\text{-prog}, \text{env}, \text{expr} \quad , \text{expr}] \Rightarrow \text{bool}$

$(\langle -, - \vdash - \rightsquigarrow - \rangle [51, 0, 0, 51] 50)$

**and** *Annos* ::  $[J\text{-prog}, \text{env}, \text{expr list}, \text{expr list}] \Rightarrow \text{bool}$

$(\langle -, - \vdash - [\rightsquigarrow] - \rangle [51, 0, 0, 51] 50)$

for  $P :: J\text{-prog}$   
 where

$AnnoNew: P, E \vdash \text{new } C \rightsquigarrow \text{new } C$   
 $| AnnoCast: P, E \vdash e \rightsquigarrow e' \implies P, E \vdash \text{Cast } C \ e \rightsquigarrow \text{Cast } C \ e'$   
 $| AnnoVal: P, E \vdash \text{Val } v \rightsquigarrow \text{Val } v$   
 $| AnnoVarVar: E \ V = \lfloor T \rfloor \implies P, E \vdash \text{Var } V \rightsquigarrow \text{Var } V$   
 $| AnnoVarField: \llbracket E \ V = \text{None}; E \ \text{this} = \lfloor \text{Class } C \rfloor; P \vdash C \ \text{sees } V, \text{NonStatic}: T \ \text{in } D \rrbracket$   
 $\implies P, E \vdash \text{Var } V \rightsquigarrow \text{Var } \text{this} \cdot V \{D\}$   
 $| AnnoBinOp:$   
 $\llbracket P, E \vdash e1 \rightsquigarrow e1'; P, E \vdash e2 \rightsquigarrow e2' \rrbracket$   
 $\implies P, E \vdash e1 \ \langle\langle \text{bop} \rangle\rangle \ e2 \rightsquigarrow e1' \ \langle\langle \text{bop} \rangle\rangle \ e2'$   
 $| AnnoLAssVar:$   
 $\llbracket E \ V = \lfloor T \rfloor; P, E \vdash e \rightsquigarrow e' \rrbracket \implies P, E \vdash V := e \rightsquigarrow V := e'$   
 $| AnnoLAssField:$   
 $\llbracket E \ V = \text{None}; E \ \text{this} = \lfloor \text{Class } C \rfloor; P \vdash C \ \text{sees } V, \text{NonStatic}: T \ \text{in } D; P, E \vdash e \rightsquigarrow e' \rrbracket$   
 $\implies P, E \vdash V := e \rightsquigarrow \text{Var } \text{this} \cdot V \{D\} := e'$   
 $| AnnoFAcc:$   
 $\llbracket P, E \vdash e \rightsquigarrow e'; P, E \vdash e' :: \text{Class } C; P \vdash C \ \text{sees } F, \text{NonStatic}: T \ \text{in } D \rrbracket$   
 $\implies P, E \vdash e \cdot F \{\} \rightsquigarrow e' \cdot F \{D\}$   
 $| AnnoSFAcc:$   
 $\llbracket P \vdash C \ \text{sees } F, \text{Static}: T \ \text{in } D \rrbracket$   
 $\implies P, E \vdash C \cdot_s F \{\} \rightsquigarrow C \cdot_s F \{D\}$   
 $| AnnoFAss: \llbracket P, E \vdash e1 \rightsquigarrow e1'; P, E \vdash e2 \rightsquigarrow e2';$   
 $P, E \vdash e1' :: \text{Class } C; P \vdash C \ \text{sees } F, \text{NonStatic}: T \ \text{in } D \rrbracket$   
 $\implies P, E \vdash e1 \cdot F \{\} := e2 \rightsquigarrow e1' \cdot F \{D\} := e2'$   
 $| AnnoSFAss: \llbracket P, E \vdash e2 \rightsquigarrow e2'; P \vdash C \ \text{sees } F, \text{Static}: T \ \text{in } D \rrbracket$   
 $\implies P, E \vdash C \cdot_s F \{\} := e2 \rightsquigarrow C \cdot_s F \{D\} := e2'$   
 $| AnnoCall:$   
 $\llbracket P, E \vdash e \rightsquigarrow e'; P, E \vdash es \ [\rightsquigarrow] \ es' \rrbracket$   
 $\implies P, E \vdash \text{Call } e \ M \ es \rightsquigarrow \text{Call } e' \ M \ es'$   
 $| AnnoSCall:$   
 $\llbracket P, E \vdash es \ [\rightsquigarrow] \ es' \rrbracket$   
 $\implies P, E \vdash \text{SCall } C \ M \ es \rightsquigarrow \text{SCall } C \ M \ es'$   
 $| AnnoBlock:$   
 $P, E(V \mapsto T) \vdash e \rightsquigarrow e' \implies P, E \vdash \{V:T; e\} \rightsquigarrow \{V:T; e'\}$   
 $| AnnoComp: \llbracket P, E \vdash e1 \rightsquigarrow e1'; P, E \vdash e2 \rightsquigarrow e2' \rrbracket$   
 $\implies P, E \vdash e1 ;; e2 \rightsquigarrow e1' ;; e2'$   
 $| AnnoCond: \llbracket P, E \vdash e \rightsquigarrow e'; P, E \vdash e1 \rightsquigarrow e1'; P, E \vdash e2 \rightsquigarrow e2' \rrbracket$   
 $\implies P, E \vdash \text{if } (e) \ e1 \ \text{else } e2 \rightsquigarrow \text{if } (e') \ e1' \ \text{else } e2'$   
 $| AnnoLoop: \llbracket P, E \vdash e \rightsquigarrow e'; P, E \vdash c \rightsquigarrow c' \rrbracket$   
 $\implies P, E \vdash \text{while } (e) \ c \rightsquigarrow \text{while } (e') \ c'$   
 $| AnnoThrow: P, E \vdash e \rightsquigarrow e' \implies P, E \vdash \text{throw } e \rightsquigarrow \text{throw } e'$   
 $| AnnoTry: \llbracket P, E \vdash e1 \rightsquigarrow e1'; P, E(V \mapsto \text{Class } C) \vdash e2 \rightsquigarrow e2' \rrbracket$   
 $\implies P, E \vdash \text{try } e1 \ \text{catch}(C \ V) \ e2 \rightsquigarrow \text{try } e1' \ \text{catch}(C \ V) \ e2'$   
  
 $| AnnoNil: P, E \vdash [] \ [\rightsquigarrow] \ []$   
 $| AnnoCons: \llbracket P, E \vdash e \rightsquigarrow e'; P, E \vdash es \ [\rightsquigarrow] \ es' \rrbracket$   
 $\implies P, E \vdash e \# es \ [\rightsquigarrow] \ e' \# es'$

end

# Chapter 2

## Jinja Virtual Machine

### 2.1 State of the JVM

**theory** *JVMState* **imports** *../Common/Objects* **begin**

**type-synonym**

*pc* = *nat*

**abbreviation** *start-sheap* :: *sheap*

**where** *start-sheap* ≡ (λ*x*. *None*)(*Start* ↦ (*Map.empty*,*Done*))

**definition** *start-sheap-preloaded* :: '*m prog* ⇒ *sheap*

**where**

*start-sheap-preloaded* *P* ≡ *fold* (λ(*C*,*cl*) *f*. *f*(*C* := *Some* (*sblank* *P C*, *Prepared*))) *P* (λ*x*. *None*)

#### 2.1.1 Frame Stack

**datatype** *init-call-status* = *No-ics* | *Calling cname cname list*  
| *Called cname list* | *Throwing cname list addr*

- *No-ics* = not currently calling or waiting for the result of an initialization procedure call
- *Calling C Cs* = current instruction is calling for initialization of classes *C#Cs* (last class is the original) – still collecting classes to be initialized, *C* most recently collected
- *Called Cs* = current instruction called initialization and is waiting for the result – now initializing classes in the list
- *Throwing Cs a* = frame threw or was thrown an error causing erroneous end of initialization procedure for classes *Cs*

**type-synonym**

*frame* = *val list* × *val list* × *cname* × *mname* × *pc* × *init-call-status*

- operand stack
- registers (including this pointer, method parameters, and local variables)
- name of class where current method is defined
- current method
- program counter within frame
- indicates frame's initialization call status

**translations**

(*type*) *frame* <= (*type*) *val list* × *val list* × *char list* × *char list* × *nat* × *init-call-status*

**fun** *curr-stk* :: *frame*  $\Rightarrow$  *val list* **where**  
*curr-stk* (*stk*, *loc*, *C*, *M*, *pc*, *ics*) = *stk*

**fun** *curr-class* :: *frame*  $\Rightarrow$  *cname* **where**  
*curr-class* (*stk*, *loc*, *C*, *M*, *pc*, *ics*) = *C*

**fun** *curr-method* :: *frame*  $\Rightarrow$  *mname* **where**  
*curr-method* (*stk*, *loc*, *C*, *M*, *pc*, *ics*) = *M*

**fun** *curr-pc* :: *frame*  $\Rightarrow$  *nat* **where**  
*curr-pc* (*stk*, *loc*, *C*, *M*, *pc*, *ics*) = *pc*

**fun** *init-status* :: *frame*  $\Rightarrow$  *init-call-status* **where**  
*init-status* (*stk*, *loc*, *C*, *M*, *pc*, *ics*) = *ics*

**fun** *ics-of* :: *frame*  $\Rightarrow$  *init-call-status* **where**  
*ics-of* *fr* = *snd*(*snd*(*snd*(*snd*(*snd* *fr*))))

## 2.1.2 Runtime State

**type-synonym**

*jvm-state* = *addr option*  $\times$  *heap*  $\times$  *frame list*  $\times$  *sheap*  
— exception flag, heap, frames, static heap

**translations**

(*type*) *jvm-state*  $\leq$  (*type*) *nat option*  $\times$  *heap*  $\times$  *frame list*  $\times$  *sheap*

**fun** *frames-of* :: *jvm-state*  $\Rightarrow$  *frame list* **where**  
*frames-of* (*xp*, *h*, *frs*, *sh*) = *frs*

**abbreviation** *sheap* :: *jvm-state*  $\Rightarrow$  *sheap* **where**  
*sheap js*  $\equiv$  *snd* (*snd* (*snd* *js*))

**end**

## 2.2 Instructions of the JVM

**theory** *JVMInstructions* **imports** *JVMState* **begin**

**datatype**

*instr* = *Load nat* — load from local variable  
| *Store nat* — store into local variable  
| *Push val* — push a value (constant)  
| *New cname* — create object  
| *Getfield vname cname* — Fetch field from object  
| *Getstatic cname vname cname* — Fetch static field from class  
| *Putfield vname cname* — Set field in object  
| *Putstatic cname vname cname* — Set static field in class  
| *Checkcast cname* — Check whether object is of given type  
| *Invoke mname nat* — inv. instance meth of an object  
| *Invokestatic cname mname nat* — inv. static method of a class  
| *Return* — return from method  
| *Pop* — pop top element from opstack

<i>IAdd</i>	— integer addition
<i>Goto int</i>	— goto relative address
<i>CmpEq</i>	— equality comparison
<i>IfFalse int</i>	— branch if top of stack false
<i>Throw</i>	— throw top of stack as exception

**type-synonym**

*bytecode* = *instr list*

**type-synonym**

*ex-entry* = *pc* × *pc* × *cname* × *pc* × *nat*

— start-pc, end-pc, exception type, handler-pc, remaining stack depth

**type-synonym**

*ex-table* = *ex-entry list*

**type-synonym**

*jvm-method* = *nat* × *nat* × *bytecode* × *ex-table*

— max stacksize

— number of local variables. Add 1 + no. of parameters to get no. of registers

— instruction sequence

— exception handler table

**type-synonym**

*jvm-prog* = *jvm-method prog*

end

## 2.3 Exception handling in the JVM

**theory** *JVMExceptions* **imports** *../Common/Exceptions JVMInstructions*  
**begin**

**definition** *matches-ex-entry* :: *'m prog* ⇒ *cname* ⇒ *pc* ⇒ *ex-entry* ⇒ *bool*

**where**

*matches-ex-entry* *P C pc xcp* ≡  
 $\text{let } (s, e, C', h, d) = xcp \text{ in}$   
 $s \leq pc \wedge pc < e \wedge P \vdash C \preceq^* C'$

**primrec** *match-ex-table* :: *'m prog* ⇒ *cname* ⇒ *pc* ⇒ *ex-table* ⇒ (*pc* × *nat*) *option*

**where**

*match-ex-table* *P C pc []* = *None*  
| *match-ex-table* *P C pc (e#es)* = (if *matches-ex-entry* *P C pc e*  
then *Some (snd(snd(snd e)))*  
else *match-ex-table* *P C pc es*)

**abbreviation**

*ex-table-of* :: *jvm-prog* ⇒ *cname* ⇒ *mname* ⇒ *ex-table* **where**

*ex-table-of* *P C M* == *snd (snd (snd (snd (snd (snd (method P C M))))))*)

**fun** *find-handler* :: *jvm-prog* ⇒ *addr* ⇒ *heap* ⇒ *frame list* ⇒ *sheap* ⇒ *jvm-state*





**lemma** *find-handler-frs-tl-neq*:

*ics-of f*  $\neq$  *No-ics*  
 $\implies (xp, h, f\#frs, sh) \neq \text{find-handler } P \text{ xa } h' (f' \# frs) sh'$   
 <proof>

**end**

## 2.4 Program Execution in the JVM

**theory** *JVMExecInstr*

**imports** *JVMInstructions JVMExceptions*

**begin**

— frame calling the class initialization method for the given class in the given program

**fun** *create-init-frame* :: [*jvm-prog*, *cname*]  $\Rightarrow$  *frame* **where**

*create-init-frame P C* =  
 (let (*D*,*b*,*Ts*,*T*,(*mxs*,*m<sub>xl0</sub>*,*ins*,*xt*)) = *method P C clinit*  
 in ( $\square$ , (*replicate m<sub>xl0</sub> undefined*), *D*, *clinit*, 0, *No-ics*)  
 )

**primrec** *exec-instr* :: [*instr*, *jvm-prog*, *heap*, *val list*, *val list*,  
*cname*, *mname*, *pc*, *init-call-status*, *frame list*, *sheap*]  $\Rightarrow$  *jvm-state*

**where**

*exec-instr-Load*:  
*exec-instr (Load n) P h stk loc C<sub>0</sub> M<sub>0</sub> pc ics frs sh* =  
 (*None*, *h*, ((*loc ! n*) # *stk*, *loc*, *C<sub>0</sub>*, *M<sub>0</sub>*, *Suc pc*, *ics*)#*frs*, *sh*)

| *exec-instr-Store*:  
*exec-instr (Store n) P h stk loc C<sub>0</sub> M<sub>0</sub> pc ics frs sh* =  
 (*None*, *h*, (*tl stk*, *loc[n:=hd stk]*, *C<sub>0</sub>*, *M<sub>0</sub>*, *Suc pc*, *ics*)#*frs*, *sh*)

| *exec-instr-Push*:  
*exec-instr (Push v) P h stk loc C<sub>0</sub> M<sub>0</sub> pc ics frs sh* =  
 (*None*, *h*, (*v* # *stk*, *loc*, *C<sub>0</sub>*, *M<sub>0</sub>*, *Suc pc*, *ics*)#*frs*, *sh*)

| *exec-instr-New*:  
*exec-instr (New C) P h stk loc C<sub>0</sub> M<sub>0</sub> pc ics frs sh* =  
 (case (*ics*, *sh C*) of  
 (*Called Cs*, -)  $\Rightarrow$   
 (case *new-Addr h* of  
*None*  $\Rightarrow$  ( $\lfloor$ *addr-of-sys-xcpt OutOfMemory* $\rfloor$ , *h*, (*stk*, *loc*, *C<sub>0</sub>*, *M<sub>0</sub>*, *pc*, *No-ics*)#*frs*, *sh*)  
 | *Some a*  $\Rightarrow$  (*None*, *h(a $\rightarrow$ blank P C)*, (*Addr a*#*stk*, *loc*, *C<sub>0</sub>*, *M<sub>0</sub>*, *Suc pc*, *No-ics*)#*frs*, *sh*)  
 )  
 | (*-*, *Some(obj, Done)*)  $\Rightarrow$   
 (case *new-Addr h* of  
*None*  $\Rightarrow$  ( $\lfloor$ *addr-of-sys-xcpt OutOfMemory* $\rfloor$ , *h*, (*stk*, *loc*, *C<sub>0</sub>*, *M<sub>0</sub>*, *pc*, *ics*)#*frs*, *sh*)  
 | *Some a*  $\Rightarrow$  (*None*, *h(a $\rightarrow$ blank P C)*, (*Addr a*#*stk*, *loc*, *C<sub>0</sub>*, *M<sub>0</sub>*, *Suc pc*, *ics*)#*frs*, *sh*)  
 )  
 | -  $\Rightarrow$  (*None*, *h*, (*stk*, *loc*, *C<sub>0</sub>*, *M<sub>0</sub>*, *pc*, *Calling C*  $\square$ )#*frs*, *sh*)  
 )

| *exec-instr-Getfield*:

$exec\text{-instr} (Getfield\ F\ C)\ P\ h\ stk\ loc\ C_0\ M_0\ pc\ ics\ frs\ sh =$   
 $(let\ v = hd\ stk;$   
 $(D,fs) = the(h(the\text{-Addr}\ v));$   
 $(D',b,t) = field\ P\ C\ F;$   
 $xp' = if\ v=Null\ then\ [addr\text{-of}\text{-sys}\text{-xcpt}\ NullPointer]$   
 $\quad else\ if\ \neg(\exists\ t\ b.\ P \vdash D\ has\ F,b:t\ in\ C)$   
 $\quad\quad then\ [addr\text{-of}\text{-sys}\text{-xcpt}\ NoSuchFieldError]$   
 $\quad\quad\quad else\ case\ b\ of\ Static \Rightarrow [addr\text{-of}\text{-sys}\text{-xcpt}\ IncompatibleClassChangeError]$   
 $\quad\quad\quad\quad | NonStatic \Rightarrow None$   
 $in\ case\ xp'\ of\ None \Rightarrow (xp',\ h,\ (the(fs(F,C))\#(tl\ stk),\ loc,\ C_0,\ M_0,\ pc+1,\ ics)\#frs,\ sh)$   
 $\quad | Some\ x \Rightarrow (xp',\ h,\ (stk,\ loc,\ C_0,\ M_0,\ pc,\ ics)\#frs,\ sh))$

|  $exec\text{-instr}\text{-Getstatic}:$

$exec\text{-instr} (Getstatic\ C\ F\ D)\ P\ h\ stk\ loc\ C_0\ M_0\ pc\ ics\ frs\ sh =$   
 $(let\ (D',b,t) = field\ P\ D\ F;$   
 $xp' = if\ \neg(\exists\ t\ b.\ P \vdash C\ has\ F,b:t\ in\ D)$   
 $\quad then\ [addr\text{-of}\text{-sys}\text{-xcpt}\ NoSuchFieldError]$   
 $\quad\quad else\ case\ b\ of\ NonStatic \Rightarrow [addr\text{-of}\text{-sys}\text{-xcpt}\ IncompatibleClassChangeError]$   
 $\quad\quad\quad | Static \Rightarrow None$   
 $in\ (case\ (xp',\ ics,\ sh\ D')\ of$   
 $\quad (Some\ a,\ -) \Rightarrow (xp',\ h,\ (stk,\ loc,\ C_0,\ M_0,\ pc,\ ics)\#frs,\ sh)$   
 $\quad | (-,\ Called\ Cs,\ -) \Rightarrow let\ (sfs,\ i) = the(sh\ D');$   
 $\quad\quad\quad\quad\quad\quad\quad\quad v = the(sfs\ F)$   
 $\quad\quad\quad\quad\quad\quad\quad\quad in\ (xp',\ h,\ (v\#stk,\ loc,\ C_0,\ M_0,\ Suc\ pc,\ No\text{-ics})\#frs,\ sh)$   
 $\quad | (-,\ -, Some\ (sfs,\ Done)) \Rightarrow let\ v = the\ (sfs\ F)$   
 $\quad\quad\quad\quad\quad\quad\quad\quad in\ (xp',\ h,\ (v\#stk,\ loc,\ C_0,\ M_0,\ Suc\ pc,\ ics)\#frs,\ sh)$   
 $\quad | - \Rightarrow (xp',\ h,\ (stk,\ loc,\ C_0,\ M_0,\ pc,\ Calling\ D'\ [])\#frs,\ sh)$   
 $)$   
 $)$

|  $exec\text{-instr}\text{-Putfield}:$

$exec\text{-instr} (Putfield\ F\ C)\ P\ h\ stk\ loc\ C_0\ M_0\ pc\ ics\ frs\ sh =$   
 $(let\ v = hd\ stk;$   
 $r = hd\ (tl\ stk);$   
 $a = the\text{-Addr}\ r;$   
 $(D,fs) = the\ (h\ a);$   
 $(D',b,t) = field\ P\ C\ F;$   
 $xp' = if\ r=Null\ then\ [addr\text{-of}\text{-sys}\text{-xcpt}\ NullPointer]$   
 $\quad else\ if\ \neg(\exists\ t\ b.\ P \vdash D\ has\ F,b:t\ in\ C)$   
 $\quad\quad then\ [addr\text{-of}\text{-sys}\text{-xcpt}\ NoSuchFieldError]$   
 $\quad\quad\quad else\ case\ b\ of\ Static \Rightarrow [addr\text{-of}\text{-sys}\text{-xcpt}\ IncompatibleClassChangeError]$   
 $\quad\quad\quad\quad | NonStatic \Rightarrow None;$   
 $\quad\quad\quad\quad\quad\quad\quad\quad h' = h(a \mapsto (D,\ fs((F,C) \mapsto v)))$   
 $in\ case\ xp'\ of\ None \Rightarrow (xp',\ h',\ (tl\ (tl\ stk),\ loc,\ C_0,\ M_0,\ pc+1,\ ics)\#frs,\ sh)$   
 $\quad | Some\ x \Rightarrow (xp',\ h,\ (stk,\ loc,\ C_0,\ M_0,\ pc,\ ics)\#frs,\ sh)$   
 $)$

|  $exec\text{-instr}\text{-Putstatic}:$

$exec\text{-instr} (Putstatic\ C\ F\ D)\ P\ h\ stk\ loc\ C_0\ M_0\ pc\ ics\ frs\ sh =$   
 $(let\ (D',b,t) = field\ P\ D\ F;$   
 $xp' = if\ \neg(\exists\ t\ b.\ P \vdash C\ has\ F,b:t\ in\ D)$   
 $\quad then\ [addr\text{-of}\text{-sys}\text{-xcpt}\ NoSuchFieldError]$   
 $\quad\quad else\ case\ b\ of\ NonStatic \Rightarrow [addr\text{-of}\text{-sys}\text{-xcpt}\ IncompatibleClassChangeError]$   
 $\quad\quad\quad | Static \Rightarrow None$

```

in (case (xp', ics, sh D') of
  (Some a, -) ⇒ (xp', h, (stk, loc, C0, M0, pc, ics)#frs, sh)
  | (-, Called Cs, -)
  ⇒ let (sfs, i) = the(sh D')
    in (xp', h, (tl stk, loc, C0, M0, Suc pc, No-ics)#frs, sh(D':=Some ((sfs(F ↦ hd stk)), i)))
    | (-, -, Some (sfs, Done))
  ⇒ (xp', h, (tl stk, loc, C0, M0, Suc pc, ics)#frs, sh(D':=Some ((sfs(F ↦ hd stk)), Done)))
  | - ⇒ (xp', h, (stk, loc, C0, M0, pc, Calling D' [])#frs, sh)
)
)

```

| *exec-instr-Checkcast*:

```

exec-instr (Checkcast C) P h stk loc C0 M0 pc ics frs sh =
  (if cast-ok P C h (hd stk)
   then (None, h, (stk, loc, C0, M0, Suc pc, ics)#frs, sh)
   else [addr-of-sys-xcpt ClassCast], h, (stk, loc, C0, M0, pc, ics)#frs, sh)
)

```

| *exec-instr-Invoke*:

```

exec-instr (Invoke M n) P h stk loc C0 M0 pc ics frs sh =
  (let ps = take n stk;
    r = stk!n;
    C = fst(the(h(the-Addr r)));
    (D,b,Ts,T,maxs,mxl0,ins,xt) = method P C M;
    xp' = if r=NULL then [addr-of-sys-xcpt NullPointer]
          else if ¬(∃ Ts T m D b. P ⊢ C sees M,b:Ts → T = m in D)
            then [addr-of-sys-xcpt NoSuchMethodError]
            else case b of Static ⇒ [addr-of-sys-xcpt IncompatibleClassChangeError]
              | NonStatic ⇒ None;
    f' = ([],[r]@(rev ps)@(replicate mxl0 undefined),D,M,0,No-ics)
  in case xp' of None ⇒ (xp', h, f'#(stk, loc, C0, M0, pc, ics)#frs, sh)
    | Some a ⇒ (xp', h, (stk, loc, C0, M0, pc, ics)#frs, sh)
)

```

| *exec-instr-Invokestatic*:

```

exec-instr (Invokestatic C M n) P h stk loc C0 M0 pc ics frs sh =
  (let ps = take n stk;
    (D,b,Ts,T,maxs,mxl0,ins,xt) = method P C M;
    xp' = if ¬(∃ Ts T m D b. P ⊢ C sees M,b:Ts → T = m in D)
          then [addr-of-sys-xcpt NoSuchMethodError]
          else case b of NonStatic ⇒ [addr-of-sys-xcpt IncompatibleClassChangeError]
            | Static ⇒ None;
    f' = ([,(rev ps)@(replicate mxl0 undefined),D,M,0,No-ics)
  in (case (xp', ics, sh D) of
    (Some a, -) ⇒ (xp', h, (stk, loc, C0, M0, pc, ics)#frs, sh)
    | (-, Called Cs, -) ⇒ (xp', h, f'#(stk, loc, C0, M0, pc, No-ics)#frs, sh)
    | (-, -, Some (sfs, Done)) ⇒ (xp', h, f'#(stk, loc, C0, M0, pc, ics)#frs, sh)
    | - ⇒ (xp', h, (stk, loc, C0, M0, pc, Calling D [])#frs, sh)
  )
)

```

| *exec-instr-Return*:

```

exec-instr Return P h stk0 loc0 C0 M0 pc ics frs sh =
  (case frs of

```

$$\begin{aligned}
& \square \Rightarrow \text{let } sh' = (\text{case } M_0 = \text{clinit of True} \Rightarrow sh(C_0 := \text{Some}(\text{fst}(\text{the}(sh\ C_0))), \text{Done})) \\
& \quad \quad \quad | - \Rightarrow sh \\
& \quad \quad \quad ) \\
& \quad \quad \quad \text{in } (None, h, [], sh') \\
& | (stk', loc', C', m', pc', ics') \# frs' \\
& \quad \Rightarrow \text{let } (D, b, Ts, T, (m\ x\ s, m\ x\ l_0, ins, xt)) = \text{method } P\ C_0\ M_0; \\
& \quad \quad \text{offset} = \text{case } b \text{ of NonStatic} \Rightarrow 1 \mid \text{Static} \Rightarrow 0; \\
& \quad \quad (sh'', stk'', pc'') = (\text{case } M_0 = \text{clinit of True} \Rightarrow (sh(C_0 := \text{Some}(\text{fst}(\text{the}(sh\ C_0))), \text{Done})), \\
& \text{stk}', pc') \\
& \quad \quad \quad | - \Rightarrow (sh, (\text{hd } stk_0) \# (\text{drop } (\text{length } Ts + \text{offset})\ stk'), \text{Suc } pc')) \\
& \quad \quad \quad ) \\
& \quad \quad \quad \text{in } (None, h, (stk'', loc', C', m', pc'', ics'') \# frs', sh'') \\
& ) \\
& | \text{exec-instr-Pop:} \\
& \text{exec-instr Pop } P\ h\ stk\ loc\ C_0\ M_0\ pc\ ics\ frs\ sh = (None, h, (\text{tl } stk, loc, C_0, M_0, \text{Suc } pc, ics) \# frs, sh) \\
& | \text{exec-instr-IAdd:} \\
& \text{exec-instr IAdd } P\ h\ stk\ loc\ C_0\ M_0\ pc\ ics\ frs\ sh = \\
& \quad (None, h, (\text{Intg } (\text{the-Intg } (\text{hd } (\text{tl } stk)) + \text{the-Intg } (\text{hd } stk))) \# (\text{tl } (\text{tl } stk)), loc, C_0, M_0, \text{Suc } pc, \\
& \text{ics}) \# frs, sh) \\
& | \text{exec-instr-IfFalse:} \\
& \text{exec-instr (IfFalse } i) P\ h\ stk\ loc\ C_0\ M_0\ pc\ ics\ frs\ sh = \\
& \quad (\text{let } pc' = \text{if } \text{hd } stk = \text{Bool False} \text{ then } \text{nat}(\text{int } pc + i) \text{ else } pc + 1 \\
& \quad \text{in } (None, h, (\text{tl } stk, loc, C_0, M_0, pc', ics) \# frs, sh)) \\
& | \text{exec-instr-CmpEq:} \\
& \text{exec-instr CmpEq } P\ h\ stk\ loc\ C_0\ M_0\ pc\ ics\ frs\ sh = \\
& \quad (None, h, (\text{Bool } (\text{hd } (\text{tl } stk) = \text{hd } stk) \# \text{tl } (\text{tl } stk), loc, C_0, M_0, \text{Suc } pc, ics) \# frs, sh) \\
& | \text{exec-instr-Goto:} \\
& \text{exec-instr (Goto } i) P\ h\ stk\ loc\ C_0\ M_0\ pc\ ics\ frs\ sh = \\
& \quad (None, h, (stk, loc, C_0, M_0, \text{nat}(\text{int } pc + i), ics) \# frs, sh) \\
& | \text{exec-instr-Throw:} \\
& \text{exec-instr Throw } P\ h\ stk\ loc\ C_0\ M_0\ pc\ ics\ frs\ sh = \\
& \quad (\text{let } xp' = \text{if } \text{hd } stk = \text{Null} \text{ then } \lfloor \text{addr-of-sys-xcpt NullPointer} \rfloor \\
& \quad \quad \text{else } \lfloor \text{the-Addr}(\text{hd } stk) \rfloor \\
& \quad \text{in } (xp', h, (stk, loc, C_0, M_0, pc, ics) \# frs, sh))
\end{aligned}$$

Given a preallocated heap, a thrown exception is either a system exception or thrown directly by *Throw*.

**lemma** *exec-instr-xcpts*:

**assumes**  $\sigma' = \text{exec-instr } i\ P\ h\ stk\ loc\ C\ M\ pc\ ics'\ frs\ sh$

**and**  $\text{fst } \sigma' = \text{Some } a$

**shows**  $i = (\text{JVMInstructions.Throw}) \vee a \in \{a. \exists x \in \text{sys-xcpts}. a = \text{addr-of-sys-xcpt } x\}$

*<proof>*

**lemma** *exec-instr-prealloc-pres*:

**assumes** *preallocated*  $h$

**and**  $\text{exec-instr } i\ P\ h\ stk\ loc\ C_0\ M_0\ pc\ ics\ frs\ sh = (xp', h', frs', sh')$

**shows** *preallocated*  $h'$

*<proof>*

end

## 2.5 Program Execution in the JVM in full small step style

```
theory JVMExec
imports JVMExecInstr
begin
```

**abbreviation**

```
instrs-of :: jvm-prog ⇒ cname ⇒ mname ⇒ instr list where
instrs-of P C M == fst(snd(snd(snd(snd(snd(snd(method P C M)))))))
```

```
fun curr-instr :: jvm-prog ⇒ frame ⇒ instr where
curr-instr P (stk,loc,C,M,pc,ics) = instrs-of P C M ! pc
```

— execution of single step of the initialization procedure

```
fun exec-Calling :: [cname, cname list, jvm-prog, heap, val list, val list,
cname, mname, pc, frame list, sheap] ⇒ jvm-state
```

**where**

```
exec-Calling C Cs P h stk loc C0 M0 pc frs sh =
  (case sh C of
    None ⇒ (None, h, (stk, loc, C0, M0, pc, Calling C Cs)#frs, sh(C := Some (sblank P C,
Prepared)))
  | Some (obj, iflag) ⇒
    (case iflag of
      Done ⇒ (None, h, (stk, loc, C0, M0, pc, Called Cs)#frs, sh)
    | Processing ⇒ (None, h, (stk, loc, C0, M0, pc, Called Cs)#frs, sh)
    | Error ⇒ (None, h, (stk, loc, C0, M0, pc,
      Throwing Cs (addr-of-sys-xcpt NoClassDefFoundError))#frs, sh)
    | Prepared ⇒
      let sh' = sh(C:=Some(fst(the(sh C))), Processing);
          D = fst(the(class P C))
      in if C = Object
        then (None, h, (stk, loc, C0, M0, pc, Called (C#Cs))#frs, sh')
        else (None, h, (stk, loc, C0, M0, pc, Calling D (C#Cs))#frs, sh')
    )
  )
```

— single step of execution without error handling

```
fun exec-step :: [jvm-prog, heap, val list, val list,
cname, mname, pc, init-call-status, frame list, sheap] ⇒ jvm-state
```

**where**

```
exec-step P h stk loc C M pc (Calling C' Cs) frs sh
  = exec-Calling C' Cs P h stk loc C M pc frs sh |
exec-step P h stk loc C M pc (Called (C'#Cs)) frs sh
  = (None, h, create-init-frame P C'#(stk, loc, C, M, pc, Called Cs)#frs, sh) |
exec-step P h stk loc C M pc (Throwing (C'#Cs) a) frs sh
  = (None, h, (stk,loc,C,M,pc,Throwing Cs a)#frs, sh(C':=Some(fst(the(sh C')), Error))) |
exec-step P h stk loc C M pc (Throwing [] a) frs sh
  = ([a], h, (stk,loc,C,M,pc,No-ics)#frs, sh) |
exec-step P h stk loc C M pc ics frs sh
  = exec-instr (instrs-of P C M ! pc) P h stk loc C M pc ics frs sh
```

— execution including error handling

**fun** *exec* :: *jvm-prog* × *jvm-state* ⇒ *jvm-state option* — single step execution **where**  
*exec* (*P*, *None*, *h*, (*stk,loc,C,M,pc,ics*)#*frs*, *sh*) =  
 (let (*xp'*, *h'*, *frs'*, *sh'*) = *exec-step P h stk loc C M pc ics frs sh*  
   in case *xp'* of  
     *None* ⇒ *Some (None,h',frs',sh')*  
     | *Some x* ⇒ *Some (find-handler P x h ((stk,loc,C,M,pc,ics)#frs) sh)*  
 )  
 | *exec* - = *None*

— relational view

**inductive-set**

*exec-1* :: *jvm-prog* ⇒ (*jvm-state* × *jvm-state*) *set*  
**and** *exec-1'* :: *jvm-prog* ⇒ *jvm-state* ⇒ *jvm-state* ⇒ *bool*  
 (⟨- | / - -jvm→<sub>1</sub> / -⟩ [61,61,61] 60)  
**for** *P* :: *jvm-prog*  
**where**  
*P* ⊢ *σ* -jvm→<sub>1</sub> *σ'* ≡ (*σ,σ'*) ∈ *exec-1 P*  
 | *exec-1I*: *exec (P,σ) = Some σ' ⇒ P ⊢ σ -jvm→<sub>1</sub> σ'*

— reflexive transitive closure:

**definition** *exec-all* :: *jvm-prog* ⇒ *jvm-state* ⇒ *jvm-state* ⇒ *bool*  
 (⟨- | / - -jvm→ / -⟩ [61,61,61] 60) **where**  
*exec-all-def1*: *P* ⊢ *σ* -jvm→ *σ'* ↔ (*σ,σ'*) ∈ (*exec-1 P*)\*

**notation** (*ASCII*)

*exec-all* (⟨- | / - -jvm→ / -⟩ [61,61,61] 60)

**lemma** *exec-1-eq*:

*exec-1 P* = {(*σ,σ'*). *exec (P,σ) = Some σ'*}⟨*proof*⟩

**lemma** *exec-1-iff*:

*P* ⊢ *σ* -jvm→<sub>1</sub> *σ'* = (*exec (P,σ) = Some σ'*)⟨*proof*⟩

**lemma** *exec-all-def*:

*P* ⊢ *σ* -jvm→ *σ'* = ((*σ,σ'*) ∈ {(*σ,σ'*). *exec (P,σ) = Some σ'*}\*)⟨*proof*⟩

**lemma** *jvm-refl*[*iff*]: *P* ⊢ *σ* -jvm→ *σ*⟨*proof*⟩

**lemma** *jvm-trans*[*trans*]:

[[ *P* ⊢ *σ* -jvm→ *σ'*; *P* ⊢ *σ'* -jvm→ *σ''* ]] ⇒ *P* ⊢ *σ* -jvm→ *σ''*⟨*proof*⟩

**lemma** *jvm-one-step1*[*trans*]:

[[ *P* ⊢ *σ* -jvm→<sub>1</sub> *σ'*; *P* ⊢ *σ'* -jvm→ *σ''* ]] ⇒ *P* ⊢ *σ* -jvm→ *σ''*⟨*proof*⟩

**lemma** *jvm-one-step2*[*trans*]:

[[ *P* ⊢ *σ* -jvm→ *σ'*; *P* ⊢ *σ'* -jvm→<sub>1</sub> *σ''* ]] ⇒ *P* ⊢ *σ* -jvm→ *σ''*⟨*proof*⟩

**lemma** *exec-all-conf*:

[[ *P* ⊢ *σ* -jvm→ *σ'*; *P* ⊢ *σ* -jvm→ *σ''* ]]  
 ⇒ *P* ⊢ *σ'* -jvm→ *σ''* ∨ *P* ⊢ *σ''* -jvm→ *σ'*⟨*proof*⟩

**lemma** *exec-1-exec-all-conf*:

[[ *exec (P, σ) = Some σ'*; *P* ⊢ *σ* -jvm→ *σ''*; *σ* ≠ *σ''* ]]  
 ⇒ *P* ⊢ *σ'* -jvm→ *σ''*  
 ⟨*proof*⟩

**lemma** *exec-all-finalD*: *P* ⊢ (*x*, *h*, [], *sh*) -jvm→ *σ* ⇒ *σ* = (*x*, *h*, [], *sh*)⟨*proof*⟩

**lemma** *exec-all-deterministic*:

[[ *P* ⊢ *σ* -jvm→ (*x,h,[],sh*); *P* ⊢ *σ* -jvm→ *σ'* ]] ⇒ *P* ⊢ *σ'* -jvm→ (*x,h,[],sh*)⟨*proof*⟩

### 2.5.1 Preservation of preallocated

**lemma** *exec-Calling-prealloc-pres*:

**assumes** *preallocated h*

**and** *exec-Calling C Cs P h stk loc C<sub>0</sub> M<sub>0</sub> pc frs sh = (xp',h',frs',sh')*

**shows** *preallocated h'*

*<proof>*

**lemma** *exec-step-prealloc-pres*:

**assumes** *pre: preallocated h*

**and** *exec-step P h stk loc C M pc ics frs sh = (xp',h',frs',sh')*

**shows** *preallocated h'*

*<proof>*

**lemma** *exec-prealloc-pres*:

**assumes** *pre: preallocated h*

**and** *exec (P, xp, h, frs, sh) = Some(xp',h',frs',sh')*

**shows** *preallocated h'*

*<proof>*

### 2.5.2 Start state

The *Start* class is defined based on a given initial class and method. It has two methods: one that calls the initial method in the initial class, which is called by the starting program, and *clinit*, as required for the class to be well-formed.

**definition** *start-method* :: *cname*  $\Rightarrow$  *mname*  $\Rightarrow$  *jvm-method mdecl* **where**

*start-method C M = (start-m, Static, [], Void, (1,0,[Invokestatic C M 0,Return],[]))*

**abbreviation** *start-clinit* :: *jvm-method mdecl* **where**

*start-clinit  $\equiv$  (clinit, Static, [], Void, (1,0,[Push Unit,Return],[]))*

**definition** *start-class* :: *cname*  $\Rightarrow$  *mname*  $\Rightarrow$  *jvm-method cdecl* **where**

*start-class C M = (Start, Object, [], [start-method C M, start-clinit])*

The start configuration of the JVM in program *P*: in the start heap, we call the “start” method of the “Start”; this method performs *Invokestatic* on the class and method given to create the start program’s *Start* class. This allows the initialization procedure to be called on the initial class in a natural way before the initial method is performed. There is no *this* pointer of the frame as *start* is *Static*. The start sheap has every class pre-prepared; this decision is not necessary. The starting program includes the added *Start* class, given a method *M* of class *C*, added to program *P*.

**definition** *start-state* :: *jvm-prog*  $\Rightarrow$  *jvm-state* **where**

*start-state P = (None, start-heap P, [([], [], Start, start-m, 0, No-ics)], start-sheap)*

**abbreviation** *start-prog* :: *jvm-prog*  $\Rightarrow$  *cname*  $\Rightarrow$  *mname*  $\Rightarrow$  *jvm-prog* **where**

*start-prog P C M  $\equiv$  start-class C M # P*

**end**

## 2.6 A Defensive JVM

**theory** *JVMDefensive*

**imports** *JVMExec ../Common/Conform*

**begin**

Extend the state space by one element indicating a type error (or other abnormal termination)

**datatype**  $'a$  *type-error* = *TypeError* | *Normal 'a*

**fun** *is-Addr* :: *val*  $\Rightarrow$  *bool* **where**

*is-Addr* (*Addr a*)  $\longleftrightarrow$  *True*

| *is-Addr v*  $\longleftrightarrow$  *False*

**fun** *is-Intg* :: *val*  $\Rightarrow$  *bool* **where**

*is-Intg* (*Intg i*)  $\longleftrightarrow$  *True*

| *is-Intg v*  $\longleftrightarrow$  *False*

**fun** *is-Bool* :: *val*  $\Rightarrow$  *bool* **where**

*is-Bool* (*Bool b*)  $\longleftrightarrow$  *True*

| *is-Bool v*  $\longleftrightarrow$  *False*

**definition** *is-Ref* :: *val*  $\Rightarrow$  *bool* **where**

*is-Ref v*  $\longleftrightarrow$   $v = \text{Null} \vee \text{is-Addr } v$

**primrec** *check-instr* :: [*instr*, *jvm-prog*, *heap*, *val list*, *val list*,  
*cname*, *mname*, *pc*, *frame list*, *sheap*]  $\Rightarrow$  *bool* **where**

*check-instr-Load*:

*check-instr* (*Load n*) *P h stk loc C M<sub>0</sub> pc frs sh* =  
( $n < \text{length } \text{loc}$ )

| *check-instr-Store*:

*check-instr* (*Store n*) *P h stk loc C<sub>0</sub> M<sub>0</sub> pc frs sh* =  
( $0 < \text{length } \text{stk} \wedge n < \text{length } \text{loc}$ )

| *check-instr-Push*:

*check-instr* (*Push v*) *P h stk loc C<sub>0</sub> M<sub>0</sub> pc frs sh* =  
( $\neg \text{is-Addr } v$ )

| *check-instr-New*:

*check-instr* (*New C*) *P h stk loc C<sub>0</sub> M<sub>0</sub> pc frs sh* =  
*is-class P C*

| *check-instr-Getfield*:

*check-instr* (*Getfield F C*) *P h stk loc C<sub>0</sub> M<sub>0</sub> pc frs sh* =  
( $0 < \text{length } \text{stk} \wedge (\exists C' T. P \vdash C \text{ sees } F, \text{NonStatic}:T \text{ in } C') \wedge$   
(*let* (*C'*, *b*, *T*) = *field P C F*; *ref* = *hd stk* *in*  
 $C' = C \wedge \text{is-Ref } \text{ref} \wedge (\text{ref} \neq \text{Null} \longrightarrow$   
 $h (\text{the-Addr } \text{ref}) \neq \text{None} \wedge$   
(*let* (*D*,*vs*) = *the* (*h* (*the-Addr ref*)) *in*  
 $P \vdash D \preceq^* C \wedge \text{vs } (F,C) \neq \text{None} \wedge P, h \vdash \text{the } (\text{vs } (F,C)) \preceq T$ ))))

| *check-instr-Getstatic*:

*check-instr* (*Getstatic C F D*) *P h stk loc C<sub>0</sub> M<sub>0</sub> pc frs sh* =  
( $(\exists T. P \vdash C \text{ sees } F, \text{Static}:T \text{ in } D) \wedge$   
(*let* (*C'*, *b*, *T*) = *field P C F* *in*  
 $C' = D \wedge (\text{sh } D \neq \text{None} \longrightarrow$   
(*let* (*sfs*,*i*) = *the* (*sh D*) *in*  
 $\text{sfs } F \neq \text{None} \wedge P, h \vdash \text{the } (\text{sfs } F) \preceq T$ ))))



| *check-instr-Putfield*:

$$\begin{aligned} & \text{check-instr (Putfield } F \ C) \ P \ h \ \text{stk} \ \text{loc} \ C_0 \ M_0 \ \text{pc} \ \text{frs} \ \text{sh} = \\ & (1 < \text{length} \ \text{stk} \wedge (\exists C' \ T. \ P \vdash C \ \text{sees} \ F, \text{NonStatic}:T \ \text{in} \ C') \wedge \\ & (\text{let } (C', \ b, \ T) = \text{field} \ P \ C \ F; \ v = \text{hd} \ \text{stk}; \ \text{ref} = \text{hd} \ (\text{tl} \ \text{stk}) \ \text{in} \\ & \quad C' = C \wedge \text{is-Ref} \ \text{ref} \wedge (\text{ref} \neq \text{Null} \longrightarrow \\ & \quad \quad h \ (\text{the-Addr} \ \text{ref}) \neq \text{None} \wedge \\ & \quad \quad (\text{let } D = \text{fst} \ (\text{the} \ (h \ (\text{the-Addr} \ \text{ref})))) \ \text{in} \\ & \quad \quad P \vdash D \preceq^* C \wedge P, h \vdash v : \leq T))) \end{aligned}$$

| *check-instr-Putstatic*:

$$\begin{aligned} & \text{check-instr (Putstatic } C \ F \ D) \ P \ h \ \text{stk} \ \text{loc} \ C_0 \ M_0 \ \text{pc} \ \text{frs} \ \text{sh} = \\ & (0 < \text{length} \ \text{stk} \wedge (\exists T. \ P \vdash C \ \text{sees} \ F, \text{Static}:T \ \text{in} \ D) \wedge \\ & (\text{let } (C', \ b, \ T) = \text{field} \ P \ C \ F; \ v = \text{hd} \ \text{stk} \ \text{in} \\ & \quad C' = D \wedge P, h \vdash v : \leq T)) \end{aligned}$$

| *check-instr-Checkcast*:

$$\begin{aligned} & \text{check-instr (Checkcast } C) \ P \ h \ \text{stk} \ \text{loc} \ C_0 \ M_0 \ \text{pc} \ \text{frs} \ \text{sh} = \\ & (0 < \text{length} \ \text{stk} \wedge \text{is-class} \ P \ C \wedge \text{is-Ref} \ (\text{hd} \ \text{stk})) \end{aligned}$$

| *check-instr-Invoke*:

$$\begin{aligned} & \text{check-instr (Invoke } M \ n) \ P \ h \ \text{stk} \ \text{loc} \ C_0 \ M_0 \ \text{pc} \ \text{frs} \ \text{sh} = \\ & (n < \text{length} \ \text{stk} \wedge \text{is-Ref} \ (\text{stk}!n) \wedge \\ & (\text{stk}!n \neq \text{Null} \longrightarrow \\ & \quad (\text{let } a = \text{the-Addr} \ (\text{stk}!n); \\ & \quad \quad C = \text{cname-of} \ h \ a; \\ & \quad \quad Ts = \text{fst} \ (\text{snd} \ (\text{snd} \ (\text{method} \ P \ C \ M))) \\ & \quad \text{in } h \ a \neq \text{None} \wedge P \vdash C \ \text{has} \ M, \text{NonStatic} \wedge \\ & \quad P, h \vdash \text{rev} \ (\text{take} \ n \ \text{stk}) \ [:\leq] \ Ts))) \end{aligned}$$

| *check-instr-Invokestatic*:

$$\begin{aligned} & \text{check-instr (Invokestatic } C \ M \ n) \ P \ h \ \text{stk} \ \text{loc} \ C_0 \ M_0 \ \text{pc} \ \text{frs} \ \text{sh} = \\ & (n \leq \text{length} \ \text{stk} \wedge \\ & \quad (\text{let } Ts = \text{fst} \ (\text{snd} \ (\text{snd} \ (\text{method} \ P \ C \ M))) \\ & \quad \text{in } P \vdash C \ \text{has} \ M, \text{Static} \wedge \\ & \quad P, h \vdash \text{rev} \ (\text{take} \ n \ \text{stk}) \ [:\leq] \ Ts)) \end{aligned}$$

| *check-instr-Return*:

$$\begin{aligned} & \text{check-instr Return } P \ h \ \text{stk} \ \text{loc} \ C_0 \ M_0 \ \text{pc} \ \text{frs} \ \text{sh} = \\ & (\text{case } (M_0 = \text{clinit}) \ \text{of} \ \text{False} \Rightarrow (0 < \text{length} \ \text{stk} \wedge ((0 < \text{length} \ \text{frs}) \longrightarrow \\ & \quad (\exists b. \ P \vdash C_0 \ \text{has} \ M_0, b) \wedge \\ & \quad (\text{let } v = \text{hd} \ \text{stk}; \\ & \quad \quad T = \text{fst} \ (\text{snd} \ (\text{snd} \ (\text{snd} \ (\text{method} \ P \ C_0 \ M_0)))) \\ & \quad \quad \text{in } P, h \vdash v : \leq T))) \\ & \quad | \ \text{True} \Rightarrow P \vdash C_0 \ \text{has} \ M_0, \text{Static}) \end{aligned}$$

| *check-instr-Pop*:

$$\begin{aligned} & \text{check-instr Pop } P \ h \ \text{stk} \ \text{loc} \ C_0 \ M_0 \ \text{pc} \ \text{frs} \ \text{sh} = \\ & (0 < \text{length} \ \text{stk}) \end{aligned}$$

| *check-instr-IAdd*:

$$\begin{aligned} & \text{check-instr IAdd } P \ h \ \text{stk} \ \text{loc} \ C_0 \ M_0 \ \text{pc} \ \text{frs} \ \text{sh} = \\ & (1 < \text{length} \ \text{stk} \wedge \text{is-Intg} \ (\text{hd} \ \text{stk}) \wedge \text{is-Intg} \ (\text{hd} \ (\text{tl} \ \text{stk}))) \end{aligned}$$

| *check-instr-IfFalse*:

*check-instr* (*IfFalse* *b*) *P h stk loc C<sub>0</sub> M<sub>0</sub> pc frs sh* =  
 ( $0 < \text{length } \text{stk} \wedge \text{is-Bool } (\text{hd } \text{stk}) \wedge 0 \leq \text{int } \text{pc} + \text{b}$ )

| *check-instr-CmpEq*:  
*check-instr* *CmpEq* *P h stk loc C<sub>0</sub> M<sub>0</sub> pc frs sh* =  
 ( $1 < \text{length } \text{stk}$ )

| *check-instr-Goto*:  
*check-instr* (*Goto* *b*) *P h stk loc C<sub>0</sub> M<sub>0</sub> pc frs sh* =  
 ( $0 \leq \text{int } \text{pc} + \text{b}$ )

| *check-instr-Throw*:  
*check-instr* *Throw* *P h stk loc C<sub>0</sub> M<sub>0</sub> pc frs sh* =  
 ( $0 < \text{length } \text{stk} \wedge \text{is-Ref } (\text{hd } \text{stk})$ )

**definition** *check* :: *jvm-prog*  $\Rightarrow$  *jvm-state*  $\Rightarrow$  *bool* **where**

*check* *P*  $\sigma$  = (*let* (*xcpt*, *h*, *frs*, *sh*) =  $\sigma$  *in*  
 (*case* *frs* *of* []  $\Rightarrow$  *True* | (*stk*, *loc*, *C*, *M*, *pc*, *ics*)#*frs'*  $\Rightarrow$   
 $\exists$  *b*. *P*  $\vdash$  *C* *has* *M*, *b*  $\wedge$   
 (*let* (*C'*, *b*, *Ts*, *T*, *mxs*, *mxl<sub>0</sub>*, *ins*, *xt*) = *method* *P* *C* *M*; *i* = *ins!**pc* *in*  
 $\text{pc} < \text{size } \text{ins} \wedge \text{size } \text{stk} \leq \text{mxs} \wedge$   
*check-instr* *i* *P* *h* *stk* *loc* *C* *M* *pc* *frs'* *sh*)))

**definition** *exec-d* :: *jvm-prog*  $\Rightarrow$  *jvm-state*  $\Rightarrow$  *jvm-state option type-error* **where**

*exec-d* *P*  $\sigma$  = (*if* *check* *P*  $\sigma$  *then* *Normal* (*exec* (*P*,  $\sigma$ )) *else* *TypeError*)

**inductive-set**

*exec-1-d* :: *jvm-prog*  $\Rightarrow$  (*jvm-state type-error*  $\times$  *jvm-state type-error*) *set*  
**and** *exec-1-d'* :: *jvm-prog*  $\Rightarrow$  *jvm-state type-error*  $\Rightarrow$  *jvm-state type-error*  $\Rightarrow$  *bool*  
 ( $\langle \cdot \vdash - \text{-jvmd} \rightarrow_1 \cdot \rangle [61, 61, 61] 60$ )

**for** *P* :: *jvm-prog*

**where**

*P*  $\vdash$   $\sigma \text{-jvmd} \rightarrow_1 \sigma' \equiv (\sigma, \sigma') \in \text{exec-1-d } P$

| *exec-1-d-ErrorI*: *exec-d* *P*  $\sigma$  = *TypeError*  $\implies P \vdash \text{Normal } \sigma \text{-jvmd} \rightarrow_1 \text{TypeError}$

| *exec-1-d-NormalI*: *exec-d* *P*  $\sigma$  = *Normal* (*Some*  $\sigma'$ )  $\implies P \vdash \text{Normal } \sigma \text{-jvmd} \rightarrow_1 \text{Normal } \sigma'$

— reflexive transitive closure:

**definition** *exec-all-d* :: *jvm-prog*  $\Rightarrow$  *jvm-state type-error*  $\Rightarrow$  *jvm-state type-error*  $\Rightarrow$  *bool*

( $\langle \cdot \vdash - \text{-jvmd} \rightarrow \cdot \rangle [61, 61, 61] 60$ ) **where**

*exec-all-d-def1*: *P*  $\vdash$   $\sigma \text{-jvmd} \rightarrow \sigma' \iff (\sigma, \sigma') \in (\text{exec-1-d } P)^*$

**notation** (*ASCII*)

*exec-all-d* ( $\langle \cdot \vdash - \text{-jvmd} \rightarrow \cdot \rangle [61, 61, 61] 60$ )

**lemma** *exec-1-d-eq*:

*exec-1-d* *P* =  $\{(s, t). \exists \sigma. s = \text{Normal } \sigma \wedge t = \text{TypeError} \wedge \text{exec-d } P \sigma = \text{TypeError}\} \cup$   
 $\{(s, t). \exists \sigma \sigma'. s = \text{Normal } \sigma \wedge t = \text{Normal } \sigma' \wedge \text{exec-d } P \sigma = \text{Normal } (\text{Some } \sigma')\}$

*<proof>*

**declare** *split-paired-All* [*simp del*]

**declare** *split-paired-Ex* [*simp del*]

**lemma** *if-neq* [*dest!*]:  
 (if  $P$  then  $A$  else  $B$ )  $\neq B \implies P$   
 ⟨*proof*⟩

**lemma** *exec-d-no-errorI* [*intro*]:  
 check  $P \sigma \implies \text{exec-d } P \sigma \neq \text{TypeError}$   
 ⟨*proof*⟩

**theorem** *no-type-error-commutes*:  
 exec-d  $P \sigma \neq \text{TypeError} \implies \text{exec-d } P \sigma = \text{Normal } (\text{exec } (P, \sigma))$   
 ⟨*proof*⟩

**lemma** *defensive-imp-aggressive*:  
 $P \vdash (\text{Normal } \sigma) \text{ -jvmd} \rightarrow (\text{Normal } \sigma') \implies P \vdash \sigma \text{ -jvm} \rightarrow \sigma'$ ⟨*proof*⟩  
**end**

## 2.7 The Jinja Type System as a Semilattice

**theory** *SemiType*  
**imports** ../Common/WellForm Jinja.Semilattices  
**begin**

**definition** *super* :: 'a prog  $\Rightarrow$  cname  $\Rightarrow$  cname  
**where** *super*  $P C \equiv \text{fst } (\text{the } (\text{class } P C))$

**lemma** *superI*:  
 $(C, D) \in \text{subcls1 } P \implies \text{super } P C = D$   
 ⟨*proof*⟩

**primrec** *the-Class* :: ty  $\Rightarrow$  cname  
**where**  
*the-Class* (Class  $C$ ) =  $C$

**definition** *sup* :: 'c prog  $\Rightarrow$  ty  $\Rightarrow$  ty  $\Rightarrow$  ty err  
**where**  
 $\text{sup } P T_1 T_2 \equiv$   
 if *is-refT*  $T_1 \wedge \text{is-refT } T_2$  then  
 OK (if  $T_1 = \text{NT}$  then  $T_2$  else  
 if  $T_2 = \text{NT}$  then  $T_1$  else  
 (Class (exec-lub (subcls1  $P$ ) (super  $P$ ) (the-Class  $T_1$ ) (the-Class  $T_2$ ))))  
 else  
 (if  $T_1 = T_2$  then OK  $T_1$  else Err)

**lemma** *sup-def'*:  
 $\text{sup } P = (\lambda T_1 T_2.$   
 if *is-refT*  $T_1 \wedge \text{is-refT } T_2$  then  
 OK (if  $T_1 = \text{NT}$  then  $T_2$  else  
 if  $T_2 = \text{NT}$  then  $T_1$  else  
 (Class (exec-lub (subcls1  $P$ ) (super  $P$ ) (the-Class  $T_1$ ) (the-Class  $T_2$ ))))  
 else

(if  $T_1 = T_2$  then OK  $T_1$  else Err)  
 ⟨proof⟩

**abbreviation**

$subtype :: 'c \text{ prog} \Rightarrow ty \Rightarrow ty \Rightarrow bool$   
**where**  $subtype P \equiv widen P$

**definition**  $esl :: 'c \text{ prog} \Rightarrow ty \text{ esl}$

**where**

$esl P \equiv (types P, subtype P, sup P)$

**lemma** *is-class-is-subcls*:

$wf\text{-prog } m P \Longrightarrow is\text{-class } P C = P \vdash C \preceq^* Object \langle proof \rangle$

**lemma** *subcls-antisym*:

$\llbracket wf\text{-prog } m P; P \vdash C \preceq^* D; P \vdash D \preceq^* C \rrbracket \Longrightarrow C = D$   
 ⟨proof⟩

**lemma** *widen-antisym*:

$\llbracket wf\text{-prog } m P; P \vdash T \leq U; P \vdash U \leq T \rrbracket \Longrightarrow T = U \langle proof \rangle$

**lemma** *order-widen* [intro, simp]:

$wf\text{-prog } m P \Longrightarrow order (subtype P) (types P) \langle proof \rangle$

**lemma** *NT-widen*:

$P \vdash NT \leq T = (T = NT \vee (\exists C. T = Class C)) \langle proof \rangle$

**lemma** *Class-widen2*:  $P \vdash Class C \leq T = (\exists D. T = Class D \wedge P \vdash C \preceq^* D) \langle proof \rangle$

**lemma** *wf-converse-subcls1-impl-acc-subtype*:

$wf ((subcls1 P) \hat{-} 1) \Longrightarrow acc (subtype P) \langle proof \rangle$

**lemma** *wf-subtype-acc* [intro, simp]:

$wf\text{-prog } wf\text{-mb } P \Longrightarrow acc (subtype P) \langle proof \rangle$

**lemma** *exec-lub-refl* [simp]:  $exec\text{-lub } r f T T = T \langle proof \rangle$

**lemma** *closed-err-types*:

$wf\text{-prog } wf\text{-mb } P \Longrightarrow closed (err (types P)) (lift2 (sup P)) \langle proof \rangle$

**lemma** *sup-subtype-greater*:

$\llbracket wf\text{-prog } wf\text{-mb } P; is\text{-type } P t1; is\text{-type } P t2; sup P t1 t2 = OK s \rrbracket$   
 $\Longrightarrow subtype P t1 s \wedge subtype P t2 s \langle proof \rangle$

**lemma** *sup-subtype-smallest*:

$\llbracket wf\text{-prog } wf\text{-mb } P; is\text{-type } P a; is\text{-type } P b; is\text{-type } P c;$   
 $subtype P a c; subtype P b c; sup P a b = OK d \rrbracket$   
 $\Longrightarrow subtype P d c \langle proof \rangle$

**lemma** *sup-exists*:

$\llbracket subtype P a c; subtype P b c \rrbracket \Longrightarrow \exists T. sup P a b = OK T \langle proof \rangle$

**lemma** *err-semilat-JType-esl*:

$wf\text{-prog } wf\text{-mb } P \Longrightarrow err\text{-semilat } (esl P) \langle proof \rangle$

**end**

## 2.8 The JVM Type System as Semilattice

**theory** *JVM-SemiType* **imports** *SemiType* **begin**

**type-synonym**  $ty_l = ty \text{ err list}$   
**type-synonym**  $ty_s = ty \text{ list}$   
**type-synonym**  $ty_i = ty_s \times ty_l$   
**type-synonym**  $ty_i' = ty_i \text{ option}$   
**type-synonym**  $ty_m = ty_i' \text{ list}$   
**type-synonym**  $ty_P = mname \Rightarrow cname \Rightarrow ty_m$

**definition**  $stk\text{-}esl :: 'c \text{ prog} \Rightarrow nat \Rightarrow ty_s \text{ esl}$   
**where**  
 $stk\text{-}esl P \ mxs \equiv upto\text{-}esl \ mxs \ (SemiType.esl \ P)$

**definition**  $loc\text{-}sl :: 'c \text{ prog} \Rightarrow nat \Rightarrow ty_l \text{ sl}$   
**where**  
 $loc\text{-}sl P \ m\lambda l \equiv Listn.sl \ m\lambda l \ (Err.sl \ (SemiType.esl \ P))$

**definition**  $sl :: 'c \text{ prog} \Rightarrow nat \Rightarrow nat \Rightarrow ty_i' \text{ err sl}$   
**where**  
 $sl P \ mxs \ m\lambda l \equiv$   
 $Err.sl(Opt.esl(Product.esl(stk\text{-}esl \ P \ mxs) \ (Err.esl(loc\text{-}sl \ P \ m\lambda l))))$

**definition**  $states :: 'c \text{ prog} \Rightarrow nat \Rightarrow nat \Rightarrow ty_i' \text{ err set}$   
**where**  $states P \ mxs \ m\lambda l \equiv fst(sl \ P \ mxs \ m\lambda l)$

**definition**  $le :: 'c \text{ prog} \Rightarrow nat \Rightarrow nat \Rightarrow ty_i' \text{ err ord}$   
**where**  
 $le P \ mxs \ m\lambda l \equiv fst(snd(sl \ P \ mxs \ m\lambda l))$

**definition**  $sup :: 'c \text{ prog} \Rightarrow nat \Rightarrow nat \Rightarrow ty_i' \text{ err binop}$   
**where**  
 $sup P \ mxs \ m\lambda l \equiv snd(snd(sl \ P \ mxs \ m\lambda l))$

**definition**  $sup\text{-}ty\text{-}opt :: ['c \text{ prog}, ty \text{ err}, ty \text{ err}] \Rightarrow bool$   
 $(\langle - \vdash - \leq_T \rangle \rightarrow [71, 71, 71] \ 70)$   
**where**  
 $sup\text{-}ty\text{-}opt P \equiv Err.le \ (subtype \ P)$

**definition**  $sup\text{-}state :: ['c \text{ prog}, ty_i, ty_i] \Rightarrow bool$   
 $(\langle - \vdash - \leq_i \rangle \rightarrow [71, 71, 71] \ 70)$   
**where**  
 $sup\text{-}state P \equiv Product.le \ (Listn.le \ (subtype \ P)) \ (Listn.le \ (sup\text{-}ty\text{-}opt \ P))$

**definition**  $sup\text{-}state\text{-}opt :: ['c \text{ prog}, ty_i', ty_i'] \Rightarrow bool$   
 $(\langle - \vdash - \leq'' \rangle \rightarrow [71, 71, 71] \ 70)$   
**where**  
 $sup\text{-}state\text{-}opt P \equiv Opt.le \ (sup\text{-}state \ P)$

**abbreviation**

*sup-loc* :: [*c prog, ty<sub>i</sub>, ty<sub>i</sub>*] ⇒ *bool* (↪ ⊢ - [≤<sub>⊤</sub>] → [71, 71, 71] 70)  
**where** *P* ⊢ *LT* [≤<sub>⊤</sub>] *LT'* ≡ *list-all2* (*sup-ty-opt* *P*) *LT* *LT'*

**notation** (*ASCII*)

*sup-ty-opt* (↪ | - - <= *T* → [71, 71, 71] 70) **and**  
*sup-state* (↪ | - - <= *i* → [71, 71, 71] 70) **and**  
*sup-state-opt* (↪ | - - <= ' → [71, 71, 71] 70) **and**  
*sup-loc* (↪ | - - [≤<sub>⊤</sub>] → [71, 71, 71] 70)

## 2.8.1 Unfolding

**lemma** *JVM-states-unfold*:

*states* *P mxs mxl* ≡ *err*(*opt*((*Union* {*nlists* *n* (*types* *P*) | *n*. *n* <= *mxs*} ) ×  
*nlists* *mxl* (*err*(*types* *P*))))⟨*proof*⟩

**lemma** *JVM-le-unfold*:

*le* *P m n* ≡  
*Err.le*(*Opt.le*(*Product.le*(*Listn.le*(*subtype* *P*))(*Listn.le*(*Err.le*(*subtype* *P*))))))⟨*proof*⟩

**lemma** *sl-def2*:

*JVM-SemiType.sl* *P mxs mxl* ≡  
(*states* *P mxs mxl*, *JVM-SemiType.le* *P mxs mxl*, *JVM-SemiType.sup* *P mxs mxl*)⟨*proof*⟩

**lemma** *JVM-le-conv*:

*le* *P m n* (*OK* *t1*) (*OK* *t2*) = *P* ⊢ *t1* ≤' *t2*⟨*proof*⟩

**lemma** *JVM-le-Err-conv*:

*le* *P m n* = *Err.le* (*sup-state-opt* *P*)⟨*proof*⟩

**lemma** *err-le-unfold* [*iff*]:

*Err.le* *r* (*OK* *a*) (*OK* *b*) = *r* *a* *b*⟨*proof*⟩

## 2.8.2 Semilattice

**lemma** *order-sup-state-opt'* [*intro*, *simp*]:

*wf-prog* *wf-mb* *P* ⇒  
*order* (*sup-state-opt* *P*) (*opt* ((*⋃* {*nlists* *n* (*types* *P*) | *n*. *n* ≤ *mxs*} ) × *nlists* (*Suc* (*length* *Ts* +  
*mxl*<sub>0</sub>)) (*err* (*types* *P*))))⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩

## 2.9 Effect of Instructions on the State Type

**theory** *Effect*

**imports** *JVM-SemiType* ../*JVM*/*JVMExceptions*

**begin**

— *FIXME*

**locale** *prog* =

**fixes** *P* :: 'a *prog*

**locale** *jvm-method* = *prog* +

**fixes** *mxs* :: *nat*

**fixes** *mxl*<sub>0</sub> :: *nat*

**fixes** *Ts* :: *ty list*

**fixes** *T<sub>τ</sub>* :: *ty*

**fixes** *is* :: *instr list*

**fixes** *xt* :: *ex-table*

**fixes** *mxl* :: *nat*

**defines** *max-def*:  $max \equiv 1 + size\ Ts + max_0$

Program counter of successor instructions:

**primrec** *succs* ::  $instr \Rightarrow ty_i \Rightarrow pc \Rightarrow pc\ list$  **where**

- | *succs* (*Load idx*)  $\tau\ pc = [pc+1]$
- | *succs* (*Store idx*)  $\tau\ pc = [pc+1]$
- | *succs* (*Push v*)  $\tau\ pc = [pc+1]$
- | *succs* (*Getfield F C*)  $\tau\ pc = [pc+1]$
- | *succs* (*Getstatic C F D*)  $\tau\ pc = [pc+1]$
- | *succs* (*Putfield F C*)  $\tau\ pc = [pc+1]$
- | *succs* (*Putstatic C F D*)  $\tau\ pc = [pc+1]$
- | *succs* (*New C*)  $\tau\ pc = [pc+1]$
- | *succs* (*Checkcast C*)  $\tau\ pc = [pc+1]$
- | *succs* *Pop*  $\tau\ pc = [pc+1]$
- | *succs* *IAdd*  $\tau\ pc = [pc+1]$
- | *succs* *CmpEq*  $\tau\ pc = [pc+1]$
- | *succs-IfFalse*:
  - | *succs* (*IfFalse b*)  $\tau\ pc = [pc+1, nat\ (int\ pc + b)]$
- | *succs-Goto*:
  - | *succs* (*Goto b*)  $\tau\ pc = [nat\ (int\ pc + b)]$
- | *succs-Return*:
  - | *succs* *Return*  $\tau\ pc = []$
- | *succs-Invoke*:
  - | *succs* (*Invoke M n*)  $\tau\ pc = (if\ (fst\ \tau)!n = NT\ then\ []\ else\ [pc+1])$
- | *succs-Invokestatic*:
  - | *succs* (*Invokestatic C M n*)  $\tau\ pc = [pc+1]$
- | *succs-Throw*:
  - | *succs* *Throw*  $\tau\ pc = []$

Effect of instruction on the state type:

**fun** *the-class*::  $ty \Rightarrow cname$  **where**  
*the-class* (*Class C*) = *C*

**fun** *eff<sub>i</sub>* ::  $instr \times 'm\ prog \times ty_i \Rightarrow ty_i$  **where**

- | *eff<sub>i</sub>-Load*:
  - | *eff<sub>i</sub>* (*Load n, P, (ST, LT)*) = (*ok-val* (*LT ! n*) # *ST, LT*)
- | *eff<sub>i</sub>-Store*:
  - | *eff<sub>i</sub>* (*Store n, P, (T#ST, LT)*) = (*ST, LT[n:= OK T]*)
- | *eff<sub>i</sub>-Push*:
  - | *eff<sub>i</sub>* (*Push v, P, (ST, LT)*) = (*the* (*typeof v*) # *ST, LT*)
- | *eff<sub>i</sub>-Getfield*:
  - | *eff<sub>i</sub>* (*Getfield F C, P, (T#ST, LT)*) = (*snd* (*snd* (*field P C F*)) # *ST, LT*)
- | *eff<sub>i</sub>-Getstatic*:
  - | *eff<sub>i</sub>* (*Getstatic C F D, P, (ST, LT)*) = (*snd* (*snd* (*field P C F*)) # *ST, LT*)
- | *eff<sub>i</sub>-Putfield*:
  - | *eff<sub>i</sub>* (*Putfield F C, P, (T<sub>1</sub>#T<sub>2</sub>#ST, LT)*) = (*ST, LT*)
- | *eff<sub>i</sub>-Putstatic*:
  - | *eff<sub>i</sub>* (*Putstatic C F D, P, (T#ST, LT)*) = (*ST, LT*)
- | *eff<sub>i</sub>-New*:
  - | *eff<sub>i</sub>* (*New C, P, (ST, LT)*) = (*Class C* # *ST, LT*)
- | *eff<sub>i</sub>-Checkcast*:
  - | *eff<sub>i</sub>* (*Checkcast C, P, (T#ST, LT)*) = (*Class C* # *ST, LT*)
- | *eff<sub>i</sub>-Pop*:
  - | *eff<sub>i</sub>* (*Pop, P, (T#ST, LT)*) = (*ST, LT*)

$\text{eff}_i\text{-IAdd:}$   
 $\text{eff}_i (IAdd, P, (T_1 \# T_2 \# ST, LT)) = (Integer \# ST, LT)$

$\text{eff}_i\text{-CmpEq:}$   
 $\text{eff}_i (CmpEq, P, (T_1 \# T_2 \# ST, LT)) = (Boolean \# ST, LT)$

$\text{eff}_i\text{-IfFalse:}$   
 $\text{eff}_i (IfFalse b, P, (T_1 \# ST, LT)) = (ST, LT)$

$\text{eff}_i\text{-Invoke:}$   
 $\text{eff}_i (Invoke M n, P, (ST, LT)) =$   
 $(let C = \text{the-class } (ST!n); (D, b, Ts, T_r, m) = \text{method } P C M$   
 $in (T_r \# drop (n+1) ST, LT))$

$\text{eff}_i\text{-Invokestatic:}$   
 $\text{eff}_i (Invokestatic C M n, P, (ST, LT)) =$   
 $(let (D, b, Ts, T_r, m) = \text{method } P C M$   
 $in (T_r \# drop n ST, LT))$

$\text{eff}_i\text{-Goto:}$   
 $\text{eff}_i (Goto n, P, s) = s$

**fun** *is-relevant-class* :: *instr*  $\Rightarrow$  *'m prog*  $\Rightarrow$  *cname*  $\Rightarrow$  *bool* **where**  
*rel-Getfield:*  
 $\text{is-relevant-class } (Getfield F D)$   
 $= (\lambda P C. P \vdash \text{NullPointer} \preceq^* C \vee P \vdash \text{NoSuchFieldError} \preceq^* C$   
 $\vee P \vdash \text{IncompatibleClassChangeError} \preceq^* C)$

*rel-Getstatic:*  
 $\text{is-relevant-class } (Getstatic C F D)$   
 $= (\lambda P C. \text{True})$

*rel-Putfield:*  
 $\text{is-relevant-class } (Putfield F D)$   
 $= (\lambda P C. P \vdash \text{NullPointer} \preceq^* C \vee P \vdash \text{NoSuchFieldError} \preceq^* C$   
 $\vee P \vdash \text{IncompatibleClassChangeError} \preceq^* C)$

*rel-Putstatic:*  
 $\text{is-relevant-class } (Putstatic C F D)$   
 $= (\lambda P C. \text{True})$

*rel-Checkcast:*  
 $\text{is-relevant-class } (Checkcast D) = (\lambda P C. P \vdash \text{ClassCast} \preceq^* C)$

*rel-New:*  
 $\text{is-relevant-class } (New D) = (\lambda P C. \text{True})$

*rel-Throw:*  
 $\text{is-relevant-class } Throw = (\lambda P C. \text{True})$

*rel-Invoke:*  
 $\text{is-relevant-class } (Invoke M n) = (\lambda P C. \text{True})$

*rel-Invokestatic:*  
 $\text{is-relevant-class } (Invokestatic C M n) = (\lambda P C. \text{True})$

*rel-default:*  
 $\text{is-relevant-class } i = (\lambda P C. \text{False})$

**definition** *is-relevant-entry* :: *'m prog*  $\Rightarrow$  *instr*  $\Rightarrow$  *pc*  $\Rightarrow$  *ex-entry*  $\Rightarrow$  *bool* **where**  
 $\text{is-relevant-entry } P i pc e \longleftrightarrow (let (f, t, C, h, d) = e \text{ in } \text{is-relevant-class } i P C \wedge pc \in \{f..<t\})$

**definition** *relevant-entries* :: *'m prog*  $\Rightarrow$  *instr*  $\Rightarrow$  *pc*  $\Rightarrow$  *ex-table*  $\Rightarrow$  *ex-table* **where**  
 $\text{relevant-entries } P i pc = \text{filter } (\text{is-relevant-entry } P i pc)$

**definition** *xcpt-eff* :: *instr*  $\Rightarrow$  *'m prog*  $\Rightarrow$  *pc*  $\Rightarrow$  *ty<sub>i</sub>*  
 $\Rightarrow$  *ex-table*  $\Rightarrow$   $(pc \times ty_i')$  *list* **where**  
 $\text{xcpt-eff } i P pc \tau et = (let (ST, LT) = \tau \text{ in}$



$map (\lambda(f,t,C,h,d). (h, Some (Class C\#drop (size ST - d) ST, LT))) (relevant-entries P i pc et)$

**definition**  $norm\text{-}eff :: instr \Rightarrow 'm prog \Rightarrow nat \Rightarrow ty_i \Rightarrow (pc \times ty_i')$  list **where**  
 $norm\text{-}eff i P pc \tau = map (\lambda pc'. (pc', Some (eff_i (i,P,\tau)))) (succs i \tau pc)$

**definition**  $eff :: instr \Rightarrow 'm prog \Rightarrow pc \Rightarrow ex\text{-}table \Rightarrow ty_i' \Rightarrow (pc \times ty_i')$  list **where**  
 $eff i P pc et t = (case t of$   
 $None \Rightarrow []$   
 $| Some \tau \Rightarrow (norm\text{-}eff i P pc \tau) @ (xcpt\text{-}eff i P pc \tau et))$

**lemma**  $eff\text{-}None$ :

$eff i P pc xt None = []$   
 $\langle proof \rangle$

**lemma**  $eff\text{-}Some$ :

$eff i P pc xt (Some \tau) = norm\text{-}eff i P pc \tau @ xcpt\text{-}eff i P pc \tau xt$   
 $\langle proof \rangle$

Conditions under which  $eff$  is applicable:

**fun**  $app_i :: instr \times 'm prog \times pc \times nat \times ty \times ty_i \Rightarrow bool$  **where**

$app_i\text{-}Load$ :  
 $app_i (Load n, P, pc, mxs, T_r, (ST,LT)) =$   
 $(n < length LT \wedge LT ! n \neq Err \wedge length ST < mxs)$   
 $| app_i\text{-}Store$ :  
 $app_i (Store n, P, pc, mxs, T_r, (T\#ST, LT)) =$   
 $(n < length LT)$   
 $| app_i\text{-}Push$ :  
 $app_i (Push v, P, pc, mxs, T_r, (ST,LT)) =$   
 $(length ST < mxs \wedge typeof v \neq None)$   
 $| app_i\text{-}Getfield$ :  
 $app_i (Getfield F C, P, pc, mxs, T_r, (T\#ST, LT)) =$   
 $(\exists T_f. P \vdash C \text{ sees } F, NonStatic:T_f \text{ in } C \wedge P \vdash T \leq Class C)$   
 $| app_i\text{-}Getstatic$ :  
 $app_i (Getstatic C F D, P, pc, mxs, T_r, (ST, LT)) =$   
 $(length ST < mxs \wedge (\exists T_f. P \vdash C \text{ sees } F, Static:T_f \text{ in } D))$   
 $| app_i\text{-}Putfield$ :  
 $app_i (Putfield F C, P, pc, mxs, T_r, (T_1\#T_2\#ST, LT)) =$   
 $(\exists T_f. P \vdash C \text{ sees } F, NonStatic:T_f \text{ in } C \wedge P \vdash T_2 \leq (Class C) \wedge P \vdash T_1 \leq T_f)$   
 $| app_i\text{-}Putstatic$ :  
 $app_i (Putstatic C F D, P, pc, mxs, T_r, (T\#ST, LT)) =$   
 $(\exists T_f. P \vdash C \text{ sees } F, Static:T_f \text{ in } D \wedge P \vdash T \leq T_f)$   
 $| app_i\text{-}New$ :  
 $app_i (New C, P, pc, mxs, T_r, (ST,LT)) =$   
 $(is\text{-}class P C \wedge length ST < mxs)$   
 $| app_i\text{-}Checkcast$ :  
 $app_i (Checkcast C, P, pc, mxs, T_r, (T\#ST,LT)) =$   
 $(is\text{-}class P C \wedge is\text{-}refT T)$   
 $| app_i\text{-}Pop$ :  
 $app_i (Pop, P, pc, mxs, T_r, (T\#ST,LT)) =$   
 $True$   
 $| app_i\text{-}IAdd$ :  
 $app_i (IAdd, P, pc, mxs, T_r, (T_1\#T_2\#ST,LT)) = (T_1 = T_2 \wedge T_1 = Integer)$   
 $| app_i\text{-}CmpEq$ :

$$\begin{aligned}
& \text{app}_i (\text{CmpEq}, P, pc, mxs, T_r, (T_1 \# T_2 \# ST, LT)) = \\
& \quad (T_1 = T_2 \vee \text{is-refT } T_1 \wedge \text{is-refT } T_2) \\
| \text{app}_i\text{-IfFalse}: \\
& \quad \text{app}_i (\text{IfFalse } b, P, pc, mxs, T_r, (\text{Boolean} \# ST, LT)) = \\
& \quad (0 \leq \text{int } pc + b) \\
| \text{app}_i\text{-Goto}: \\
& \quad \text{app}_i (\text{Goto } b, P, pc, mxs, T_r, s) = \\
& \quad (0 \leq \text{int } pc + b) \\
| \text{app}_i\text{-Return}: \\
& \quad \text{app}_i (\text{Return}, P, pc, mxs, T_r, (T \# ST, LT)) = \\
& \quad (P \vdash T \leq T_r) \\
| \text{app}_i\text{-Throw}: \\
& \quad \text{app}_i (\text{Throw}, P, pc, mxs, T_r, (T \# ST, LT)) = \\
& \quad \text{is-refT } T \\
| \text{app}_i\text{-Invoke}: \\
& \quad \text{app}_i (\text{Invoke } M \ n, P, pc, mxs, T_r, (ST, LT)) = \\
& \quad (n < \text{length } ST \wedge \\
& \quad (ST!n \neq NT \rightarrow \\
& \quad \quad (\exists C \ D \ Ts \ T \ m. ST!n = \text{Class } C \wedge P \vdash C \text{ sees } M, \text{NonStatic}:Ts \rightarrow T = m \text{ in } D \wedge \\
& \quad \quad \quad P \vdash \text{rev } (\text{take } n \ ST) [\leq] Ts))) \\
| \text{app}_i\text{-Invokestatic}: \\
& \quad \text{app}_i (\text{Invokestatic } C \ M \ n, P, pc, mxs, T_r, (ST, LT)) = \\
& \quad (\text{length } ST - n < mxs \wedge n \leq \text{length } ST \wedge M \neq \text{clinit} \wedge \\
& \quad (\exists D \ Ts \ T \ m. P \vdash C \text{ sees } M, \text{Static}:Ts \rightarrow T = m \text{ in } D \wedge \\
& \quad \quad P \vdash \text{rev } (\text{take } n \ ST) [\leq] Ts)) \\
| \text{app}_i\text{-default}: \\
& \quad \text{app}_i (i, P, pc, mxs, T_r, s) = \text{False}
\end{aligned}$$

**definition**  $\text{xcpt-app} :: \text{instr} \Rightarrow 'm \text{ prog} \Rightarrow pc \Rightarrow nat \Rightarrow \text{ex-table} \Rightarrow ty_i \Rightarrow \text{bool}$  **where**  
 $\text{xcpt-app } i \ P \ pc \ mxs \ xt \ \tau \longleftrightarrow (\forall (f, t, C, h, d) \in \text{set } (\text{relevant-entries } P \ i \ pc \ xt). \text{is-class } P \ C \wedge d \leq \text{size } (\text{fst } \tau) \wedge d < mxs)$

**definition**  $\text{app} :: \text{instr} \Rightarrow 'm \text{ prog} \Rightarrow nat \Rightarrow ty \Rightarrow nat \Rightarrow nat \Rightarrow \text{ex-table} \Rightarrow ty_i' \Rightarrow \text{bool}$  **where**  
 $\text{app } i \ P \ mxs \ T_r \ pc \ mpc \ xt \ t = (\text{case } t \text{ of } \text{None} \Rightarrow \text{True} \mid \text{Some } \tau \Rightarrow \\
\text{app}_i (i, P, pc, mxs, T_r, \tau) \wedge \text{xcpt-app } i \ P \ pc \ mxs \ xt \ \tau \wedge \\
(\forall (pc', \tau') \in \text{set } (\text{eff } i \ P \ pc \ xt \ t). pc' < mpc))$

**lemma**  $\text{app-Some}$ :

$$\begin{aligned}
& \text{app } i \ P \ mxs \ T_r \ pc \ mpc \ xt \ (\text{Some } \tau) = \\
& \quad (\text{app}_i (i, P, pc, mxs, T_r, \tau) \wedge \text{xcpt-app } i \ P \ pc \ mxs \ xt \ \tau \wedge \\
& \quad (\forall (pc', s') \in \text{set } (\text{eff } i \ P \ pc \ xt \ (\text{Some } \tau)). pc' < mpc)) \\
& \langle \text{proof} \rangle
\end{aligned}$$

**locale**  $\text{eff} = \text{jvm-method} +$   
**fixes**  $\text{eff}_i$  **and**  $\text{app}_i$  **and**  $\text{eff}$  **and**  $\text{app}$   
**fixes**  $\text{norm-eff}$  **and**  $\text{xcpt-app}$  **and**  $\text{xcpt-eff}$

**fixes**  $\text{mpc}$   
**defines**  $\text{mpc} \equiv \text{size } is$

**defines**  $\text{eff}_i \ i \ \tau \equiv \text{Effect.eff}_i (i, P, \tau)$

**notes**  $eff_i\text{-simps}$  [simp] = Effect.eff<sub>i</sub>.simps [where P = P, folded eff<sub>i</sub>-def]

**defines**  $app_i$  i pc τ ≡ Effect.app<sub>i</sub> (i, P, pc, mxs, T<sub>r</sub>, τ)

**notes**  $app_i\text{-simps}$  [simp] = Effect.app<sub>i</sub>.simps [where P=P and mxs=mxs and T<sub>r</sub>=T<sub>r</sub>, folded app<sub>i</sub>-def]

**defines**  $xcpt\text{-eff}$  i pc τ ≡ Effect.xcpt-eff i P pc τ xt

**notes**  $xcpt\text{-eff}$  = Effect.xcpt-eff-def [of - P - - xt, folded xcpt-eff-def]

**defines**  $norm\text{-eff}$  i pc τ ≡ Effect.norm-eff i P pc τ

**notes**  $norm\text{-eff}$  = Effect.norm-eff-def [of - P, folded norm-eff-def eff<sub>i</sub>-def]

**defines**  $eff$  i pc ≡ Effect.eff i P pc xt

**notes**  $eff$  = Effect.eff-def [of - P - xt, folded eff-def norm-eff-def xcpt-eff-def]

**defines**  $xcpt\text{-app}$  i pc τ ≡ Effect.xcpt-app i P pc mxs xt τ

**notes**  $xcpt\text{-app}$  = Effect.xcpt-app-def [of - P - mxs xt, folded xcpt-app-def]

**defines**  $app$  i pc ≡ Effect.app i P mxs T<sub>r</sub> pc mpc xt

**notes**  $app$  = Effect.app-def [of - P mxs T<sub>r</sub> - mpc xt, folded app-def xcpt-app-def app<sub>i</sub>-def eff-def]

**lemma** *length-cases2*:

**assumes**  $\bigwedge LT. P ([], LT)$

**assumes**  $\bigwedge l ST LT. P (l\#ST, LT)$

**shows** P s

*<proof>*

**lemma** *length-cases3*:

**assumes**  $\bigwedge LT. P ([], LT)$

**assumes**  $\bigwedge l LT. P ([l], LT)$

**assumes**  $\bigwedge l ST LT. P (l\#ST, LT)$

**shows** P s*<proof>*

**lemma** *length-cases4*:

**assumes**  $\bigwedge LT. P ([], LT)$

**assumes**  $\bigwedge l LT. P ([l], LT)$

**assumes**  $\bigwedge l l' LT. P ([l, l'], LT)$

**assumes**  $\bigwedge l l' ST LT. P (l\#l'\#ST, LT)$

**shows** P s*<proof>*

simp rules for *app*

**lemma** *appNone*[simp]:  $app$  i P mxs T<sub>r</sub> pc mpc et None = True

*<proof>*

**lemma** *appLoad*[simp]:

$app_i$  (Load idx, P, T<sub>r</sub>, mxs, pc, s) = (∃ ST LT. s = (ST, LT) ∧ idx < length LT ∧ LT!idx ≠ Err ∧ length ST < mxs)

*<proof>*

**lemma** *appStore*[simp]:

$app_i$  (Store idx, P, pc, mxs, T<sub>r</sub>, s) = (∃ ts ST LT. s = (ts#ST, LT) ∧ idx < length LT)

*<proof>*

**lemma** *appPush[simp]*:

$app_i (Push\ v,\ P,\ pc,\ mxs,\ T_r,\ s) =$   
 $(\exists\ ST\ LT.\ s = (ST, LT) \wedge length\ ST < mxs \wedge typeof\ v \neq None)$   
 ⟨proof⟩

**lemma** *appGetField[simp]*:

$app_i (Getfield\ F\ C,\ P,\ pc,\ mxs,\ T_r,\ s) =$   
 $(\exists\ oT\ vT\ ST\ LT.\ s = (oT\#\!ST,\ LT) \wedge$   
 $P \vdash C\ sees\ F,\ NonStatic:\!vT\ in\ C \wedge P \vdash oT \leq (Class\ C))$   
 ⟨proof⟩

**lemma** *appGetStatic[simp]*:

$app_i (Getstatic\ C\ F\ D,\ P,\ pc,\ mxs,\ T_r,\ s) =$   
 $(\exists\ vT\ ST\ LT.\ s = (ST,\ LT) \wedge length\ ST < mxs \wedge P \vdash C\ sees\ F,\ Static:\!vT\ in\ D)$   
 ⟨proof⟩

**lemma** *appPutField[simp]*:

$app_i (Putfield\ F\ C,\ P,\ pc,\ mxs,\ T_r,\ s) =$   
 $(\exists\ vT\ vT'\ oT\ ST\ LT.\ s = (vT\#\!oT\#\!ST,\ LT) \wedge$   
 $P \vdash C\ sees\ F,\ NonStatic:\!vT'\ in\ C \wedge P \vdash oT \leq (Class\ C) \wedge P \vdash vT \leq vT')$   
 ⟨proof⟩

**lemma** *appPutstatic[simp]*:

$app_i (Putstatic\ C\ F\ D,\ P,\ pc,\ mxs,\ T_r,\ s) =$   
 $(\exists\ vT\ vT'\ ST\ LT.\ s = (vT\#\!ST,\ LT) \wedge$   
 $P \vdash C\ sees\ F,\ Static:\!vT'\ in\ D \wedge P \vdash vT \leq vT')$   
 ⟨proof⟩

**lemma** *appNew[simp]*:

$app_i (New\ C,\ P,\ pc,\ mxs,\ T_r,\ s) =$   
 $(\exists\ ST\ LT.\ s = (ST, LT) \wedge is\ class\ P\ C \wedge length\ ST < mxs)$   
 ⟨proof⟩

**lemma** *appCheckcast[simp]*:

$app_i (Checkcast\ C,\ P,\ pc,\ mxs,\ T_r,\ s) =$   
 $(\exists\ T\ ST\ LT.\ s = (T\#\!ST,\ LT) \wedge is\ class\ P\ C \wedge is\ refT\ T)$   
 ⟨proof⟩

**lemma** *appPop[simp]*:

$app_i (Pop,\ P,\ pc,\ mxs,\ T_r,\ s) = (\exists\ ts\ ST\ LT.\ s = (ts\#\!ST,\ LT))$   
 ⟨proof⟩

**lemma** *appIAdd[simp]*:

$app_i (IAdd,\ P,\ pc,\ mxs,\ T_r,\ s) = (\exists\ ST\ LT.\ s = (Integer\#\!Integer\#\!ST,\ LT))$ ⟨proof⟩

**lemma** *appIfFalse [simp]*:

$app_i (IfFalse\ b,\ P,\ pc,\ mxs,\ T_r,\ s) =$   
 $(\exists\ ST\ LT.\ s = (Boolean\#\!ST,\ LT) \wedge 0 \leq int\ pc + b)$ ⟨proof⟩

**lemma** *appCmpEq[simp]*:

$app_i (CmpEq,\ P,\ pc,\ mxs,\ T_r,\ s) =$   
 $(\exists\ T_1\ T_2\ ST\ LT.\ s = (T_1\#\!T_2\#\!ST,\ LT) \wedge (\neg is\ refT\ T_1 \wedge T_2 = T_1 \vee is\ refT\ T_1 \wedge is\ refT\ T_2))$   
 ⟨proof⟩

**lemma** *appReturn*[simp]:

$app_i (Return, P, pc, mxs, T_r, s) = (\exists T ST LT. s = (T \# ST, LT) \wedge P \vdash T \leq T_r)$   
 ⟨proof⟩

**lemma** *appThrow*[simp]:

$app_i (Throw, P, pc, mxs, T_r, s) = (\exists T ST LT. s = (T \# ST, LT) \wedge is-refT T)$   
 ⟨proof⟩

**lemma** *effNone*:

$(pc', s') \in set (eff\ i\ P\ pc\ et\ None) \implies s' = None$   
 ⟨proof⟩

some helpers to make the specification directly executable:

**lemma** *relevant-entries-append* [simp]:

$relevant-entries\ P\ i\ pc\ (xt\ @\ xt') = relevant-entries\ P\ i\ pc\ xt\ @\ relevant-entries\ P\ i\ pc\ xt'$   
 ⟨proof⟩

**lemma** *xcpt-app-append* [iff]:

$xcpt-app\ i\ P\ pc\ mxs\ (xt@xt')\ \tau = (xcpt-app\ i\ P\ pc\ mxs\ xt\ \tau \wedge xcpt-app\ i\ P\ pc\ mxs\ xt'\ \tau)$   
 ⟨proof⟩

**lemma** *xcpt-eff-append* [simp]:

$xcpt-eff\ i\ P\ pc\ \tau\ (xt@xt') = xcpt-eff\ i\ P\ pc\ \tau\ xt\ @\ xcpt-eff\ i\ P\ pc\ \tau\ xt'$   
 ⟨proof⟩

**lemma** *app-append* [simp]:

$app\ i\ P\ pc\ T\ mxs\ mpc\ (xt@xt')\ \tau = (app\ i\ P\ pc\ T\ mxs\ mpc\ xt\ \tau \wedge app\ i\ P\ pc\ T\ mxs\ mpc\ xt'\ \tau)$   
 ⟨proof⟩

end

## 2.10 Monotonicity of eff and app

theory *EffectMono* imports *Effect* begin

declare *not-Err-eq* [iff]

**lemma** *app<sub>i</sub>-mono*:

**assumes** *wf*: *wf-prog* *p* *P*  
**assumes** *less*:  $P \vdash \tau \leq_i \tau'$   
**shows**  $app_i (i, P, mxs, mpc, rT, \tau) \implies app_i (i, P, mxs, mpc, rT, \tau)$ ⟨proof⟩

**lemma** *succs-mono*:

**assumes** *wf*: *wf-prog* *p* *P* **and** *app<sub>i</sub>*:  $app_i (i, P, mxs, mpc, rT, \tau')$   
**shows**  $P \vdash \tau \leq_i \tau' \implies set (succs\ i\ \tau\ pc) \subseteq set (succs\ i\ \tau'\ pc)$ ⟨proof⟩

**lemma** *app-mono*:

**assumes** *wf*: *wf-prog* *p* *P*  
**assumes** *less'*:  $P \vdash \tau \leq' \tau'$   
**shows**  $app\ i\ P\ m\ rT\ pc\ mpc\ xt\ \tau \implies app\ i\ P\ m\ rT\ pc\ mpc\ xt\ \tau$ ⟨proof⟩

**lemma** *eff<sub>i</sub>-mono*:

**assumes** *wf*: *wf-prog* *p* *P*  
**assumes** *less*:  $P \vdash \tau \leq_i \tau'$   
**assumes** *app<sub>i</sub>*:  $app\ i\ P\ m\ rT\ pc\ mpc\ xt\ (Some\ \tau')$

```

assumes succs: succs i  $\tau$  pc  $\neq []$  succs i  $\tau'$  pc  $\neq []$ 
shows  $P \vdash \text{eff}_i (i, P, \tau) \leq_i \text{eff}_i (i, P, \tau') \langle \text{proof} \rangle$ 
end

```

## 2.11 The Bytecode Verifier

```

theory BVSpec
imports Effect
begin

```

This theory contains a specification of the BV. The specification describes correct typings of method bodies; it corresponds to type *checking*.

### definition

— The method type only contains declared classes:  
 $\text{check-types} :: 'm \text{ prog} \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow \text{ty}_i' \text{ err list} \Rightarrow \text{bool}$

### where

$$\text{check-types } P \text{ } m\text{x}s \text{ } m\text{x}l \text{ } \tau s \equiv \text{set } \tau s \subseteq \text{states } P \text{ } m\text{x}s \text{ } m\text{x}l$$

— An instruction is welltyped if it is applicable and its effect  
 — is compatible with the type at all successor instructions:

### definition

$$\text{wt-instr} :: ['m \text{ prog}, \text{ty}, \text{nat}, \text{pc}, \text{ex-table}, \text{instr}, \text{pc}, \text{ty}_m] \Rightarrow \text{bool}$$

$$\langle \langle -, -, -, -, - \vdash -, - :: \rightarrow [60, 0, 0, 0, 0, 0, 0, 61] \ 60 \rangle \rangle$$

### where

$$P, T, m\text{x}s, m\text{pc}, \text{xt} \vdash i, \text{pc} :: \tau s \equiv$$

$$\text{app } i \ P \ m\text{x}s \ T \ \text{pc} \ m\text{pc} \ \text{xt} \ (\tau s! \text{pc}) \wedge$$

$$(\forall (pc', \tau') \in \text{set} (\text{eff } i \ P \ \text{pc} \ \text{xt} \ (\tau s! \text{pc})). P \vdash \tau' \leq' \tau s! \text{pc}' )$$

— The type at  $\text{pc}=0$  conforms to the method calling convention:

### definition

$$\text{wt-start} :: ['m \text{ prog}, \text{cname}, \text{staticb}, \text{ty list}, \text{nat}, \text{ty}_m] \Rightarrow \text{bool}$$

### where

$$\text{wt-start } P \ C \ b \ T_s \ m\text{x}l_0 \ \tau s \equiv$$

$$\text{case } b \ \text{of } \text{NonStatic} \Rightarrow P \vdash \text{Some } ([], \text{OK } (\text{Class } C) \# \text{map } \text{OK } T_s @ \text{replicate } m\text{x}l_0 \ \text{Err}) \leq' \tau s! 0$$

$$| \text{Static} \Rightarrow P \vdash \text{Some } ([], \text{map } \text{OK } T_s @ \text{replicate } m\text{x}l_0 \ \text{Err}) \leq' \tau s! 0$$

— A method is welltyped if the body is not empty,  
 — if the method type covers all instructions and mentions  
 — declared classes only, if the method calling convention is respected, and  
 — if all instructions are welltyped.

### definition

$$\text{wt-method} :: ['m \text{ prog}, \text{cname}, \text{staticb}, \text{ty list}, \text{ty}, \text{nat}, \text{instr list},$$

$$\text{ex-table}, \text{ty}_m] \Rightarrow \text{bool}$$

### where

$$\text{wt-method } P \ C \ b \ T_s \ T_r \ m\text{x}s \ m\text{x}l_0 \ \text{is} \ \text{xt} \ \tau s \equiv (b = \text{Static} \vee b = \text{NonStatic}) \wedge$$

$$0 < \text{size } \text{is} \wedge \text{size } \tau s = \text{size } \text{is} \wedge$$

$$\text{check-types } P \ m\text{x}s \ ((\text{case } b \ \text{of } \text{Static} \Rightarrow 0 \ | \ \text{NonStatic} \Rightarrow 1) + \text{size } T_s + m\text{x}l_0) \ (\text{map } \text{OK } \tau s) \wedge$$

$$\text{wt-start } P \ C \ b \ T_s \ m\text{x}l_0 \ \tau s \wedge$$

$$(\forall \text{pc} < \text{size } \text{is}. P, T_r, m\text{x}s, \text{size } \text{is}, \text{xt} \vdash \text{is}! \text{pc}, \text{pc} :: \tau s)$$

— A program is welltyped if it is wellformed and all methods are welltyped

### definition

$$\text{wf-jvm-prog-phi} :: \text{ty}_P \Rightarrow \text{jvm-prog} \Rightarrow \text{bool} \ (\langle \text{wf}'\text{-jvm}'\text{-prog} \rangle)$$

### where

$$\text{wf-jvm-prog}_{\Phi} \equiv$$

$$\text{wf-prog } (\lambda P \ C \ (M, b, T_s, T_r, (m\text{x}s, m\text{x}l_0, \text{is}, \text{xt})).$$

*wt-method*  $P C b Ts T_r mxs mxl_0 is xt (\Phi C M)$

**definition** *wf-jvm-prog* :: *jvm-prog*  $\Rightarrow$  *bool*

**where**

*wf-jvm-prog*  $P \equiv \exists \Phi. wf-jvm-prog_{\Phi} P$

**lemma** *wt-jvm-progD*:

*wf-jvm-prog\_{\Phi} P \implies \exists wt. wf-prog wt P \langle proof \rangle*

**lemma** *wt-jvm-prog-impl-wt-instr*:

**assumes** *wf*: *wf-jvm-prog\_{\Phi} P* **and**

*sees*:  $P \vdash C \text{ sees } M, b: Ts \rightarrow T = (mxs, mxl_0, ins, xt)$  in  $C$  **and**

*pc*:  $pc < size\ ins$

**shows**  $P, T, mxs, size\ ins, xt \vdash ins!pc, pc :: \Phi C M \langle proof \rangle$

**lemma** *wt-jvm-prog-impl-wt-start*:

**assumes** *wf*: *wf-jvm-prog\_{\Phi} P* **and**

*sees*:  $P \vdash C \text{ sees } M, b: Ts \rightarrow T = (mxs, mxl_0, ins, xt)$  in  $C$

**shows**  $0 < size\ ins \wedge wt-start\ P\ C\ b\ Ts\ mxl_0 (\Phi C M) \langle proof \rangle$

**lemma** *wf-jvm-prog-nclinit*:

**assumes** *wtp*: *wf-jvm-prog\_{\Phi} P*

**and** *meth*:  $P \vdash C \text{ sees } M, b : Ts \rightarrow T = (mxs, mxl_0, ins, xt)$  in  $D$

**and** *wt*:  $P, T, mxs, size\ ins, xt \vdash ins!pc, pc :: \Phi C M$

**and** *pc*:  $pc < length\ ins$  **and**  $\Phi: \Phi C M ! pc = Some(ST, LT)$

**and** *ins*:  $ins ! pc = Invokestatic\ C_0\ M_0\ n$

**shows**  $M_0 \neq clinit$

$\langle proof \rangle$

**end**

## 2.12 The Typing Framework for the JVM

**theory** *TF-JVM*

**imports** *Jinja.Typing-Framework-err EffectMono BVSpec*

**begin**

**definition** *exec* :: *jvm-prog*  $\Rightarrow$  *nat*  $\Rightarrow$  *ty*  $\Rightarrow$  *ex-table*  $\Rightarrow$  *instr list*  $\Rightarrow$  *ty<sub>i</sub>'* *err step-type*

**where**

*exec*  $G\ mxs\ rT\ et\ bs \equiv$

*err-step* (*size* *bs*) ( $\lambda pc. app\ (bs!pc)\ G\ mxs\ rT\ pc\ (size\ bs)\ et$ )

( $\lambda pc. eff\ (bs!pc)\ G\ pc\ et$ )

**locale** *JVM-sl* =

**fixes**  $P :: jvm-prog$  **and**  $mxs$  **and**  $mxl_0$  **and**  $n$

**fixes**  $b$  **and**  $Ts :: ty\ list$  **and**  $is$  **and**  $xt$  **and**  $T_r$

**fixes**  $mxl$  **and**  $A$  **and**  $r$  **and**  $f$  **and**  $app$  **and**  $eff$  **and**  $step$

**defines** [*simp*]:  $mxl \equiv (case\ b\ of\ Static \Rightarrow 0 \mid NonStatic \Rightarrow 1) + size\ Ts + mxl_0$

**defines** [*simp*]:  $A \equiv states\ P\ mxs\ mxl$

**defines** [*simp*]:  $r \equiv JVM-SemiType.le\ P\ mxs\ mxl$

**defines** [*simp*]:  $f \equiv JVM-SemiType.sup\ P\ mxs\ mxl$

**defines** [*simp*]:  $app \equiv \lambda pc. Effect.app\ (is!pc)\ P\ mxs\ T_r\ pc\ (size\ is)\ xt$

**defines** [*simp*]:  $eff \equiv \lambda pc. Effect.eff\ (is!pc)\ P\ pc\ xt$

**defines** [*simp*]:  $step \equiv err-step\ (size\ is)\ app\ eff$

**defines**  $[simp]$ :  $n \equiv size\ is$   
**assumes**  $staticb$ :  $b = Static \vee b = NonStatic$

**locale**  $start-context = JVM-sl +$   
**fixes**  $p$  **and**  $C$

**assumes**  $wf$ :  $wf-prog\ p\ P$   
**assumes**  $C$ :  $is-class\ P\ C$   
**assumes**  $Ts$ :  $set\ Ts \subseteq types\ P$

**fixes**  $first :: ty_i'$  **and**  $start$

**defines**  $[simp]$ :

$first \equiv Some\ (\[], (case\ b\ of\ Static \Rightarrow [] \mid NonStatic \Rightarrow [OK\ (Class\ C)]) @\ map\ OK\ Ts @\ replicate\ mxl_0\ Err)$

**defines**  $[simp]$ :

$start \equiv (OK\ first) \# replicate\ (size\ is - 1)\ (OK\ None)$

**thm**  $start-context.intro$

**lemma**  $start-context-intro-auxi$ :

**fixes**  $P\ b\ Ts\ p\ C$

**assumes**  $b = Static \vee b = NonStatic$

**and**  $wf-prog\ p\ P$

**and**  $is-class\ P\ C$

**and**  $set\ Ts \subseteq types\ P$

**shows**  $start-context\ P\ b\ Ts\ p\ C$

$\langle proof \rangle$

## 2.12.1 Connecting JVM and Framework

**lemma** (**in**  $start-context$ )  $semi$ :  $semilat\ (A, r, f)$

$\langle proof \rangle$

**lemma** (**in**  $JVM-sl$ )  $step-def-exec$ :  $step \equiv exec\ P\ mxs\ T_r\ at\ is$

$\langle proof \rangle$

**lemma**  $special-ex-swap-lemma$   $[iff]$ :

$(? X. (? n. X = A\ n \ \&\ P\ n) \ \&\ Q\ X) = (? n. Q(A\ n) \ \&\ P\ n)$

$\langle proof \rangle$

**lemma**  $ex-in-list$   $[iff]$ :

$(\exists n. ST \in nlists\ n\ A \wedge n \leq mxs) = (set\ ST \subseteq A \wedge size\ ST \leq mxs)$

$\langle proof \rangle$

**lemma**  $singleton-nlists$ :

$(\exists n. [Class\ C] \in nlists\ n\ (types\ P) \wedge n \leq mxs) = (is-class\ P\ C \wedge 0 < mxs)$

$\langle proof \rangle$

**lemma**  $set-drop-subset$ :

$set\ xs \subseteq A \implies set\ (drop\ n\ xs) \subseteq A$

$\langle proof \rangle$



**lemma** *Suc-minus-minus-le*:  
 $n < mxs \implies \text{Suc } (n - (n - b)) \leq mxs$   
 ⟨proof⟩

**lemma** *in-nlistsE*:  
 $\llbracket xs \in \text{nlists } n \ A; \llbracket \text{size } xs = n; \text{ set } xs \subseteq A \rrbracket \implies P \rrbracket \implies P$   
 ⟨proof⟩

**declare** *is-relevant-entry-def* [simp]  
**declare** *set-drop-subset* [simp]

**theorem** (in *start-context*) *exec-pres-type*:  
 $\text{pres-type step (size is) } A \langle \text{proof} \rangle$   
**declare** *is-relevant-entry-def* [simp del]  
**declare** *set-drop-subset* [simp del]

**lemma** *lesubstep-type-simple*:  
 $xs \llbracket \sqsubseteq_{\text{Product.le } (=) \ r} \rrbracket ys \implies \text{set } xs \llbracket \sqsubseteq_r \rrbracket \text{ set } ys \langle \text{proof} \rangle$   
**declare** *is-relevant-entry-def* [simp del]

**lemma** *conjI2*:  $\llbracket A; A \implies B \rrbracket \implies A \wedge B \langle \text{proof} \rangle$

**lemma** (in *JVM-sl*) *eff-mono*:  
**assumes** *wf*:  $wf\text{-prog } p \ P$  **and** *pc* < *length is* **and**  
*lesub*:  $s \llbracket \sqsubseteq_{\text{sup-state-opt } P} \rrbracket t$  **and** *app*:  $\text{app } pc \ t$   
**shows**  $\text{set } (\text{eff } pc \ s) \llbracket \sqsubseteq_{\text{sup-state-opt } P} \rrbracket \text{ set } (\text{eff } pc \ t) \langle \text{proof} \rangle$   
**lemma** (in *JVM-sl*) *bounded-step*:  $\text{bounded step (size is)} \langle \text{proof} \rangle$   
**theorem** (in *JVM-sl*) *step-mono*:  
 $wf\text{-prog } wf\text{-mb } P \implies \text{mono } r \ \text{step (size is) } A \langle \text{proof} \rangle$

**lemma** (in *start-context*) *first-in-A* [iff]:  $OK \ \text{first} \in A$   
 ⟨proof⟩

**lemma** (in *JVM-sl*) *wt-method-def2*:  
 $\text{wt-method } P \ C' \ b \ Ts \ T_r \ mxs \ mxl_0 \ is \ xt \ \tau s =$   
 $(is \neq [] \wedge$   
 $(b = \text{Static} \vee b = \text{NonStatic}) \wedge$   
 $\text{size } \tau s = \text{size } is \wedge$   
 $OK \ ' \ \text{set } \tau s \subseteq \text{states } P \ mxs \ mxl \wedge$   
 $\text{wt-start } P \ C' \ b \ Ts \ mxl_0 \ \tau s \wedge$   
 $\text{wt-app-eff } (\text{sup-state-opt } P) \ \text{app } \text{eff } \tau s) \langle \text{proof} \rangle$

end

## 2.13 Kildall for the JVM

**theory** *BVExec*  
**imports** *Jinja.Abstract-BV TF-JVM Jinja.Typing-Framework-2*  
**begin**

**definition** *kiljvm* ::  $jvm\text{-prog} \Rightarrow nat \Rightarrow nat \Rightarrow ty \Rightarrow$



## 2.14 LBV for the JVM

**theory** *LBVJVM*

**imports** *Jinja.Abstract-BV TF-JVM*

**begin**

**type-synonym** *prog-cert* = *cname*  $\Rightarrow$  *mname*  $\Rightarrow$  *ty<sub>i</sub>' err list*

**definition** *check-cert* :: *jvm-prog*  $\Rightarrow$  *nat*  $\Rightarrow$  *nat*  $\Rightarrow$  *nat*  $\Rightarrow$  *ty<sub>i</sub>' err list*  $\Rightarrow$  *bool*

**where**

*check-cert* *P m<sub>x</sub>s m<sub>x</sub>l n cert*  $\equiv$  *check-types* *P m<sub>x</sub>s m<sub>x</sub>l cert*  $\wedge$  *size cert* = *n+1*  $\wedge$   
 $(\forall i < n. \text{cert}!i \neq \text{Err}) \wedge \text{cert}!n = \text{OK None}$

**definition** *lbv<sub>jvm</sub>* :: *jvm-prog*  $\Rightarrow$  *nat*  $\Rightarrow$  *nat*  $\Rightarrow$  *ty*  $\Rightarrow$  *ex-table*  $\Rightarrow$

*ty<sub>i</sub>' err list*  $\Rightarrow$  *instr list*  $\Rightarrow$  *ty<sub>i</sub>' err*  $\Rightarrow$  *ty<sub>i</sub>' err*

**where**

*lbv<sub>jvm</sub>* *P m<sub>x</sub>s m<sub>a</sub>x<sub>r</sub> T<sub>r</sub> et cert bs*  $\equiv$

*wtl-inst-list* *bs cert* (*JVM-SemiType.sup* *P m<sub>x</sub>s m<sub>a</sub>x<sub>r</sub>*) (*JVM-SemiType.le* *P m<sub>x</sub>s m<sub>a</sub>x<sub>r</sub>*) *Err* (*OK None*) (*exec* *P m<sub>x</sub>s T<sub>r</sub> et bs*) *0*

**definition** *wt-lbv* :: *jvm-prog*  $\Rightarrow$  *cname*  $\Rightarrow$  *staticb*  $\Rightarrow$  *ty list*  $\Rightarrow$  *ty*  $\Rightarrow$  *nat*  $\Rightarrow$  *nat*  $\Rightarrow$

*ex-table*  $\Rightarrow$  *ty<sub>i</sub>' err list*  $\Rightarrow$  *instr list*  $\Rightarrow$  *bool*

**where**

*wt-lbv* *P C b Ts T<sub>r</sub> m<sub>x</sub>s m<sub>x</sub>l<sub>0</sub> et cert ins*  $\equiv$  (*b* = *Static*  $\vee$  *b* = *NonStatic*)  $\wedge$

*check-cert* *P m<sub>x</sub>s* ((*case b of Static*  $\Rightarrow$  *0* | *NonStatic*  $\Rightarrow$  *1*) + *size Ts + m<sub>x</sub>l<sub>0</sub>*) (*size ins*) *cert*  $\wedge$   
 $0 < \text{size ins}$   $\wedge$

(*let start* = *Some* ( $\square$ , (*case b of Static*  $\Rightarrow$   $\square$  | *NonStatic*  $\Rightarrow$  [*OK* (*Class C*)])  
 $\text{@}((\text{map OK } Ts))\text{@}(\text{replicate } m_{x}l_0 \text{ Err});$

*result* = *lbv<sub>jvm</sub>* *P m<sub>x</sub>s* ((*case b of Static*  $\Rightarrow$  *0* | *NonStatic*  $\Rightarrow$  *1*) + *size Ts + m<sub>x</sub>l<sub>0</sub>*) *T<sub>r</sub> et cert ins*  
(*OK start*)

*in result*  $\neq$  *Err*)

**definition** *wt-jvm-prog-lbv* :: *jvm-prog*  $\Rightarrow$  *prog-cert*  $\Rightarrow$  *bool*

**where**

*wt-jvm-prog-lbv* *P cert*  $\equiv$

*wf-prog* ( $\lambda P C (mn, b, Ts, T_r, (m_{x}s, m_{x}l_0, ins, et)). \text{wt-lbv } P C b Ts T_r m_{x}s m_{x}l_0 et (cert C mn) ins$ ) *P*

**definition** *mk-cert* :: *jvm-prog*  $\Rightarrow$  *nat*  $\Rightarrow$  *ty*  $\Rightarrow$  *ex-table*  $\Rightarrow$  *instr list*

$\Rightarrow$  *ty<sub>m</sub>*  $\Rightarrow$  *ty<sub>i</sub>' err list*

**where**

*mk-cert* *P m<sub>x</sub>s T<sub>r</sub> et bs phi*  $\equiv$  *make-cert* (*exec* *P m<sub>x</sub>s T<sub>r</sub> et bs*) (*map* *OK phi*) (*OK None*)

**definition** *prg-cert* :: *jvm-prog*  $\Rightarrow$  *ty<sub>P</sub>*  $\Rightarrow$  *prog-cert*

**where**

*prg-cert* *P phi C mn*  $\equiv$  *let* (*C, b, Ts, T<sub>r</sub>, (m<sub>x</sub>s, m<sub>x</sub>l<sub>0</sub>, ins, et)*) = *method* *P C mn*  
*in* *mk-cert* *P m<sub>x</sub>s T<sub>r</sub> et ins* (*phi C mn*)

**lemma** *check-certD* [*intro?*]:

*check-cert* *P m<sub>x</sub>s m<sub>x</sub>l n cert*  $\implies$  *cert-ok* *cert n Err* (*OK None*) (*states* *P m<sub>x</sub>s m<sub>x</sub>l*)  
(*proof*)

**lemma** (**in** *start-context*) *wt-lbv-wt-step*:

**assumes** *lbv*: *wt-lbv* *P C b Ts T<sub>r</sub> m<sub>x</sub>s m<sub>x</sub>l<sub>0</sub> xt cert is*

**shows**  $\exists \tau s \in nlists$  (size is)  $A$ . *wt-step*  $r$  *Err step*  $\tau s \wedge OK$  first  $\sqsubseteq_r \tau s!0$ ⟨proof⟩

**lemma** (in *start-context*) *wt-lbv-wt-method*:

**assumes** *lbv*: *wt-lbv*  $P$   $C$   $b$   $Ts$   $T_r$   $m\&s$   $m\&l_0$  *xt cert is*

**shows**  $\exists \tau s$ . *wt-method*  $P$   $C$   $b$   $Ts$   $T_r$   $m\&s$   $m\&l_0$  *is xt*  $\tau s$ ⟨proof⟩

**lemma** (in *start-context*) *wt-method-wt-lbv*:

**assumes** *wt*: *wt-method*  $P$   $C$   $b$   $Ts$   $T_r$   $m\&s$   $m\&l_0$  *is xt*  $\tau s$

**defines** [*simp*]: *cert*  $\equiv$  *mk-cert*  $P$   $m\&s$   $T_r$  *xt is*  $\tau s$

**shows** *wt-lbv*  $P$   $C$   $b$   $Ts$   $T_r$   $m\&s$   $m\&l_0$  *xt cert is*⟨proof⟩

**theorem** *jvm-lbv-correct*:

*wt-jvm-prog-lbv*  $P$  *Cert*  $\implies$  *wf-jvm-prog*  $P$ ⟨proof⟩

**theorem** *jvm-lbv-complete*:

**assumes** *wt*: *wf-jvm-prog* $\Phi$   $P$

**shows** *wt-jvm-prog-lbv*  $P$  (*prg-cert*  $P$   $\Phi$ )⟨proof⟩

**end**

## 2.15 BV Type Safety Invariant

**theory** *BVConform*

**imports** *BVSpec* *../JVM/JVMEExec* *../Common/Conform*

**begin**

### 2.15.1 correct-state definitions

**definition** *confT*  $:: 'c$  *prog*  $\Rightarrow$  *heap*  $\Rightarrow$  *val*  $\Rightarrow$  *ty* *err*  $\Rightarrow$  *bool*

( $\langle -, - \vdash - : \leq_{\top} \rightarrow [51, 51, 51, 51] 50$ )

**where**

$P, h \vdash v : \leq_{\top} E \equiv$  *case*  $E$  *of* *Err*  $\Rightarrow$  *True* | *OK*  $T \Rightarrow P, h \vdash v : \leq T$

**notation** (*ASCII*)

*confT* ( $\langle -, - \mid - - : \leq = T \rightarrow [51, 51, 51, 51] 50$ )

**abbreviation**

*confTs*  $:: 'c$  *prog*  $\Rightarrow$  *heap*  $\Rightarrow$  *val list*  $\Rightarrow$  *ty<sub>l</sub>*  $\Rightarrow$  *bool*

( $\langle -, - \vdash - [: \leq_{\top}] \rightarrow [51, 51, 51, 51] 50$ ) **where**

$P, h \vdash vs [: \leq_{\top}] Ts \equiv$  *list-all2* (*confT*  $P$   $h$ ) *vs*  $Ts$

**notation** (*ASCII*)

*confTs* ( $\langle -, - \mid - - [: \leq = T] \rightarrow [51, 51, 51, 51] 50$ )

**fun** *Called-context*  $::$  *jvm-prog*  $\Rightarrow$  *cname*  $\Rightarrow$  *instr*  $\Rightarrow$  *bool* **where**

*Called-context*  $P$   $C_0$  (*New*  $C'$ ) = ( $C_0 = C'$ ) |

*Called-context*  $P$   $C_0$  (*Getstatic*  $C$   $F$   $D$ ) = ( $(C_0 = D) \wedge (\exists t. P \vdash C$  *has*  $F, Static:t$  *in*  $D)$ ) |

*Called-context*  $P$   $C_0$  (*Putstatic*  $C$   $F$   $D$ ) = ( $(C_0 = D) \wedge (\exists t. P \vdash C$  *has*  $F, Static:t$  *in*  $D)$ ) |

*Called-context*  $P$   $C_0$  (*Invokestatic*  $C$   $M$   $n$ )

=  $(\exists Ts$   $T$   $m$   $D. (C_0 = D) \wedge P \vdash C$  *sees*  $M, Static:Ts \rightarrow T = m$  *in*  $D)$  |

*Called-context*  $P$   $- =$  *False*

**abbreviation** *Called-set*  $::$  *instr set* **where**

*Called-set*  $\equiv$   $\{i. \exists C. i =$  *New*  $C\} \cup \{i. \exists C$   $M$   $n. i =$  *Invokestatic*  $C$   $M$   $n\}$

$$\cup \{i. \exists C F D. i = \text{Getstatic } C F D\} \cup \{i. \exists C F D. i = \text{Putstatic } C F D\}$$

**lemma** *Called-context-Called-set*:

*Called-context*  $P D i \implies i \in \text{Called-set}$   $\langle \text{proof} \rangle$

**fun** *valid-ics* ::  $\text{jvm-prog} \Rightarrow \text{heap} \Rightarrow \text{sheap} \Rightarrow \text{cname} \times \text{mname} \times \text{pc} \times \text{init-call-status} \Rightarrow \text{bool}$

$(\langle -, -, \vdash_i \rightarrow [51, 51, 51, 51] 50 \rangle)$  **where**  
*valid-ics*  $P h sh (C, M, pc, \text{Calling } C' Cs)$   
 $= (\text{let } ins = \text{instrs-of } P C M \text{ in } \text{Called-context } P (\text{last } (C' \# Cs)) (ins!pc)$   
 $\wedge \text{is-class } P C') \mid$   
*valid-ics*  $P h sh (C, M, pc, \text{Throwing } Cs a)$   
 $= (\text{let } ins = \text{instrs-of } P C M \text{ in } \exists C1. \text{Called-context } P C1 (ins!pc)$   
 $\wedge (\exists obj. h a = \text{Some } obj)) \mid$   
*valid-ics*  $P h sh (C, M, pc, \text{Called } Cs)$   
 $= (\text{let } ins = \text{instrs-of } P C M$   
 $\text{in } \exists C1 \text{ subj. } \text{Called-context } P C1 (ins!pc) \wedge sh C1 = \text{Some } subj) \mid$   
*valid-ics*  $P - - - = \text{True}$

**definition** *conf-f* ::  $\text{jvm-prog} \Rightarrow \text{heap} \Rightarrow \text{sheap} \Rightarrow \text{ty}_i \Rightarrow \text{bytecode} \Rightarrow \text{frame} \Rightarrow \text{bool}$

**where**

*conf-f*  $P h sh \equiv \lambda(ST, LT) \text{ is } (stk, loc, C, M, pc, ics).$   
 $P, h \vdash stk [\leq] ST \wedge P, h \vdash loc [\leq_{\top}] LT \wedge pc < \text{size } is \wedge P, h, sh \vdash_i (C, M, pc, ics)$

**lemma** *conf-f-def2*:

*conf-f*  $P h sh (ST, LT) \text{ is } (stk, loc, C, M, pc, ics) \equiv$   
 $P, h \vdash stk [\leq] ST \wedge P, h \vdash loc [\leq_{\top}] LT \wedge pc < \text{size } is \wedge P, h, sh \vdash_i (C, M, pc, ics)$   
 $\langle \text{proof} \rangle$

**primrec** *conf-fs* ::  $[\text{jvm-prog}, \text{heap}, \text{sheap}, \text{ty}_P, \text{cname}, \text{mname}, \text{nat}, \text{ty}, \text{frame list}] \Rightarrow \text{bool}$

**where**

*conf-fs*  $P h sh \Phi C_0 M_0 n_0 T_0 [] = \text{True}$   
 $\mid \text{conf-fs } P h sh \Phi C_0 M_0 n_0 T_0 (f \# \text{frs}) =$   
 $(\text{let } (stk, loc, C, M, pc, ics) = f \text{ in}$   
 $(\exists ST LT b Ts T mxs mxl_0 is xt.$   
 $\Phi C M ! pc = \text{Some } (ST, LT) \wedge$   
 $(P \vdash C \text{ sees } M, b:Ts \rightarrow T = (mxs, mxl_0, is, xt) \text{ in } C) \wedge$   
 $((\exists D Ts' T' m D'. M_0 \neq \text{clinit} \wedge ics = \text{No-ics} \wedge$   
 $is!pc = \text{Invoke } M_0 n_0 \wedge ST!n_0 = \text{Class } D \wedge$   
 $P \vdash D \text{ sees } M_0, \text{NonStatic}:Ts' \rightarrow T' = m \text{ in } D' \wedge P \vdash C_0 \leq^* D' \wedge P \vdash T_0 \leq T') \vee$   
 $(\exists D Ts' T' m. M_0 \neq \text{clinit} \wedge ics = \text{No-ics} \wedge$   
 $is!pc = \text{Invokestatic } D M_0 n_0 \wedge$   
 $P \vdash D \text{ sees } M_0, \text{Static}:Ts' \rightarrow T' = m \text{ in } C_0 \wedge P \vdash T_0 \leq T') \vee$   
 $(M_0 = \text{clinit} \wedge (\exists Cs. ics = \text{Called } Cs))) \wedge$   
 $\text{conf-f } P h sh (ST, LT) \text{ is } f \wedge \text{conf-fs } P h sh \Phi C M (\text{size } Ts) T \text{ frs}))$

**fun** *ics-classes* ::  $\text{init-call-status} \Rightarrow \text{cname list}$  **where**

*ics-classes*  $(\text{Calling } C Cs) = Cs \mid$   
*ics-classes*  $(\text{Throwing } Cs a) = Cs \mid$   
*ics-classes*  $(\text{Called } Cs) = Cs \mid$   
*ics-classes*  $- = []$

**fun** *frame-clinit-classes* ::  $\text{frame} \Rightarrow \text{cname list}$  **where**

*frame-clinit-classes*  $(stk, loc, C, M, pc, ics) = (\text{if } M = \text{clinit} \text{ then } [C] \text{ else } []) @ \text{ics-classes } ics$

**abbreviation** *clinit-classes* :: *frame list*  $\Rightarrow$  *cname list* **where**  
*clinit-classes frs*  $\equiv$  *concat* (*map frame-clinit-classes frs*)

**definition** *distinct-clinit* :: *frame list*  $\Rightarrow$  *bool* **where**  
*distinct-clinit frs*  $\equiv$  *distinct* (*clinit-classes frs*)

**definition** *conf-clinit* :: *jvm-prog*  $\Rightarrow$  *sheap*  $\Rightarrow$  *frame list*  $\Rightarrow$  *bool* **where**  
*conf-clinit P sh frs*  
 $\equiv$  *distinct-clinit frs*  $\wedge$   
 $(\forall C \in \text{set}(\text{clinit-classes frs}). \text{is-class } P \ C \wedge (\exists \text{sfs. sh } C = \text{Some}(\text{sfs}, \text{Processing})))$

**definition** *correct-state* :: [*jvm-prog*, *ty<sub>P</sub>*, *jvm-state*]  $\Rightarrow$  *bool* ( $\langle \cdot, \cdot \vdash \cdot \sqrt{\cdot} \rangle$  [61,0,0] 61)  
**where**

*correct-state P  $\Phi$*   $\equiv$   $\lambda(xp, h, frs, sh).$   
*case xp of*  
*None*  $\Rightarrow$  (*case frs of*  
 $\square \Rightarrow \text{True}$   
 $| (f \# fs) \Rightarrow P \vdash h \sqrt{\cdot} \wedge P, h \vdash_s sh \sqrt{\cdot} \wedge \text{conf-clinit } P \ sh \ frs \wedge$   
 $(\text{let } (stk, loc, C, M, pc, ics) = f$   
 $\text{in } \exists b \ Ts \ T \ mxs \ mxl_0 \ is \ xt \ \tau.$   
 $(P \vdash C \ \text{sees } M, b: Ts \rightarrow T = (mxs, mxl_0, is, xt) \text{ in } C) \wedge$   
 $\Phi \ C \ M \ ! \ pc = \text{Some } \tau \wedge$   
 $\text{conf-f } P \ h \ sh \ \tau \ \text{is } f \wedge \text{conf-fs } P \ h \ sh \ \Phi \ C \ M \ (\text{size } Ts) \ T \ fs))$   
 $| \text{Some } x \Rightarrow frs = \square$

**notation**

*correct-state* ( $\langle \cdot, \cdot \mid \cdot \text{ - } [ok] \rangle$  [61,0,0] 61)

## 2.15.2 Values and $\top$

**lemma** *confT-Err* [*iff*]:  $P, h \vdash x : \leq_{\top} \text{Err}$   
*<proof>*

**lemma** *confT-OK* [*iff*]:  $P, h \vdash x : \leq_{\top} \text{OK } T = (P, h \vdash x : \leq T)$   
*<proof>*

**lemma** *confT-cases*:

$P, h \vdash x : \leq_{\top} X = (X = \text{Err} \vee (\exists T. X = \text{OK } T \wedge P, h \vdash x : \leq T))$   
*<proof>*

**lemma** *confT-hext* [*intro?*, *trans*]:

$\llbracket P, h \vdash x : \leq_{\top} T; h \sqsubseteq h' \rrbracket \Longrightarrow P, h' \vdash x : \leq_{\top} T$   
*<proof>*

**lemma** *confT-widen* [*intro?*, *trans*]:

$\llbracket P, h \vdash x : \leq_{\top} T; P \vdash T \leq_{\top} T' \rrbracket \Longrightarrow P, h \vdash x : \leq_{\top} T'$   
*<proof>*

## 2.15.3 Stack and Registers

**lemmas** *confTs-Cons1* [*iff*] = *list-all2-Cons1* [*of confT P h*] **for**  $P \ h$

**lemma** *confTs-confT-sup*:

**assumes** *confTs*:  $P, h \vdash \text{loc} [\leq_{\top}] LT$  **and**  $n < \text{size } LT$  **and**

$LTn$ :  $LT!n = OK T$  **and** *subtype*:  $P \vdash T \leq T'$

**shows**  $P, h \vdash (\text{loc}!n) :\leq T'$  *(proof)*

**lemma** *confTs-heat* [*intro?*]:

$P, h \vdash \text{loc} [\leq_{\top}] LT \implies h \trianglelefteq h' \implies P, h' \vdash \text{loc} [\leq_{\top}] LT$

*(proof)*

**lemma** *confTs-widen* [*intro?*, *trans*]:

$P, h \vdash \text{loc} [\leq_{\top}] LT \implies P \vdash LT [\leq_{\top}] LT' \implies P, h \vdash \text{loc} [\leq_{\top}] LT'$

*(proof)*

**lemma** *confTs-map* [*iff*]:

$\bigwedge \text{vs. } (P, h \vdash \text{vs} [\leq_{\top}] \text{map } OK Ts) = (P, h \vdash \text{vs} [\leq] Ts)$

*(proof)*

**lemma** *reg-widen-Err* [*iff*]:

$\bigwedge LT. (P \vdash \text{replicate } n \text{ Err} [\leq_{\top}] LT) = (LT = \text{replicate } n \text{ Err})$

*(proof)*

**lemma** *confTs-Err* [*iff*]:

$P, h \vdash \text{replicate } n v [\leq_{\top}] \text{replicate } n \text{ Err}$

*(proof)*

#### 2.15.4 valid init-call-status

**lemma** *valid-ics-shupd*:

**assumes**  $P, h, sh \vdash_i (C, M, pc, ics)$  **and** *distinct*  $(C' \# \text{ics-classes } ics)$

**shows**  $P, h, sh(C' \mapsto (sfs, i')) \vdash_i (C, M, pc, ics)$

*(proof)*

#### 2.15.5 correct-frame

**lemma** *conf-f-Throwing*:

**assumes** *conf-f*  $P h sh (ST, LT)$  *is*  $(stk, loc, C, M, pc, \text{Called } Cs)$

**and** *is-class*  $P C'$  **and**  $h \text{ xcp} = \text{Some obj}$  **and**  $sh C' = \text{Some}(sfs, \text{Processing})$

**shows** *conf-f*  $P h sh (ST, LT)$  *is*  $(stk, loc, C, M, pc, \text{Throwing } (C' \# Cs) \text{ xcp})$

*(proof)*

**lemma** *conf-f-shupd*:

**assumes** *conf-f*  $P h sh (ST, LT)$  *ins*  $f$

**and**  $i = \text{Processing}$

$\vee (\text{distinct } (C \# \text{ics-classes } (ics\text{-of } f)) \wedge (\text{curr-method } f = \text{clinit} \longrightarrow C \neq \text{curr-class } f))$

**shows** *conf-f*  $P h (sh(C \mapsto (sfs, i))) (ST, LT)$  *ins*  $f$

*(proof)*

**lemma** *conf-f-shupd'*:

**assumes** *conf-f*  $P h sh (ST, LT)$  *ins*  $f$

**and**  $sh C = \text{Some}(sfs, i)$

**shows** *conf-f*  $P h (sh(C \mapsto (sfs', i))) (ST, LT)$  *ins*  $f$

*(proof)*

#### 2.15.6 correct-frames

**lemmas** [*simp del*] = *fun-upd-apply*

**lemma** *conf-fs-heat*:

$\bigwedge C M n T_r.$   
 $\llbracket \text{conf-fs } P h sh \Phi C M n T_r \text{ frs}; h \leq h' \rrbracket \implies \text{conf-fs } P h' sh \Phi C M n T_r \text{ frs} \langle \text{proof} \rangle$

**lemma** *conf-fs-shupd*:

**assumes** *conf-fs*  $P h sh \Phi C_0 M n T \text{ frs}$   
**and** *dist*: *distinct* ( $C \# \text{clinit-classes frs}$ )  
**shows** *conf-fs*  $P h (sh(C \mapsto (sfs, i))) \Phi C_0 M n T \text{ frs}$   
 $\langle \text{proof} \rangle$

**lemma** *conf-fs-shupd'*:

**assumes** *conf-fs*  $P h sh \Phi C_0 M n T \text{ frs}$   
**and** *shC*:  $sh C = \text{Some}(sfs, i)$   
**shows** *conf-fs*  $P h (sh(C \mapsto (sfs', i))) \Phi C_0 M n T \text{ frs}$   
 $\langle \text{proof} \rangle$

## 2.15.7 correctness wrt *clinit* use

**lemma** *conf-clinit-Cons*:

**assumes** *conf-clinit*  $P sh (f \# \text{frs})$   
**shows** *conf-clinit*  $P sh \text{ frs}$   
 $\langle \text{proof} \rangle$

**lemma** *conf-clinit-Cons-Cons*:

*conf-clinit*  $P sh (f' \# f \# \text{frs}) \implies \text{conf-clinit } P sh (f' \# \text{frs})$   
 $\langle \text{proof} \rangle$

**lemma** *conf-clinit-diff*:

**assumes** *conf-clinit*  $P sh ((stk, loc, C, M, pc, ics) \# \text{frs})$   
**shows** *conf-clinit*  $P sh ((stk', loc', C, M, pc', ics) \# \text{frs})$   
 $\langle \text{proof} \rangle$

**lemma** *conf-clinit-diff'*:

**assumes** *conf-clinit*  $P sh ((stk, loc, C, M, pc, ics) \# \text{frs})$   
**shows** *conf-clinit*  $P sh ((stk', loc', C, M, pc', \text{No-ics}) \# \text{frs})$   
 $\langle \text{proof} \rangle$

**lemma** *conf-clinit-Called-Throwing*:

*conf-clinit*  $P sh ((stk', loc', C', \text{clinit}, pc', ics') \# (stk, loc, C, M, pc, \text{Called } Cs) \# \text{fs})$   
 $\implies \text{conf-clinit } P sh ((stk, loc, C, M, pc, \text{Throwing } (C' \# Cs) \text{ xcp}) \# \text{fs})$   
 $\langle \text{proof} \rangle$

**lemma** *conf-clinit-Throwing*:

*conf-clinit*  $P sh ((stk, loc, C, M, pc, \text{Throwing } (C' \# Cs) \text{ xcp}) \# \text{fs})$   
 $\implies \text{conf-clinit } P sh ((stk, loc, C, M, pc, \text{Throwing } Cs \text{ xcp}) \# \text{fs})$   
 $\langle \text{proof} \rangle$

**lemma** *conf-clinit-Called*:

$\llbracket \text{conf-clinit } P sh ((stk, loc, C, M, pc, \text{Called } (C' \# Cs)) \# \text{frs});$   
 $P \vdash C' \text{ sees } \text{clinit}, \text{Static}: \square \rightarrow \text{Void} = (m\text{xs}', m\text{xl}', \text{ins}', \text{xt}') \text{ in } C' \rrbracket$   
 $\implies \text{conf-clinit } P sh (\text{create-init-frame } P C' \# (stk, loc, C, M, pc, \text{Called } Cs) \# \text{frs})$   
 $\langle \text{proof} \rangle$



**lemma** *conf-clinit-Cons-nclinit*:

**assumes** *conf-clinit*  $P$  *sh frs* **and** *nclinit*:  $M \neq \text{clinit}$   
**shows** *conf-clinit*  $P$  *sh*  $((\text{stk}, \text{loc}, C, M, \text{pc}, \text{No-ics}) \# \text{frs})$   
 $\langle \text{proof} \rangle$

**lemma** *conf-clinit-Invoke*:

**assumes** *conf-clinit*  $P$  *sh*  $((\text{stk}, \text{loc}, C, M, \text{pc}, \text{ics}) \# \text{frs})$  **and**  $M' \neq \text{clinit}$   
**shows** *conf-clinit*  $P$  *sh*  $((\text{stk}', \text{loc}', C', M', \text{pc}', \text{No-ics}) \# (\text{stk}, \text{loc}, C, M, \text{pc}, \text{No-ics}) \# \text{frs})$   
 $\langle \text{proof} \rangle$

**lemma** *conf-clinit-nProc-dist*:

**assumes** *conf-clinit*  $P$  *sh frs*  
**and**  $\forall \text{sfs}. \text{sh } C \neq \text{Some}(\text{sfs}, \text{Processing})$   
**shows** *distinct*  $(C \# \text{clinit-classes } \text{frs})$   
 $\langle \text{proof} \rangle$

**lemma** *conf-clinit-shupd*:

**assumes** *conf-clinit*  $P$  *sh frs*  
**and** *dist*: *distinct*  $(C \# \text{clinit-classes } \text{frs})$   
**shows** *conf-clinit*  $P$   $(\text{sh}(C \mapsto (\text{sfs}, i))) \text{ frs}$   
 $\langle \text{proof} \rangle$

**lemma** *conf-clinit-shupd'*:

**assumes** *conf-clinit*  $P$  *sh frs*  
**and** *sh*  $C = \text{Some}(\text{sfs}, i)$   
**shows** *conf-clinit*  $P$   $(\text{sh}(C \mapsto (\text{sfs}', i))) \text{ frs}$   
 $\langle \text{proof} \rangle$

**lemma** *conf-clinit-shupd-Called*:

**assumes** *conf-clinit*  $P$  *sh*  $((\text{stk}, \text{loc}, C, M, \text{pc}, \text{Calling } C' \text{ } Cs) \# \text{frs})$   
**and** *dist*: *distinct*  $(C' \# \text{clinit-classes } ((\text{stk}, \text{loc}, C, M, \text{pc}, \text{Calling } C' \text{ } Cs) \# \text{frs}))$   
**and** *cls*: *is-class*  $P \text{ } C'$   
**shows** *conf-clinit*  $P$   $(\text{sh}(C' \mapsto (\text{sfs}, \text{Processing}))) ((\text{stk}, \text{loc}, C, M, \text{pc}, \text{Called } (C' \# Cs)) \# \text{frs})$   
 $\langle \text{proof} \rangle$

**lemma** *conf-clinit-shupd-Calling*:

**assumes** *conf-clinit*  $P$  *sh*  $((\text{stk}, \text{loc}, C, M, \text{pc}, \text{Calling } C' \text{ } Cs) \# \text{frs})$   
**and** *dist*: *distinct*  $(C' \# \text{clinit-classes } ((\text{stk}, \text{loc}, C, M, \text{pc}, \text{Calling } C' \text{ } Cs) \# \text{frs}))$   
**and** *cls*: *is-class*  $P \text{ } C'$   
**shows** *conf-clinit*  $P$   $(\text{sh}(C' \mapsto (\text{sfs}, \text{Processing})))$   
 $((\text{stk}, \text{loc}, C, M, \text{pc}, \text{Calling } (\text{fst}(\text{the}(\text{class } P \text{ } C')))) (C' \# Cs)) \# \text{frs})$   
 $\langle \text{proof} \rangle$

### 2.15.8 correct state

**lemma** *correct-state-Cons*:

**assumes** *cr*:  $P, \Phi \mid - (xp, h, f \# \text{frs}, sh) \text{ [ok]}$   
**shows**  $P, \Phi \mid - (xp, h, \text{frs}, sh) \text{ [ok]}$   
 $\langle \text{proof} \rangle$

**lemma** *correct-state-shupd*:

**assumes** *cs*:  $P, \Phi \mid - (xp, h, \text{frs}, sh) \text{ [ok]}$  **and** *shC*: *sh*  $C = \text{Some}(\text{sfs}, i)$   
**and** *dist*: *distinct*  $(C \# \text{clinit-classes } \text{frs})$

**shows**  $P, \Phi \vdash (xp, h, frs, sh(C \mapsto (sfs, i'))) [ok]$   
 $\langle proof \rangle$

**lemma** *correct-state-Throwing-ex*:

**assumes** *correct*:  $P, \Phi \vdash (xp, h, (stk, loc, C, M, pc, ics) \# frs, sh) \checkmark$   
**shows**  $\bigwedge Cs a. ics = Throwing\ Cs\ a \implies \exists obj. h\ a = Some\ obj$   
 $\langle proof \rangle$

**end**

## 2.16 Property preservation under *class-add*

**theory** *ClassAdd*

**imports** *BVConform*

**begin**

**lemma** *err-mono*:  $A \subseteq B \implies err\ A \subseteq err\ B$   
 $\langle proof \rangle$

**lemma** *opt-mono*:  $A \subseteq B \implies opt\ A \subseteq opt\ B$   
 $\langle proof \rangle$

**abbreviation** *class-add* :: *jvm-prog*  $\Rightarrow$  *jvm-method cdecl*  $\Rightarrow$  *jvm-prog* **where**  
*class-add*  $P\ cd \equiv cd \# P$

### 2.16.1 Fields

**lemma** *class-add-has-fields*:

**assumes** *fs*:  $P \vdash D\ has\ fields\ FDTs$  **and** *nc*:  $\neg is\ class\ P\ C$   
**shows** *class-add*  $P\ (C, cdec) \vdash D\ has\ fields\ FDTs$   
 $\langle proof \rangle$

**lemma** *class-add-has-fields-rev*:

$\llbracket class\ add\ P\ (C, cdec) \vdash D\ has\ fields\ FDTs; \neg P \vdash D \preceq^* C \rrbracket$   
 $\implies P \vdash D\ has\ fields\ FDTs$   
 $\langle proof \rangle$

**lemma** *class-add-has-field*:

**assumes**  $P \vdash C_0\ has\ F, b: T\ in\ D$  **and**  $\neg is\ class\ P\ C$   
**shows** *class-add*  $P\ (C, cdec) \vdash C_0\ has\ F, b: T\ in\ D$   
 $\langle proof \rangle$

**lemma** *class-add-has-field-rev*:

**assumes** *has*: *class-add*  $P\ (C, cdec) \vdash C_0\ has\ F, b: T\ in\ D$   
**and** *nep*:  $\bigwedge D'. P \vdash C_0 \preceq^* D' \implies D' \neq C$   
**shows**  $P \vdash C_0\ has\ F, b: T\ in\ D$   
 $\langle proof \rangle$

**lemma** *class-add-sees-field*:

**assumes**  $P \vdash C_0\ sees\ F, b: T\ in\ D$  **and**  $\neg is\ class\ P\ C$   
**shows** *class-add*  $P\ (C, cdec) \vdash C_0\ sees\ F, b: T\ in\ D$   
 $\langle proof \rangle$

**lemma** *class-add-sees-field-rev*:

**assumes** *has*:  $\text{class-add } P (C, \text{cdec}) \vdash C_0 \text{ sees } F, b:T \text{ in } D$   
**and** *ncp*:  $\bigwedge D'. P \vdash C_0 \preceq^* D' \implies D' \neq C$   
**shows**  $P \vdash C_0 \text{ sees } F, b:T \text{ in } D$   
 $\langle \text{proof} \rangle$

**lemma** *class-add-field*:  
**assumes** *fd*:  $P \vdash C_0 \text{ sees } F, b:T \text{ in } D$  **and**  $\neg \text{is-class } P C$   
**shows**  $\text{field } P C_0 F = \text{field } (\text{class-add } P (C, \text{cdec})) C_0 F$   
 $\langle \text{proof} \rangle$

## 2.16.2 Methods

**lemma** *class-add-sees-methods*:  
**assumes** *ms*:  $P \vdash D \text{ sees-methods } Mm$  **and** *nc*:  $\neg \text{is-class } P C$   
**shows**  $\text{class-add } P (C, \text{cdec}) \vdash D \text{ sees-methods } Mm$   
 $\langle \text{proof} \rangle$

**lemma** *class-add-sees-methods-rev*:  
 $\llbracket \text{class-add } P (C, \text{cdec}) \vdash D \text{ sees-methods } Mm;$   
 $\bigwedge D'. P \vdash D \preceq^* D' \implies D' \neq C \rrbracket$   
 $\implies P \vdash D \text{ sees-methods } Mm$   
 $\langle \text{proof} \rangle$

**lemma** *class-add-sees-methods-Obj*:  
**assumes**  $P \vdash \text{Object sees-methods } Mm$  **and** *nObj*:  $C \neq \text{Object}$   
**shows**  $\text{class-add } P (C, \text{cdec}) \vdash \text{Object sees-methods } Mm$   
 $\langle \text{proof} \rangle$

**lemma** *class-add-sees-methods-rev-Obj*:  
**assumes**  $\text{class-add } P (C, \text{cdec}) \vdash \text{Object sees-methods } Mm$  **and** *nObj*:  $C \neq \text{Object}$   
**shows**  $P \vdash \text{Object sees-methods } Mm$   
 $\langle \text{proof} \rangle$

**lemma** *class-add-sees-method*:  
**assumes**  $P \vdash C_0 \text{ sees } M_0, b : Ts \rightarrow T = m \text{ in } D$  **and**  $\neg \text{is-class } P C$   
**shows**  $\text{class-add } P (C, \text{cdec}) \vdash C_0 \text{ sees } M_0, b : Ts \rightarrow T = m \text{ in } D$   
 $\langle \text{proof} \rangle$

**lemma** *class-add-method*:  
**assumes** *md*:  $P \vdash C_0 \text{ sees } M_0, b : Ts \rightarrow T = m \text{ in } D$  **and**  $\neg \text{is-class } P C$   
**shows**  $\text{method } P C_0 M_0 = \text{method } (\text{class-add } P (C, \text{cdec})) C_0 M_0$   
 $\langle \text{proof} \rangle$

**lemma** *class-add-sees-method-rev*:  
 $\llbracket \text{class-add } P (C, \text{cdec}) \vdash C_0 \text{ sees } M_0, b : Ts \rightarrow T = m \text{ in } D;$   
 $\neg P \vdash C_0 \preceq^* C \rrbracket$   
 $\implies P \vdash C_0 \text{ sees } M_0, b : Ts \rightarrow T = m \text{ in } D$   
 $\langle \text{proof} \rangle$

**lemma** *class-add-sees-method-Obj*:  
 $\llbracket P \vdash \text{Object sees } M_0, b : Ts \rightarrow T = m \text{ in } D; C \neq \text{Object} \rrbracket$   
 $\implies \text{class-add } P (C, \text{cdec}) \vdash \text{Object sees } M_0, b : Ts \rightarrow T = m \text{ in } D$   
 $\langle \text{proof} \rangle$

**lemma** *class-add-sees-method-rev-Obj*:

$$\begin{aligned} & \llbracket \text{class-add } P (C, \text{cdec}) \vdash \text{Object sees } M_0, b : Ts \rightarrow T = m \text{ in } D; C \neq \text{Object} \rrbracket \\ & \implies P \vdash \text{Object sees } M_0, b : Ts \rightarrow T = m \text{ in } D \\ & \langle \text{proof} \rangle \end{aligned}$$

### 2.16.3 Types and states

**lemma** *class-add-is-type*:

$$\begin{aligned} & \text{is-type } P T \implies \text{is-type } (\text{class-add } P (C, \text{cdec})) T \\ & \langle \text{proof} \rangle \end{aligned}$$

**lemma** *class-add-types*:

$$\begin{aligned} & \text{types } P \subseteq \text{types } (\text{class-add } P (C, \text{cdec})) \\ & \langle \text{proof} \rangle \end{aligned}$$

**lemma** *class-add-states*:

$$\begin{aligned} & \text{states } P \text{ mxs } \text{m}\lambda l \subseteq \text{states } (\text{class-add } P (C, \text{cdec})) \text{ mxs } \text{m}\lambda l \\ & \langle \text{proof} \rangle \end{aligned}$$

**lemma** *class-add-check-types*:

$$\begin{aligned} & \text{check-types } P \text{ mxs } \text{m}\lambda l \tau s \implies \text{check-types } (\text{class-add } P (C, \text{cdec})) \text{ mxs } \text{m}\lambda l \tau s \\ & \langle \text{proof} \rangle \end{aligned}$$

### 2.16.4 Subclasses and subtypes

**lemma** *class-add-subcls*:

$$\begin{aligned} & \llbracket P \vdash D \preceq^* D'; \neg \text{is-class } P C \rrbracket \\ & \implies \text{class-add } P (C, \text{cdec}) \vdash D \preceq^* D' \\ & \langle \text{proof} \rangle \end{aligned}$$

**lemma** *class-add-subcls-rev*:

$$\begin{aligned} & \llbracket \text{class-add } P (C, \text{cdec}) \vdash D \preceq^* D'; \neg P \vdash D \preceq^* C \rrbracket \\ & \implies P \vdash D \preceq^* D' \\ & \langle \text{proof} \rangle \end{aligned}$$

**lemma** *class-add-subtype*:

$$\begin{aligned} & \llbracket \text{subtype } P x y; \neg \text{is-class } P C \rrbracket \\ & \implies \text{subtype } (\text{class-add } P (C, \text{cdec})) x y \\ & \langle \text{proof} \rangle \end{aligned}$$

**lemma** *class-add-widens*:

$$\begin{aligned} & \llbracket P \vdash Ts [\leq] Ts'; \neg \text{is-class } P C \rrbracket \\ & \implies (\text{class-add } P (C, \text{cdec})) \vdash Ts [\leq] Ts' \\ & \langle \text{proof} \rangle \end{aligned}$$

**lemma** *class-add-sup-ty-opt*:

$$\begin{aligned} & \llbracket P \vdash l1 \leq_{\top} l2; \neg \text{is-class } P C \rrbracket \\ & \implies \text{class-add } P (C, \text{cdec}) \vdash l1 \leq_{\top} l2 \\ & \langle \text{proof} \rangle \end{aligned}$$

**lemma** *class-add-sup-loc*:

$$\begin{aligned} & \llbracket P \vdash LT [\leq_{\top}] LT'; \neg \text{is-class } P C \rrbracket \\ & \implies \text{class-add } P (C, \text{cdec}) \vdash LT [\leq_{\top}] LT' \\ & \langle \text{proof} \rangle \end{aligned}$$

**lemma** *class-add-sup-state*:

$\llbracket P \vdash \tau \leq_i \tau'; \neg \text{is-class } P \ C \rrbracket$   
 $\implies \text{class-add } P \ (C, \text{cdec}) \vdash \tau \leq_i \tau'$   
 ⟨proof⟩

**lemma** *class-add-sup-state-opt*:

$\llbracket P \vdash \tau \leq' \tau'; \neg \text{is-class } P \ C \rrbracket$   
 $\implies \text{class-add } P \ (C, \text{cdec}) \vdash \tau \leq' \tau'$   
 ⟨proof⟩

### 2.16.5 Effect

**lemma** *class-add-is-relevant-class*:

$\llbracket \text{is-relevant-class } i \ P \ C_0; \neg \text{is-class } P \ C \rrbracket$   
 $\implies \text{is-relevant-class } i \ (\text{class-add } P \ (C, \text{cdec})) \ C_0$   
 ⟨proof⟩

**lemma** *class-add-is-relevant-class-rev*:

**assumes** *irc*:  $\text{is-relevant-class } i \ (\text{class-add } P \ (C, \text{cdec})) \ C_0$   
**and** *nep*:  $\bigwedge cd \ D'. \ cd \in \text{set } P \implies \neg P \vdash \text{fst } cd \preceq^* C$   
**and** *wfcp*:  $\text{wf-syscls } P$   
**shows**  $\text{is-relevant-class } i \ P \ C_0$   
 ⟨proof⟩

**lemma** *class-add-is-relevant-entry*:

$\llbracket \text{is-relevant-entry } P \ i \ pc \ e; \neg \text{is-class } P \ C \rrbracket$   
 $\implies \text{is-relevant-entry } (\text{class-add } P \ (C, \text{cdec})) \ i \ pc \ e$   
 ⟨proof⟩

**lemma** *class-add-is-relevant-entry-rev*:

$\llbracket \text{is-relevant-entry } (\text{class-add } P \ (C, \text{cdec})) \ i \ pc \ e;$   
 $\bigwedge cd \ D'. \ cd \in \text{set } P \implies \neg P \vdash \text{fst } cd \preceq^* C;$   
 $\text{wf-syscls } P \rrbracket$   
 $\implies \text{is-relevant-entry } P \ i \ pc \ e$   
 ⟨proof⟩

**lemma** *class-add-relevant-entries*:

$\neg \text{is-class } P \ C$   
 $\implies \text{set } (\text{relevant-entries } P \ i \ pc \ xt) \subseteq \text{set } (\text{relevant-entries } (\text{class-add } P \ (C, \text{cdec})) \ i \ pc \ xt)$   
 ⟨proof⟩

**lemma** *class-add-relevant-entries-eq*:

**assumes** *wf*:  $\text{wf-prog } wf\text{-md } P$  **and** *nclass*:  $\neg \text{is-class } P \ C$   
**shows**  $\text{relevant-entries } P \ i \ pc \ xt = \text{relevant-entries } (\text{class-add } P \ (C, \text{cdec})) \ i \ pc \ xt$   
 ⟨proof⟩

**lemma** *class-add-norm-eff-pc*:

**assumes** *ne*:  $\forall (pc', \tau') \in \text{set } (\text{norm-eff } i \ P \ pc \ \tau). \ pc' < mpc$   
**shows**  $\forall (pc', \tau') \in \text{set } (\text{norm-eff } i \ (\text{class-add } P \ (C, \text{cdec})) \ pc \ \tau). \ pc' < mpc$   
 ⟨proof⟩

**lemma** *class-add-norm-eff-sup-state-opt*:

**assumes** *ne*:  $\forall (pc', \tau') \in \text{set } (\text{norm-eff } i \ P \ pc \ \tau). \ P \vdash \tau' \leq' \tau s!pc'$

**and**  $nclass: \neg is-class P C$  **and**  $app: app_i (i, P, pc, mxs, T, \tau)$   
**shows**  $\forall (pc', \tau') \in set (norm-eff i (class-add P (C, cdec)) pc \tau). (class-add P (C, cdec)) \vdash \tau' \leq' \tau s!pc'$   
 $\langle proof \rangle$

**lemma** *class-add-xcpt-eff-eq*:

**assumes**  $wf: wf-prog wf-md P$  **and**  $nclass: \neg is-class P C$   
**shows**  $xcpt-eff i P pc \tau xt = xcpt-eff i (class-add P (C, cdec)) pc \tau xt$   
 $\langle proof \rangle$

**lemma** *class-add-eff-pc*:

**assumes**  $eff: \forall (pc', \tau') \in set (eff i P pc xt (Some \tau)). pc' < mpc$   
**and**  $wf: wf-prog wf-md P$  **and**  $nclass: \neg is-class P C$   
**shows**  $\forall (pc', \tau') \in set (eff i (class-add P (C, cdec)) pc xt (Some \tau)). pc' < mpc$   
 $\langle proof \rangle$

**lemma** *class-add-eff-sup-state-opt*:

**assumes**  $eff: \forall (pc', \tau') \in set (eff i P pc xt (Some \tau)). P \vdash \tau' \leq' \tau s!pc'$   
**and**  $wf: wf-prog wf-md P$  **and**  $nclass: \neg is-class P C$   
**and**  $app: app_i (i, P, pc, mxs, T, \tau)$   
**shows**  $\forall (pc', \tau') \in set (eff i (class-add P (C, cdec)) pc xt (Some \tau)).$   
 $(class-add P (C, cdec)) \vdash \tau' \leq' \tau s!pc'$   
 $\langle proof \rangle$

**lemma** *class-add-app\_i*:

**assumes**  $app_i (i, P, pc, mxs, T_r, ST, LT)$  **and**  $\neg is-class P C$   
**shows**  $app_i (i, class-add P (C, cdec), pc, mxs, T_r, ST, LT)$   
 $\langle proof \rangle$

**lemma** *class-add-xcpt-app*:

**assumes**  $xa: xcpt-app i P pc mxs xt \tau$   
**and**  $wf: wf-prog wf-md P$  **and**  $nclass: \neg is-class P C$   
**shows**  $xcpt-app i (class-add P (C, cdec)) pc mxs xt \tau$   
 $\langle proof \rangle$

**lemma** *class-add-app*:

**assumes**  $app: app i P mxs T pc mpc xt t$   
**and**  $wf: wf-prog wf-md P$  **and**  $nclass: \neg is-class P C$   
**shows**  $app i (class-add P (C, cdec)) mxs T pc mpc xt t$   
 $\langle proof \rangle$

## 2.16.6 Well-formedness and well-typedness

**lemma** *class-add-wf-mdecl*:

$\llbracket wf-mdecl wf-md P C_0 md;$   
 $\bigwedge C_0 md. wf-md P C_0 md \implies wf-md (class-add P (C, cdec)) C_0 md \rrbracket$   
 $\implies wf-mdecl wf-md (class-add P (C, cdec)) C_0 md$   
 $\langle proof \rangle$

**lemma** *class-add-wf-mdecl'*:

**assumes**  $wfd: wf-mdecl wf-md P C_0 md$   
**and**  $ms: (C_0, S, fs, ms) \in set P$  **and**  $md: md \in set ms$   
**and**  $wf-md': \bigwedge C_0 S fs ms m. \llbracket (C_0, S, fs, ms) \in set P; m \in set ms \rrbracket \implies wf-md' (class-add P (C, cdec))$   
 $C_0 m$

**shows**  $wf\text{-mdecl } wf\text{-md}' (class\text{-add } P (C, cdec)) C_0 md$   
 $\langle proof \rangle$

**lemma** *class-add-wf-cdecl*:

**assumes**  $wfcd: wf\text{-decl } wf\text{-md } P cd$  **and**  $cdP: cd \in set P$   
**and**  $nep: \neg P \vdash fst cd \preceq^* C$  **and**  $dist: distinct\text{-fst } P$   
**and**  $wfmd: \bigwedge C_0 md. wf\text{-md } P C_0 md \implies wf\text{-md } (class\text{-add } P (C, cdec)) C_0 md$   
**and**  $nclass: \neg is\text{-class } P C$   
**shows**  $wf\text{-cdecl } wf\text{-md } (class\text{-add } P (C, cdec)) cd$   
 $\langle proof \rangle$

**lemma** *class-add-wf-cdecl'*:

**assumes**  $wfcd: wf\text{-cdecl } wf\text{-md } P cd$  **and**  $cdP: cd \in set P$   
**and**  $nep: \neg P \vdash fst cd \preceq^* C$  **and**  $dist: distinct\text{-fst } P$   
**and**  $wfmd: \bigwedge C_0 S fs ms m. [(C_0, S, fs, ms) \in set P; m \in set ms] \implies wf\text{-md}' (class\text{-add } P (C, cdec)) C_0 m$   
**and**  $nclass: \neg is\text{-class } P C$   
**shows**  $wf\text{-cdecl } wf\text{-md}' (class\text{-add } P (C, cdec)) cd$   
 $\langle proof \rangle$

**lemma** *class-add-wt-start*:

$[[ wt\text{-start } P C_0 b Ts mxl \tau s; \neg is\text{-class } P C ]]$   
 $\implies wt\text{-start } (class\text{-add } P (C, cdec)) C_0 b Ts mxl \tau s$   
 $\langle proof \rangle$

**lemma** *class-add-wt-instr*:

**assumes**  $wti: P, T, mxs, mpc, xt \vdash i, pc :: \tau s$   
**and**  $wf: wf\text{-prog } wf\text{-md } P$  **and**  $nclass: \neg is\text{-class } P C$   
**shows**  $class\text{-add } P (C, cdec), T, mxs, mpc, xt \vdash i, pc :: \tau s$   
 $\langle proof \rangle$

**lemma** *class-add-wt-method*:

**assumes**  $wtm: wt\text{-method } P C_0 b Ts T_r mxs mxl_0 is xt (\Phi C_0 M_0)$   
**and**  $wf: wf\text{-prog } wf\text{-md } P$  **and**  $nclass: \neg is\text{-class } P C$   
**shows**  $wt\text{-method } (class\text{-add } P (C, cdec)) C_0 b Ts T_r mxs mxl_0 is xt (\Phi C_0 M_0)$   
 $\langle proof \rangle$

**lemma** *class-add-wt-method'*:

$[[ (\lambda P C (M, b, Ts, T_r, (mxs, mxl_0, is, xt)). wt\text{-method } P C b Ts T_r mxs mxl_0 is xt (\Phi C M)) P C_0 md;$   
 $wf\text{-prog } wf\text{-md } P; \neg is\text{-class } P C ]]$   
 $\implies (\lambda P C (M, b, Ts, T_r, (mxs, mxl_0, is, xt)). wt\text{-method } P C b Ts T_r mxs mxl_0 is xt (\Phi C M))$   
 $(class\text{-add } P (C, cdec)) C_0 md$   
 $\langle proof \rangle$

## 2.16.7 distinct-fst

**lemma** *class-add-distinct-fst*:

$[[ distinct\text{-fst } P; \neg is\text{-class } P C ]]$   
 $\implies distinct\text{-fst } (class\text{-add } P (C, cdec))$   
 $\langle proof \rangle$

## 2.16.8 Conformance

**lemma** *class-add-conf*:

$\llbracket P, h \vdash v : \leq T; \neg \text{is-class } P \ C \rrbracket$   
 $\implies \text{class-add } P \ (C, \text{cdec}), h \vdash v : \leq T$   
 $\langle \text{proof} \rangle$

**lemma** *class-add-oconf*:

**fixes** *obj::obj*

**assumes** *oc*:  $P, h \vdash \text{obj} \checkmark$  **and** *ns*:  $\neg \text{is-class } P \ C$   
**and** *nep*:  $\bigwedge D'. P \vdash \text{fst}(\text{obj}) \preceq^* D' \implies D' \neq C$

**shows**  $(\text{class-add } P \ (C, \text{cdec})), h \vdash \text{obj} \checkmark$   
 $\langle \text{proof} \rangle$

**lemma** *class-add-soconf*:

**assumes** *soc*:  $P, h, C_0 \vdash_s \text{sfs} \checkmark$  **and** *ns*:  $\neg \text{is-class } P \ C$   
**and** *nep*:  $\bigwedge D'. P \vdash C_0 \preceq^* D' \implies D' \neq C$

**shows**  $(\text{class-add } P \ (C, \text{cdec})), h, C_0 \vdash_s \text{sfs} \checkmark$   
 $\langle \text{proof} \rangle$

**lemma** *class-add-hconf*:

**assumes**  $P \vdash h \checkmark$  **and**  $\neg \text{is-class } P \ C$

**and**  $\bigwedge a \text{ obj } D'. h \ a = \text{Some } \text{obj} \implies P \vdash \text{fst}(\text{obj}) \preceq^* D' \implies D' \neq C$

**shows**  $\text{class-add } P \ (C, \text{cdec}) \vdash h \checkmark$   
 $\langle \text{proof} \rangle$

**lemma** *class-add-hconf-wf*:

**assumes** *wf*: *wf-prog wf-md*  $P$  **and**  $P \vdash h \checkmark$  **and**  $\neg \text{is-class } P \ C$

**and**  $\bigwedge a \text{ obj}. h \ a = \text{Some } \text{obj} \implies \text{fst}(\text{obj}) \neq C$

**shows**  $\text{class-add } P \ (C, \text{cdec}) \vdash h \checkmark$   
 $\langle \text{proof} \rangle$

**lemma** *class-add-shconf*:

**assumes**  $P, h \vdash_s \text{sh} \checkmark$  **and** *ns*:  $\neg \text{is-class } P \ C$

**and**  $\bigwedge C \text{ sobj } D'. \text{sh } C = \text{Some } \text{sobj} \implies P \vdash C \preceq^* D' \implies D' \neq C$

**shows**  $\text{class-add } P \ (C, \text{cdec}), h \vdash_s \text{sh} \checkmark$   
 $\langle \text{proof} \rangle$

**lemma** *class-add-shconf-wf*:

**assumes** *wf*: *wf-prog wf-md*  $P$  **and**  $P, h \vdash_s \text{sh} \checkmark$  **and**  $\neg \text{is-class } P \ C$

**and**  $\bigwedge C \text{ sobj}. \text{sh } C = \text{Some } \text{sobj} \implies C \neq C$

**shows**  $\text{class-add } P \ (C, \text{cdec}), h \vdash_s \text{sh} \checkmark$   
 $\langle \text{proof} \rangle$

**end**

## 2.17 Properties and types of the starting program

**theory** *StartProg*

**imports** *ClassAdd*

**begin**

**lemmas** *wt-defs* = *correct-state-def conf-f-def wt-instr-def eff-def norm-eff-def app-def xcpt-app-def*

**declare** *wt-defs* [*simp*] — removed from *simp* at the end of file



**declare** *start-class-def* [simp]

### 2.17.1 Types

**abbreviation** *start- $\varphi_m$*  :: *ty<sub>m</sub>* **where**  
*start- $\varphi_m$*   $\equiv$  [Some([],[]),Some([Void],[])]

**fun**  $\Phi$ -*start* :: *ty<sub>P</sub>*  $\Rightarrow$  *ty<sub>P</sub>* **where**  
 $\Phi$ -*start*  $\Phi$  *C M* = (if *C=Start*  $\wedge$  (*M=start-m*  $\vee$  *M=clinit*) then *start- $\varphi_m$*  else  $\Phi$  *C M*)

**lemma**  $\Phi$ -*start*:  $\bigwedge C. C \neq \text{Start} \implies \Phi\text{-start } \Phi C = \Phi C$   
 $\Phi$ -*start*  $\Phi$  *Start start-m* = *start- $\varphi_m$*   $\Phi$ -*start*  $\Phi$  *Start clinit* = *start- $\varphi_m$*   
 ⟨proof⟩

**lemma** *check-types- $\varphi_m$* : *check-types* (*start-prog P C M*) 1 0 (map OK *start- $\varphi_m$* )  
 ⟨proof⟩

### 2.17.2 Some simple properties

**lemma** *preallocated-start-state*: *start-state P* =  $\sigma \implies$  *preallocated* (fst(snd  $\sigma$ ))  
 ⟨proof⟩

**lemma** *start-prog-Start-super*: *start-prog P C M*  $\vdash$  *Start*  $\prec^1$  *Object*  
 ⟨proof⟩

**lemma** *start-prog-Start-fields*:  
*start-prog P C M*  $\vdash$  *Start has-fields FDTs*  $\implies$  *map-of FDTs* (*F*, *Start*) = None  
 ⟨proof⟩

**lemma** *start-prog-Start-soconf*:  
 (*start-prog P C M*),*h*,*Start*  $\vdash_s$  *Map.empty*  $\checkmark$   
 ⟨proof⟩

**lemma** *start-prog-start-shconf*:  
*start-prog P C M*,*start-heap P*  $\vdash_s$  *start-sheap*  $\checkmark$ ⟨proof⟩

### 2.17.3 Well-typed and well-formed

**lemma** *start-wt-method*:  
**assumes** *P*  $\vdash$  *C* sees *M*, *Static* :  $\square \rightarrow \text{Void} = m$  in *D* **and** *M*  $\neq$  *clinit* **and**  $\neg$  *is-class P Start*  
**shows** *wt-method* (*start-prog P C M*) *Start Static*  $\square$  *Void 1 0* [*Invokestatic C M 0*, *Return*]  $\square$  *start- $\varphi_m$*   
 (**is wt-method** ?*P* ?*C* ?*b* ?*Ts* ?*T<sub>r</sub>* ?*m<sub>xs</sub>* ?*m<sub>xl</sub>* ?*is* ?*xt* ? *$\tau$ s*)  
 ⟨proof⟩

**lemma** *start-clinit-wt-method*:  
**assumes** *P*  $\vdash$  *C* sees *M*, *Static* :  $\square \rightarrow \text{Void} = m$  in *D* **and** *M*  $\neq$  *clinit* **and**  $\neg$  *is-class P Start*  
**shows** *wt-method* (*start-prog P C M*) *Start Static*  $\square$  *Void 1 0* [*Push Unit*,*Return*]  $\square$  *start- $\varphi_m$*   
 (**is wt-method** ?*P* ?*C* ?*b* ?*Ts* ?*T<sub>r</sub>* ?*m<sub>xs</sub>* ?*m<sub>xl</sub>* ?*is* ?*xt* ? *$\tau$ s*)  
 ⟨proof⟩

**lemma** *start-class-wf*:  
**assumes** *P*  $\vdash$  *C* sees *M*, *Static* :  $\square \rightarrow \text{Void} = m$  in *D*  
**and** *M*  $\neq$  *clinit* **and**  $\neg$  *is-class P Start*  
**and**  $\Phi$  *Start start-m* = *start- $\varphi_m$*  **and**  $\Phi$  *Start clinit* = *start- $\varphi_m$*   
**and** *is-class P Object*

**and**  $\bigwedge b' Ts' T' m' D'. P \vdash \text{Object sees start-}m, b' : Ts' \rightarrow T' = m' \text{ in } D'$   
 $\implies b' = \text{Static} \wedge Ts' = [] \wedge T' = \text{Void}$   
**and**  $\bigwedge b' Ts' T' m' D'. P \vdash \text{Object sees clinit}, b' : Ts' \rightarrow T' = m' \text{ in } D'$   
 $\implies b' = \text{Static} \wedge Ts' = [] \wedge T' = \text{Void}$   
**shows**  $\text{wf-cdecl } (\lambda P C (M, b, Ts, T_r, (m_x s, m_x l_0, is, xt)). \text{wt-method } P C b Ts T_r m_x s m_x l_0 \text{ is } xt (\Phi C M))$   
 $(\text{start-prog } P C M) (\text{start-class } C M)$   
 $\langle \text{proof} \rangle$

**lemma** *start-prog-wf-jvm-prog-phi*:  
**assumes**  $\text{wtp} : \text{wf-jvm-prog}_{\Phi} P$   
**and**  $\text{nstart} : \neg \text{is-class } P \text{ Start}$   
**and**  $\text{meth} : P \vdash C \text{ sees } M, \text{Static} : [] \rightarrow \text{Void} = m \text{ in } D$  **and**  $\text{nclinit} : M \neq \text{clinit}$   
**and**  $\Phi : \bigwedge C. C \neq \text{Start} \implies \Phi' C = \Phi C$   
**and**  $\Phi' : \Phi' \text{ Start start-}m = \text{start-}\varphi_m \Phi' \text{ Start clinit} = \text{start-}\varphi_m$   
**and**  $\text{Obj-start-}m : \bigwedge b' Ts' T' m' D'. P \vdash \text{Object sees start-}m, b' : Ts' \rightarrow T' = m' \text{ in } D'$   
 $\implies b' = \text{Static} \wedge Ts' = [] \wedge T' = \text{Void}$   
**shows**  $\text{wf-jvm-prog}_{\Phi'} (\text{start-prog } P C M)$   
 $\langle \text{proof} \rangle$

**lemma** *start-prog-wf-jvm-prog*:  
**assumes**  $\text{wf} : \text{wf-jvm-prog } P$   
**and**  $\text{nstart} : \neg \text{is-class } P \text{ Start}$   
**and**  $\text{meth} : P \vdash C \text{ sees } M, \text{Static} : [] \rightarrow \text{Void} = m \text{ in } D$  **and**  $\text{nclinit} : M \neq \text{clinit}$   
**and**  $\text{Obj-start-}m : \bigwedge b' Ts' T' m' D'. P \vdash \text{Object sees start-}m, b' : Ts' \rightarrow T' = m' \text{ in } D'$   
 $\implies b' = \text{Static} \wedge Ts' = [] \wedge T' = \text{Void}$   
**shows**  $\text{wf-jvm-prog } (\text{start-prog } P C M)$   
 $\langle \text{proof} \rangle$

## 2.17.4 Methods and instructions

**lemma** *start-prog-Start-sees-methods*:  
 $P \vdash \text{Object sees-methods } Mm$   
 $\implies \text{start-prog } P C M \vdash$   
 $\text{Start sees-methods } Mm ++ (\text{map-option } (\lambda m. (m, \text{Start})) \circ \text{map-of } [\text{start-method } C M, \text{start-clinit}])$   
 $\langle \text{proof} \rangle$

**lemma** *start-prog-Start-sees-start-method*:  
 $P \vdash \text{Object sees-methods } Mm$   
 $\implies \text{start-prog } P C M \vdash$   
 $\text{Start sees start-}m, \text{Static} : [] \rightarrow \text{Void} = (1, 0, [\text{Invokestatic } C M 0, \text{Return}], []) \text{ in } \text{Start}$   
 $\langle \text{proof} \rangle$

**lemma** *wf-start-prog-Start-sees-start-method*:  
**assumes**  $\text{wf} : \text{wf-prog } \text{wf-md } P$   
**shows**  $\text{start-prog } P C M \vdash$   
 $\text{Start sees start-}m, \text{Static} : [] \rightarrow \text{Void} = (1, 0, [\text{Invokestatic } C M 0, \text{Return}], []) \text{ in } \text{Start}$   
 $\langle \text{proof} \rangle$

**lemma** *start-prog-start-m-instrs*:  
**assumes**  $\text{wf} : \text{wf-prog } \text{wf-md } P$   
**shows**  $(\text{instrs-of } (\text{start-prog } P C M) \text{ Start start-}m) = [\text{Invokestatic } C M 0, \text{Return}]$   
 $\langle \text{proof} \rangle$

**declare** *wt-defs* [*simp del*]

**end**

## 2.18 BV Type Safety Proof

**theory** *BVSpecTypeSafe*

**imports** *BVConform StartProg*

**begin**

This theory contains proof that the specification of the bytecode verifier only admits type safe programs.

### 2.18.1 Preliminaries

Simp and intro setup for the type safety proof:

**lemmas** *defs1* = *correct-state-def conf-f-def wt-instr-def eff-def norm-eff-def app-def xcpt-app-def*

**lemmas** *widen-rules* [*intro*] = *conf-widen confT-widen confs-widens confTs-widen*

### 2.18.2 Exception Handling

For the *Invoke* instruction the BV has checked all handlers that guard the current *pc*.

**lemma** *Invoke-handlers*:

$$\begin{aligned} & \text{match-ex-table } P \ C \ pc \ xt = \text{Some } (pc', d') \implies \\ & \exists (f, t, D, h, d) \in \text{set } (\text{relevant-entries } P \ (\text{Invoke } n \ M) \ pc \ xt). \\ & P \vdash C \preceq^* D \wedge pc \in \{f..<t\} \wedge pc' = h \wedge d' = d \\ & \langle \text{proof} \rangle \end{aligned}$$

For the *Invokestatic* instruction the BV has checked all handlers that guard the current *pc*.

**lemma** *Invokestatic-handlers*:

$$\begin{aligned} & \text{match-ex-table } P \ C \ pc \ xt = \text{Some } (pc', d') \implies \\ & \exists (f, t, D, h, d) \in \text{set } (\text{relevant-entries } P \ (\text{Invokestatic } C_0 \ n \ M) \ pc \ xt). \\ & P \vdash C \preceq^* D \wedge pc \in \{f..<t\} \wedge pc' = h \wedge d' = d \\ & \langle \text{proof} \rangle \end{aligned}$$

For the instrs in *Called-set* the BV has checked all handlers that guard the current *pc*.

**lemma** *Called-set-handlers*:

$$\begin{aligned} & \text{match-ex-table } P \ C \ pc \ xt = \text{Some } (pc', d') \implies i \in \text{Called-set} \implies \\ & \exists (f, t, D, h, d) \in \text{set } (\text{relevant-entries } P \ i \ pc \ xt). \\ & P \vdash C \preceq^* D \wedge pc \in \{f..<t\} \wedge pc' = h \wedge d' = d \\ & \langle \text{proof} \rangle \end{aligned}$$

We can prove separately that the recursive search for exception handlers (*find-handler*) in the frame stack results in a conforming state (if there was no matching exception handler in the current frame). We require that the exception is a valid heap address, and that the state before the exception occurred conforms.

**lemma** *uncaught-xcpt-correct*:

**assumes** *wt: wf-jvm-prog*<sub>Φ</sub> *P*

**assumes**  $h$ :  $h \text{ xcp} = \text{Some } \text{obj}$   
**shows**  $\bigwedge f. P, \Phi \vdash (\text{None}, h, f\# \text{frs}, sh) \surd$   
 $\implies \text{curr-method } f \neq \text{clinit} \implies P, \Phi \vdash \text{find-handler } P \text{ xcp } h \text{ frs } sh \surd$   
**(is**  $\bigwedge f. ?\text{correct} (\text{None}, h, f\# \text{frs}, sh) \implies ?\text{prem } f \implies ?\text{correct} (? \text{find frs}) \langle \text{proof} \rangle$

The requirement of lemma *uncaught-xcpt-correct* (that the exception is a valid reference on the heap) is always met for welltyped instructions and conformant states:

**lemma** *exec-instr-xcpt-h*:

$\llbracket \text{fst} (\text{exec-instr} (\text{ins!pc}) P h \text{ stk vars } C M \text{ pc ics frs sh}) = \text{Some } \text{xcp};$   
 $P, T, \text{mxs}, \text{size } \text{ins}, \text{xt} \vdash \text{ins!pc}, \text{pc} :: \Phi C M;$   
 $P, \Phi \vdash (\text{None}, h, (\text{stk}, \text{loc}, C, M, \text{pc}, \text{ics})\# \text{frs}, sh) \surd \rrbracket$   
 $\implies \exists \text{obj}. h \text{ xcp} = \text{Some } \text{obj}$   
**(is**  $\llbracket ?\text{xcpt}; ?\text{wt}; ?\text{correct} \rrbracket \implies ?\text{thesis} \langle \text{proof} \rangle$

**lemma** *exec-step-xcpt-h*:

**assumes**  $\text{xcpt}$ :  $\text{fst} (\text{exec-step } P h \text{ stk vars } C M \text{ pc ics frs sh}) = \text{Some } \text{xcp}$   
**and**  $\text{ins}$ :  $\text{instrs-of } P C M = \text{ins}$   
**and**  $\text{wti}$ :  $P, T, \text{mxs}, \text{size } \text{ins}, \text{xt} \vdash \text{ins!pc}, \text{pc} :: \Phi C M$   
**and**  $\text{correct}$ :  $P, \Phi \vdash (\text{None}, h, (\text{stk}, \text{loc}, C, M, \text{pc}, \text{ics})\# \text{frs}, sh) \surd$   
**shows**  $\exists \text{obj}. h \text{ xcp} = \text{Some } \text{obj}$   
 $\langle \text{proof} \rangle$

**lemma** *conf-sys-xcpt*:

$\llbracket \text{preallocated } h; C \in \text{sys-xcpts} \rrbracket \implies P, h \vdash \text{Addr} (\text{addr-of-sys-xcpt } C) : \leq \text{Class } C$   
 $\langle \text{proof} \rangle$

**lemma** *match-ex-table-SomeD*:

$\text{match-ex-table } P C \text{ pc } \text{xt} = \text{Some } (\text{pc}', d') \implies$   
 $\exists (f, t, D, h, d) \in \text{set } \text{xt}. \text{matches-ex-entry } P C \text{ pc } (f, t, D, h, d) \wedge h = \text{pc}' \wedge d = d'$   
 $\langle \text{proof} \rangle$

Finally we can state that, whenever an exception occurs, the next state always conforms:

**lemma** *xcpt-correct*:

**fixes**  $\sigma' :: \text{jvm-state}$   
**assumes**  $\text{wtp}$ :  $\text{wf-jvm-prog}_\Phi P$   
**assumes**  $\text{meth}$ :  $P \vdash C \text{ sees } M, b: Ts \rightarrow T = (\text{mxs}, \text{mxl}_0, \text{ins}, \text{xt}) \text{ in } C$   
**assumes**  $\text{wt}$ :  $P, T, \text{mxs}, \text{size } \text{ins}, \text{xt} \vdash \text{ins!pc}, \text{pc} :: \Phi C M$   
**assumes**  $\text{xp}$ :  $\text{fst} (\text{exec-step } P h \text{ stk loc } C M \text{ pc ics frs sh}) = \text{Some } \text{xcp}$   
**assumes**  $s'$ :  $\text{Some } \sigma' = \text{exec} (P, \text{None}, h, (\text{stk}, \text{loc}, C, M, \text{pc}, \text{ics})\# \text{frs}, sh)$   
**assumes**  $\text{correct}$ :  $P, \Phi \vdash (\text{None}, h, (\text{stk}, \text{loc}, C, M, \text{pc}, \text{ics})\# \text{frs}, sh) \surd$   
**shows**  $P, \Phi \vdash \sigma' \surd \langle \text{proof} \rangle$

**declare**  $\text{defs1}$  [*simp*]

### 2.18.3 Initialization procedure steps

In this section we prove that, for states that result in a step of the initialization procedure rather than an instruction execution, the state after execution of the step still conforms.

**lemma** *Calling-correct*:

**fixes**  $\sigma' :: \text{jvm-state}$   
**assumes**  $\text{wtprog}$ :  $\text{wf-jvm-prog}_\Phi P$   
**assumes**  $mC$ :  $P \vdash C \text{ sees } M, b: Ts \rightarrow T = (\text{mxs}, \text{mxl}_0, \text{ins}, \text{xt}) \text{ in } C$   
**assumes**  $s'$ :  $\text{Some } \sigma' = \text{exec} (P, \text{None}, h, (\text{stk}, \text{loc}, C, M, \text{pc}, \text{ics})\# \text{frs}, sh)$   
**assumes**  $\text{cf}$ :  $P, \Phi \vdash (\text{None}, h, (\text{stk}, \text{loc}, C, M, \text{pc}, \text{ics})\# \text{frs}, sh) \surd$

**assumes**  $xc$ :  $fst (exec\text{-}step\ P\ h\ stk\ loc\ C\ M\ pc\ ics\ frs\ sh) = None$   
**assumes**  $ics$ :  $ics = Calling\ C'\ Cs$

**shows**  $P, \Phi \vdash \sigma' \surd$   
 $\langle proof \rangle$

**lemma** *Throwing-correct*:

**fixes**  $\sigma' :: jvm\text{-}state$   
**assumes**  $wtprog$ :  $wf\text{-}jvm\text{-}prog_{\Phi}\ P$   
**assumes**  $mC$ :  $P \vdash C\ sees\ M, b: Ts \rightarrow T = (m\ x\ s, m\ x\ l_0, ins, xt)$  in  $C$   
**assumes**  $s'$ :  $Some\ \sigma' = exec\ (P, None, h, (stk, loc, C, M, pc, ics) \# frs, sh)$   
**assumes**  $cf$ :  $P, \Phi \vdash (None, h, (stk, loc, C, M, pc, ics) \# frs, sh) \surd$   
**assumes**  $xc$ :  $fst (exec\text{-}step\ P\ h\ stk\ loc\ C\ M\ pc\ ics\ frs\ sh) = None$   
**assumes**  $ics$ :  $ics = Throwing\ (C' \# Cs)\ a$

**shows**  $P, \Phi \vdash \sigma' \surd$   
 $\langle proof \rangle$

**lemma** *Called-correct*:

**fixes**  $\sigma' :: jvm\text{-}state$   
**assumes**  $wtprog$ :  $wf\text{-}jvm\text{-}prog_{\Phi}\ P$   
**assumes**  $mC$ :  $P \vdash C\ sees\ M, b: Ts \rightarrow T = (m\ x\ s, m\ x\ l_0, ins, xt)$  in  $C$   
**assumes**  $s'$ :  $Some\ \sigma' = exec\ (P, None, h, (stk, loc, C, M, pc, ics) \# frs, sh)$   
**assumes**  $cf$ :  $P, \Phi \vdash (None, h, (stk, loc, C, M, pc, ics) \# frs, sh) \surd$   
**assumes**  $xc$ :  $fst (exec\text{-}step\ P\ h\ stk\ loc\ C\ M\ pc\ ics\ frs\ sh) = None$   
**assumes**  $ics[simp]$ :  $ics = Called\ (C' \# Cs)$

**shows**  $P, \Phi \vdash \sigma' \surd$   
 $\langle proof \rangle$

## 2.18.4 Single Instructions

In this section we prove for each single (welltyped) instruction that the state after execution of the instruction still conforms. Since we have already handled exceptions above, we can now assume that no exception occurs in this step. For instructions that may call the initialization procedure, we cover the calling and non-calling cases separately.

**lemma** *Invoke-correct*:

**fixes**  $\sigma' :: jvm\text{-}state$   
**assumes**  $wtprog$ :  $wf\text{-}jvm\text{-}prog_{\Phi}\ P$   
**assumes**  $meth\text{-}C$ :  $P \vdash C\ sees\ M, b: Ts \rightarrow T = (m\ x\ s, m\ x\ l_0, ins, xt)$  in  $C$   
**assumes**  $ins$ :  $ins ! pc = Invoke\ M'\ n$   
**assumes**  $wti$ :  $P, T, m\ x\ s, size\ ins, xt \vdash ins ! pc, pc :: \Phi\ C\ M$   
**assumes**  $\sigma'$ :  $Some\ \sigma' = exec\ (P, None, h, (stk, loc, C, M, pc, ics) \# frs, sh)$   
**assumes**  $approx$ :  $P, \Phi \vdash (None, h, (stk, loc, C, M, pc, ics) \# frs, sh) \surd$   
**assumes**  $no\text{-}xcp$ :  $fst (exec\text{-}step\ P\ h\ stk\ loc\ C\ M\ pc\ ics\ frs\ sh) = None$   
**shows**  $P, \Phi \vdash \sigma' \surd \langle proof \rangle$

**lemma** *Invokestatic-nInit-correct*:

**fixes**  $\sigma' :: jvm\text{-}state$   
**assumes**  $wtprog$ :  $wf\text{-}jvm\text{-}prog_{\Phi}\ P$   
**assumes**  $meth\text{-}C$ :  $P \vdash C\ sees\ M, b: Ts \rightarrow T = (m\ x\ s, m\ x\ l_0, ins, xt)$  in  $C$   
**assumes**  $ins$ :  $ins ! pc = Invokestatic\ D\ M'\ n$  **and**  $nclinit$ :  $M' \neq clinit$   
**assumes**  $wti$ :  $P, T, m\ x\ s, size\ ins, xt \vdash ins ! pc, pc :: \Phi\ C\ M$   
**assumes**  $\sigma'$ :  $Some\ \sigma' = exec\ (P, None, h, (stk, loc, C, M, pc, ics) \# frs, sh)$

**assumes** *approx*:  $P, \Phi \vdash (\text{None}, h, (\text{stk}, \text{loc}, C, M, \text{pc}, \text{ics}) \# \text{frs}, \text{sh}) \surd$   
**assumes** *no-xcp*:  $\text{fst} (\text{exec-step } P \ h \ \text{stk} \ \text{loc} \ C \ M \ \text{pc} \ \text{ics} \ \text{frs} \ \text{sh}) = \text{None}$   
**assumes** *cs*:  $\text{ics} = \text{Called } [] \vee (\text{ics} = \text{No-ics} \wedge (\exists \text{sfs}. \text{sh} (\text{fst}(\text{method } P \ D \ M')) = \text{Some}(\text{sfs}, \text{Done})))$   
**shows**  $P, \Phi \vdash \sigma' \surd \langle \text{proof} \rangle$

**lemma** *Invokestatic-Init-correct*:

**fixes**  $\sigma' :: \text{jvm-state}$   
**assumes** *wfprog*:  $\text{wf-jvm-prog}_{\Phi} P$   
**assumes** *meth-C*:  $P \vdash C \text{ sees } M, b: Ts \rightarrow T = (\text{mxs}, \text{mxl}_0, \text{ins}, \text{xt}) \text{ in } C$   
**assumes** *ins*:  $\text{ins} ! \text{pc} = \text{Invokestatic } D \ M' \ n$  **and** *nclinit*:  $M' \neq \text{clinit}$   
**assumes** *wti*:  $P, T, \text{mxs}, \text{size} \ \text{ins}, \text{xt} \vdash \text{ins} ! \text{pc}, \text{pc} :: \Phi \ C \ M$   
**assumes**  $\sigma'$ :  $\text{Some } \sigma' = \text{exec} (P, \text{None}, h, (\text{stk}, \text{loc}, C, M, \text{pc}, \text{No-ics}) \# \text{frs}, \text{sh})$   
**assumes** *approx*:  $P, \Phi \vdash (\text{None}, h, (\text{stk}, \text{loc}, C, M, \text{pc}, \text{No-ics}) \# \text{frs}, \text{sh}) \surd$   
**assumes** *no-xcp*:  $\text{fst} (\text{exec-step } P \ h \ \text{stk} \ \text{loc} \ C \ M \ \text{pc} \ \text{No-ics} \ \text{frs} \ \text{sh}) = \text{None}$   
**assumes** *nDone*:  $\forall \text{sfs}. \text{sh} (\text{fst}(\text{method } P \ D \ M')) \neq \text{Some}(\text{sfs}, \text{Done})$   
**shows**  $P, \Phi \vdash \sigma' \surd \langle \text{proof} \rangle$   
**declare** *list-all2-Cons2* [*iff*]

**lemma** *Return-correct*:

**fixes**  $\sigma' :: \text{jvm-state}$   
**assumes** *wf-prog*:  $\text{wf-jvm-prog}_{\Phi} P$   
**assumes** *meth*:  $P \vdash C \text{ sees } M, b: Ts \rightarrow T = (\text{mxs}, \text{mxl}_0, \text{ins}, \text{xt}) \text{ in } C$   
**assumes** *ins*:  $\text{ins} ! \text{pc} = \text{Return}$   
**assumes** *wt*:  $P, T, \text{mxs}, \text{size} \ \text{ins}, \text{xt} \vdash \text{ins} ! \text{pc}, \text{pc} :: \Phi \ C \ M$   
**assumes**  $\sigma'$ :  $\text{Some } \sigma' = \text{exec} (P, \text{None}, h, (\text{stk}, \text{loc}, C, M, \text{pc}, \text{ics}) \# \text{frs}, \text{sh})$   
**assumes** *correct*:  $P, \Phi \vdash (\text{None}, h, (\text{stk}, \text{loc}, C, M, \text{pc}, \text{ics}) \# \text{frs}, \text{sh}) \surd$

**shows**  $P, \Phi \vdash \sigma' \surd \langle \text{proof} \rangle$

**declare** *sup-state-opt-any-Some* [*iff*]

**declare** *not-Err-eq* [*iff*]

**lemma** *Load-correct*:

**assumes** *wf-prog wt P and*  
 $mC: P \vdash C \text{ sees } M, b: Ts \rightarrow T = (\text{mxs}, \text{mxl}_0, \text{ins}, \text{xt}) \text{ in } C$  **and**  
 $i: \text{ins} ! \text{pc} = \text{Load } \text{idx}$  **and**  
 $P, T, \text{mxs}, \text{size} \ \text{ins}, \text{xt} \vdash \text{ins} ! \text{pc}, \text{pc} :: \Phi \ C \ M$  **and**  
 $\text{Some } \sigma' = \text{exec} (P, \text{None}, h, (\text{stk}, \text{loc}, C, M, \text{pc}, \text{ics}) \# \text{frs}, \text{sh})$  **and**  
 $cf: P, \Phi \vdash (\text{None}, h, (\text{stk}, \text{loc}, C, M, \text{pc}, \text{ics}) \# \text{frs}, \text{sh}) \surd$   
**shows**  $P, \Phi \vdash \sigma' \text{ [ok]} \langle \text{proof} \rangle$   
**declare** [*simproc del: list-to-set-comprehension*]

**lemma** *Store-correct*:

**assumes** *wf-prog wt P and*  
 $mC: P \vdash C \text{ sees } M, b: Ts \rightarrow T = (\text{mxs}, \text{mxl}_0, \text{ins}, \text{xt}) \text{ in } C$  **and**  
 $i: \text{ins} ! \text{pc} = \text{Store } \text{idx}$  **and**  
 $P, T, \text{mxs}, \text{size} \ \text{ins}, \text{xt} \vdash \text{ins} ! \text{pc}, \text{pc} :: \Phi \ C \ M$  **and**  
 $\text{Some } \sigma' = \text{exec} (P, \text{None}, h, (\text{stk}, \text{loc}, C, M, \text{pc}, \text{ics}) \# \text{frs}, \text{sh})$  **and**  
 $cf: P, \Phi \vdash (\text{None}, h, (\text{stk}, \text{loc}, C, M, \text{pc}, \text{ics}) \# \text{frs}, \text{sh}) \surd$   
**shows**  $P, \Phi \vdash \sigma' \text{ [ok]} \langle \text{proof} \rangle$

**lemma** *Push-correct*:

**assumes** *wf-prog wt P and*  
 $mC: P \vdash C \text{ sees } M, b: Ts \rightarrow T = (\text{mxs}, \text{mxl}_0, \text{ins}, \text{xt}) \text{ in } C$  **and**  
 $i: \text{ins} ! \text{pc} = \text{Push } v$  **and**  
 $P, T, \text{mxs}, \text{size} \ \text{ins}, \text{xt} \vdash \text{ins} ! \text{pc}, \text{pc} :: \Phi \ C \ M$  **and**



As corollary we get that the aggressive and the defensive machine are equivalent for well-typed programs (if started in a conformant state or in the canonical start state)

**corollary** *welltyped-commutes*:

**fixes**  $\sigma :: \text{jvm-state}$

**assumes**  $wf: wf\text{-jvm-prog}_{\Phi} P$  **and**  $conforms: P, \Phi \vdash \sigma \checkmark$

**shows**  $P \vdash (\text{Normal } \sigma) \text{-jvmd} \rightarrow (\text{Normal } \sigma') = P \vdash \sigma \text{-jvm} \rightarrow \sigma'$

*\langle proof \rangle*

**corollary** *welltyped-initial-commutes*:

**assumes**  $wf: wf\text{-jvm-prog } P$

**assumes**  $nstart: \neg \text{is-class } P \text{ Start}$

**assumes**  $meth: P \vdash C \text{ sees } M, \text{Static}: [] \rightarrow \text{Void} = b \text{ in } C$

**assumes**  $nclinit: M \neq \text{clinit}$

**assumes** *Obj-start-m*:

$(\bigwedge b' Ts' T' m' D'. P \vdash \text{Object sees start-m}, b' : Ts' \rightarrow T' = m' \text{ in } D'$   
 $\implies b' = \text{Static} \wedge Ts' = [] \wedge T' = \text{Void})$

**defines**  $start: \sigma \equiv \text{start-state } P$

**shows**  $start\text{-prog } P C M \vdash (\text{Normal } \sigma) \text{-jvmd} \rightarrow (\text{Normal } \sigma') = start\text{-prog } P C M \vdash \sigma \text{-jvm} \rightarrow \sigma'$

*\langle proof \rangle*

**lemma** *not-TypeError-eq [iff]*:

$x \neq \text{TypeError} = (\exists t. x = \text{Normal } t)$

*\langle proof \rangle*

**locale** *cnf* =

**fixes**  $P$  **and**  $\Phi$  **and**  $\sigma$

**assumes**  $wf: wf\text{-jvm-prog}_{\Phi} P$

**assumes**  $cnf: \text{correct-state } P \Phi \sigma$

**theorem** (**in** *cnf*) *no-type-errors*:

$P \vdash (\text{Normal } \sigma) \text{-jvmd} \rightarrow \sigma' \implies \sigma' \neq \text{TypeError}$

*\langle proof \rangle*

**locale** *start* =

**fixes**  $P$  **and**  $C$  **and**  $M$  **and**  $\sigma$  **and**  $T$  **and**  $b$  **and**  $P_0$

**assumes**  $wf: wf\text{-jvm-prog } P$

**assumes**  $nstart: \neg \text{is-class } P \text{ Start}$

**assumes**  $sees: P \vdash C \text{ sees } M, \text{Static}: [] \rightarrow \text{Void} = b \text{ in } C$

**assumes**  $nclinit: M \neq \text{clinit}$

**assumes** *Obj-start-m*:  $(\bigwedge b' Ts' T' m' D'. P \vdash \text{Object sees start-m}, b' : Ts' \rightarrow T' = m' \text{ in } D'$   
 $\implies b' = \text{Static} \wedge Ts' = [] \wedge T' = \text{Void})$

**defines**  $\sigma \equiv \text{Normal } (\text{start-state } P)$

**defines**  $[simp]: P_0 \equiv \text{start-prog } P C M$

**corollary** (**in** *start*) *bv-no-type-error*:

**shows**  $P_0 \vdash \sigma \text{-jvmd} \rightarrow \sigma' \implies \sigma' \neq \text{TypeError}$

*\langle proof \rangle*

**end**



# Chapter 3

## Compilation

### 3.1 An Intermediate Language

theory *J1* imports *../J/BigStep* begin

**type-synonym** *expr*<sub>1</sub> = *nat exp*

**type-synonym** *J*<sub>1</sub>-*prog* = *expr*<sub>1</sub> *prog*

**type-synonym** *state*<sub>1</sub> = *heap* × (*val list*) × *sheap*

**definition** *hp*<sub>1</sub> :: *state*<sub>1</sub> ⇒ *heap*

**where**

*hp*<sub>1</sub> ≡ *fst*

**definition** *lcl*<sub>1</sub> :: *state*<sub>1</sub> ⇒ *val list*

**where**

*lcl*<sub>1</sub> ≡ *fst* ∘ *snd*

**definition** *shp*<sub>1</sub> :: *state*<sub>1</sub> ⇒ *sheap*

**where**

*shp*<sub>1</sub> ≡ *snd* ∘ *snd*

**primrec**

*max-vars* :: 'a *exp* ⇒ *nat*

**and** *max-varss* :: 'a *exp list* ⇒ *nat*

**where**

*max-vars*(*new C*) = 0  
| *max-vars*(*Cast C e*) = *max-vars e*  
| *max-vars*(*Val v*) = 0  
| *max-vars*(*e*<sub>1</sub> «*bop*» *e*<sub>2</sub>) = *max* (*max-vars e*<sub>1</sub>) (*max-vars e*<sub>2</sub>)  
| *max-vars*(*Var V*) = 0  
| *max-vars*(*V:=e*) = *max-vars e*  
| *max-vars*(*e*·*F*{*D*}) = *max-vars e*  
| *max-vars*(*C*·<sub>s</sub>*F*{*D*}) = 0  
| *max-vars*(*FAss e*<sub>1</sub> *F D e*<sub>2</sub>) = *max* (*max-vars e*<sub>1</sub>) (*max-vars e*<sub>2</sub>)  
| *max-vars*(*SFAss C F D e*<sub>2</sub>) = *max-vars e*<sub>2</sub>  
| *max-vars*(*e*·*M*(*es*)) = *max* (*max-vars e*) (*max-varss es*)  
| *max-vars*(*C*·<sub>s</sub>*M*(*es*)) = *max-varss es*  
| *max-vars*({*V:T*; *e*}) = *max-vars e* + 1  
| *max-vars*(*e*<sub>1</sub>;;*e*<sub>2</sub>) = *max* (*max-vars e*<sub>1</sub>) (*max-vars e*<sub>2</sub>)  
| *max-vars*(*if* (*e*) *e*<sub>1</sub> *else e*<sub>2</sub>) =  
  *max* (*max-vars e*) (*max* (*max-vars e*<sub>1</sub>) (*max-vars e*<sub>2</sub>))  
| *max-vars*(*while* (*b*) *e*) = *max* (*max-vars b*) (*max-vars e*)

|  $max\text{-vars}(throw\ e) = max\text{-vars}\ e$   
|  $max\text{-vars}(try\ e_1\ catch(C\ V)\ e_2) = max\ (max\text{-vars}\ e_1)\ (max\text{-vars}\ e_2 + 1)$   
|  $max\text{-vars}(INIT\ C\ (Cs,b) \leftarrow e) = max\text{-vars}\ e$   
|  $max\text{-vars}(RI(C,e);Cs \leftarrow e') = max\ (max\text{-vars}\ e)\ (max\text{-vars}\ e')$

|  $max\text{-varss}\ [] = 0$   
|  $max\text{-varss}\ (e\#\!es) = max\ (max\text{-vars}\ e)\ (max\text{-varss}\ es)$

**inductive**

$eval_1 :: J_1\text{-prog} \Rightarrow expr_1 \Rightarrow state_1 \Rightarrow expr_1 \Rightarrow state_1 \Rightarrow bool$   
 $(\leftarrow \vdash_1 ((I\langle -,/- \rangle) \Rightarrow / (I\langle -,/- \rangle))) \triangleright [51,0,0,0,0] 81$

**and**  $evals_1 :: J_1\text{-prog} \Rightarrow expr_1\ list \Rightarrow state_1 \Rightarrow expr_1\ list \Rightarrow state_1 \Rightarrow bool$   
 $(\leftarrow \vdash_1 ((I\langle -,/- \rangle) [\Rightarrow] / (I\langle -,/- \rangle))) \triangleright [51,0,0,0,0] 81$

**for**  $P :: J_1\text{-prog}$

**where**

$New_1:$

$\llbracket sh\ C = Some\ (sfs,\ Done); new\text{-Addr}\ h = Some\ a;$   
 $P \vdash C\ has\text{-fields}\ FDTs; h' = h(a \mapsto blank\ P\ C) \rrbracket$   
 $\implies P \vdash_1 \langle new\ C, (h,l,sh) \rangle \Rightarrow \langle addr\ a, (h',l,sh) \rangle$

|  $NewFail_1:$

$\llbracket sh\ C = Some\ (sfs,\ Done); new\text{-Addr}\ h = None \rrbracket \implies$   
 $P \vdash_1 \langle new\ C, (h,l,sh) \rangle \Rightarrow \langle THROW\ OutOfMemory, (h,l,sh) \rangle$

|  $NewInit_1:$

$\llbracket \#sfs.\ sh\ C = Some\ (sfs,\ Done); P \vdash_1 \langle INIT\ C\ ([C],False) \leftarrow unit, (h,l,sh) \rangle \Rightarrow \langle Val\ v', (h',l',sh') \rangle;$   
 $new\text{-Addr}\ h' = Some\ a; P \vdash C\ has\text{-fields}\ FDTs; h'' = h'(a \mapsto blank\ P\ C) \rrbracket$   
 $\implies P \vdash_1 \langle new\ C, (h,l,sh) \rangle \Rightarrow \langle addr\ a, (h'',l',sh') \rangle$

|  $NewInitOOM_1:$

$\llbracket \#sfs.\ sh\ C = Some\ (sfs,\ Done); P \vdash_1 \langle INIT\ C\ ([C],False) \leftarrow unit, (h,l,sh) \rangle \Rightarrow \langle Val\ v', (h',l',sh') \rangle;$   
 $new\text{-Addr}\ h' = None; is\text{-class}\ P\ C \rrbracket$   
 $\implies P \vdash_1 \langle new\ C, (h,l,sh) \rangle \Rightarrow \langle THROW\ OutOfMemory, (h',l',sh') \rangle$

|  $NewInitThrow_1:$

$\llbracket \#sfs.\ sh\ C = Some\ (sfs,\ Done); P \vdash_1 \langle INIT\ C\ ([C],False) \leftarrow unit, (h,l,sh) \rangle \Rightarrow \langle throw\ a, s' \rangle;$   
 $is\text{-class}\ P\ C \rrbracket$   
 $\implies P \vdash_1 \langle new\ C, (h,l,sh) \rangle \Rightarrow \langle throw\ a, s' \rangle$

|  $Cast_1:$

$\llbracket P \vdash_1 \langle e, s_0 \rangle \Rightarrow \langle addr\ a, (h,l,sh) \rangle; h\ a = Some(D,fs); P \vdash D \preceq^* C \rrbracket$   
 $\implies P \vdash_1 \langle Cast\ C\ e, s_0 \rangle \Rightarrow \langle addr\ a, (h,l,sh) \rangle$

|  $CastNull_1:$

$P \vdash_1 \langle e, s_0 \rangle \Rightarrow \langle null, s_1 \rangle \implies$   
 $P \vdash_1 \langle Cast\ C\ e, s_0 \rangle \Rightarrow \langle null, s_1 \rangle$

|  $CastFail_1:$

$\llbracket P \vdash_1 \langle e, s_0 \rangle \Rightarrow \langle addr\ a, (h,l,sh) \rangle; h\ a = Some(D,fs); \neg P \vdash D \preceq^* C \rrbracket$   
 $\implies P \vdash_1 \langle Cast\ C\ e, s_0 \rangle \Rightarrow \langle THROW\ ClassCast, (h,l,sh) \rangle$

|  $CastThrow_1:$

$P \vdash_1 \langle e, s_0 \rangle \Rightarrow \langle throw\ e', s_1 \rangle \implies$   
 $P \vdash_1 \langle Cast\ C\ e, s_0 \rangle \Rightarrow \langle throw\ e', s_1 \rangle$

|  $Val_1:$

$P \vdash_1 \langle Val\ v, s \rangle \Rightarrow \langle Val\ v, s \rangle$

|  $BinOp_1:$

$\llbracket P \vdash_1 \langle e_1, s_0 \rangle \Rightarrow \langle Val\ v_1, s_1 \rangle; P \vdash_1 \langle e_2, s_1 \rangle \Rightarrow \langle Val\ v_2, s_2 \rangle; binop(bop, v_1, v_2) = Some\ v \rrbracket$

$$\begin{aligned} & \Longrightarrow P \vdash_1 \langle e_1 \text{ «bop» } e_2, s_0 \rangle \Rightarrow \langle \text{Val } v, s_2 \rangle \\ | \text{BinOpThrow}_{11}: & \\ & P \vdash_1 \langle e_1, s_0 \rangle \Rightarrow \langle \text{throw } e, s_1 \rangle \Longrightarrow \\ & P \vdash_1 \langle e_1 \text{ «bop» } e_2, s_0 \rangle \Rightarrow \langle \text{throw } e, s_1 \rangle \\ | \text{BinOpThrow}_{21}: & \\ & \llbracket P \vdash_1 \langle e_1, s_0 \rangle \Rightarrow \langle \text{Val } v_1, s_1 \rangle; P \vdash_1 \langle e_2, s_1 \rangle \Rightarrow \langle \text{throw } e, s_2 \rangle \rrbracket \\ & \Longrightarrow P \vdash_1 \langle e_1 \text{ «bop» } e_2, s_0 \rangle \Rightarrow \langle \text{throw } e, s_2 \rangle \\ | \text{Var}_1: & \\ & \llbracket \text{ls!}i = v; i < \text{size } \text{ls} \rrbracket \Longrightarrow \\ & P \vdash_1 \langle \text{Var } i, (h, \text{ls}, \text{sh}) \rangle \Rightarrow \langle \text{Val } v, (h, \text{ls}, \text{sh}) \rangle \\ | \text{LAss}_1: & \\ & \llbracket P \vdash_1 \langle e, s_0 \rangle \Rightarrow \langle \text{Val } v, (h, \text{ls}, \text{sh}) \rangle; i < \text{size } \text{ls}; \text{ls}' = \text{ls}[i := v] \rrbracket \\ & \Longrightarrow P \vdash_1 \langle i := e, s_0 \rangle \Rightarrow \langle \text{unit}, (h, \text{ls}', \text{sh}) \rangle \\ | \text{LAssThrow}_1: & \\ & P \vdash_1 \langle e, s_0 \rangle \Rightarrow \langle \text{throw } e', s_1 \rangle \Longrightarrow \\ & P \vdash_1 \langle i := e, s_0 \rangle \Rightarrow \langle \text{throw } e', s_1 \rangle \\ | \text{FAcc}_1: & \\ & \llbracket P \vdash_1 \langle e, s_0 \rangle \Rightarrow \langle \text{addr } a, (h, \text{ls}, \text{sh}) \rangle; h a = \text{Some}(C, \text{fs}); \\ & \quad P \vdash C \text{ has } F, \text{NonStatic}:t \text{ in } D; \\ & \quad \text{fs}(F, D) = \text{Some } v \rrbracket \\ & \Longrightarrow P \vdash_1 \langle e \cdot F\{D\}, s_0 \rangle \Rightarrow \langle \text{Val } v, (h, \text{ls}, \text{sh}) \rangle \\ | \text{FAccNull}_1: & \\ & P \vdash_1 \langle e, s_0 \rangle \Rightarrow \langle \text{null}, s_1 \rangle \Longrightarrow \\ & P \vdash_1 \langle e \cdot F\{D\}, s_0 \rangle \Rightarrow \langle \text{THROW NullPointer}, s_1 \rangle \\ | \text{FAccThrow}_1: & \\ & P \vdash_1 \langle e, s_0 \rangle \Rightarrow \langle \text{throw } e', s_1 \rangle \Longrightarrow \\ & P \vdash_1 \langle e \cdot F\{D\}, s_0 \rangle \Rightarrow \langle \text{throw } e', s_1 \rangle \\ | \text{FAccNone}_1: & \\ & \llbracket P \vdash_1 \langle e, s_0 \rangle \Rightarrow \langle \text{addr } a, (h, \text{ls}, \text{sh}) \rangle; h a = \text{Some}(C, \text{fs}); \\ & \quad \neg(\exists b t. P \vdash C \text{ has } F, b:t \text{ in } D) \rrbracket \\ & \Longrightarrow P \vdash_1 \langle e \cdot F\{D\}, s_0 \rangle \Rightarrow \langle \text{THROW NoSuchFieldError}, (h, \text{ls}, \text{sh}) \rangle \\ | \text{FAccStatic}_1: & \\ & \llbracket P \vdash_1 \langle e, s_0 \rangle \Rightarrow \langle \text{addr } a, (h, \text{ls}, \text{sh}) \rangle; h a = \text{Some}(C, \text{fs}); \\ & \quad P \vdash C \text{ has } F, \text{Static}:t \text{ in } D \rrbracket \\ & \Longrightarrow P \vdash_1 \langle e \cdot F\{D\}, s_0 \rangle \Rightarrow \langle \text{THROW IncompatibleClassChangeError}, (h, \text{ls}, \text{sh}) \rangle \\ | \text{SFAcc}_1: & \\ & \llbracket P \vdash C \text{ has } F, \text{Static}:t \text{ in } D; \\ & \quad \text{sh } D = \text{Some}(\text{sfs}, \text{Done}); \\ & \quad \text{sfs } F = \text{Some } v \rrbracket \\ & \Longrightarrow P \vdash_1 \langle C \cdot_s F\{D\}, (h, \text{ls}, \text{sh}) \rangle \Rightarrow \langle \text{Val } v, (h, \text{ls}, \text{sh}) \rangle \\ | \text{SFAccInit}_1: & \\ & \llbracket P \vdash C \text{ has } F, \text{Static}:t \text{ in } D; \\ & \quad \# \text{sfs}. \text{sh } D = \text{Some}(\text{sfs}, \text{Done}); P \vdash_1 \langle \text{INIT } D ([D], \text{False}) \leftarrow \text{unit}, (h, \text{ls}, \text{sh}) \rangle \Rightarrow \langle \text{Val } v', (h', \text{ls}', \text{sh}') \rangle; \\ & \quad \text{sh}' D = \text{Some}(\text{sfs}, i); \\ & \quad \text{sfs } F = \text{Some } v \rrbracket \\ & \Longrightarrow P \vdash_1 \langle C \cdot_s F\{D\}, (h, \text{ls}, \text{sh}) \rangle \Rightarrow \langle \text{Val } v, (h', \text{ls}', \text{sh}') \rangle \\ | \text{SFAccInitThrow}_1: & \\ & \llbracket P \vdash C \text{ has } F, \text{Static}:t \text{ in } D; \\ & \quad \# \text{sfs}. \text{sh } D = \text{Some}(\text{sfs}, \text{Done}); P \vdash_1 \langle \text{INIT } D ([D], \text{False}) \leftarrow \text{unit}, (h, \text{ls}, \text{sh}) \rangle \Rightarrow \langle \text{throw } a, s^\wedge \rrbracket \\ & \Longrightarrow P \vdash_1 \langle C \cdot_s F\{D\}, (h, \text{ls}, \text{sh}) \rangle \Rightarrow \langle \text{throw } a, s^\wedge \rangle \end{aligned}$$

$|$  *SFAccNone*<sub>1</sub>:  
 $\llbracket \neg(\exists b t. P \vdash C \text{ has } F, b:t \text{ in } D) \rrbracket$   
 $\implies P \vdash_1 \langle C \cdot_s F\{D\}, s \rangle \Rightarrow \langle \text{THROW NoSuchFieldError}, s \rangle$

$|$  *SFAccNonStatic*<sub>1</sub>:  
 $\llbracket P \vdash C \text{ has } F, \text{NonStatic}:t \text{ in } D \rrbracket$   
 $\implies P \vdash_1 \langle C \cdot_s F\{D\}, s \rangle \Rightarrow \langle \text{THROW IncompatibleClassChangeError}, s \rangle$

$|$  *FAss*<sub>1</sub>:  
 $\llbracket P \vdash_1 \langle e_1, s_0 \rangle \Rightarrow \langle \text{addr } a, s_1 \rangle; P \vdash_1 \langle e_2, s_1 \rangle \Rightarrow \langle \text{Val } v, (h_2, l_2, sh_2) \rangle;$   
 $h_2 a = \text{Some}(C, fs); P \vdash C \text{ has } F, \text{NonStatic}:t \text{ in } D;$   
 $fs' = fs(F, D) \mapsto v; h_2' = h_2(a \mapsto (C, fs')) \rrbracket$   
 $\implies P \vdash_1 \langle e_1 \cdot F\{D\} := e_2, s_0 \rangle \Rightarrow \langle \text{unit}, (h_2', l_2, sh_2) \rangle$

$|$  *FAssNull*<sub>1</sub>:  
 $\llbracket P \vdash_1 \langle e_1, s_0 \rangle \Rightarrow \langle \text{null}, s_1 \rangle; P \vdash_1 \langle e_2, s_1 \rangle \Rightarrow \langle \text{Val } v, s_2 \rangle \rrbracket \implies$   
 $P \vdash_1 \langle e_1 \cdot F\{D\} := e_2, s_0 \rangle \Rightarrow \langle \text{THROW NullPointer}, s_2 \rangle$

$|$  *FAssThrow*<sub>11</sub>:  
 $P \vdash_1 \langle e_1, s_0 \rangle \Rightarrow \langle \text{throw } e', s_1 \rangle \implies$   
 $P \vdash_1 \langle e_1 \cdot F\{D\} := e_2, s_0 \rangle \Rightarrow \langle \text{throw } e', s_1 \rangle$

$|$  *FAssThrow*<sub>21</sub>:  
 $\llbracket P \vdash_1 \langle e_1, s_0 \rangle \Rightarrow \langle \text{Val } v, s_1 \rangle; P \vdash_1 \langle e_2, s_1 \rangle \Rightarrow \langle \text{throw } e', s_2 \rangle \rrbracket$   
 $\implies P \vdash_1 \langle e_1 \cdot F\{D\} := e_2, s_0 \rangle \Rightarrow \langle \text{throw } e', s_2 \rangle$

$|$  *FAssNone*<sub>1</sub>:  
 $\llbracket P \vdash_1 \langle e_1, s_0 \rangle \Rightarrow \langle \text{addr } a, s_1 \rangle; P \vdash_1 \langle e_2, s_1 \rangle \Rightarrow \langle \text{Val } v, (h_2, l_2, sh_2) \rangle;$   
 $h_2 a = \text{Some}(C, fs); \neg(\exists b t. P \vdash C \text{ has } F, b:t \text{ in } D) \rrbracket$   
 $\implies P \vdash_1 \langle e_1 \cdot F\{D\} := e_2, s_0 \rangle \Rightarrow \langle \text{THROW NoSuchFieldError}, (h_2, l_2, sh_2) \rangle$

$|$  *FAssStatic*<sub>1</sub>:  
 $\llbracket P \vdash_1 \langle e_1, s_0 \rangle \Rightarrow \langle \text{addr } a, s_1 \rangle; P \vdash_1 \langle e_2, s_1 \rangle \Rightarrow \langle \text{Val } v, (h_2, l_2, sh_2) \rangle;$   
 $h_2 a = \text{Some}(C, fs); P \vdash C \text{ has } F, \text{Static}:t \text{ in } D \rrbracket$   
 $\implies P \vdash_1 \langle e_1 \cdot F\{D\} := e_2, s_0 \rangle \Rightarrow \langle \text{THROW IncompatibleClassChangeError}, (h_2, l_2, sh_2) \rangle$

$|$  *SFAss*<sub>1</sub>:  
 $\llbracket P \vdash_1 \langle e_2, s_0 \rangle \Rightarrow \langle \text{Val } v, (h_1, l_1, sh_1) \rangle;$   
 $P \vdash C \text{ has } F, \text{Static}:t \text{ in } D;$   
 $sh_1 D = \text{Some}(sfs, \text{Done}); sfs' = sfs(F \mapsto v); sh_1' = sh_1(D \mapsto (sfs', \text{Done})) \rrbracket$   
 $\implies P \vdash_1 \langle C \cdot_s F\{D\} := e_2, s_0 \rangle \Rightarrow \langle \text{unit}, (h_1, l_1, sh_1') \rangle$

$|$  *SFAssInit*<sub>1</sub>:  
 $\llbracket P \vdash_1 \langle e_2, s_0 \rangle \Rightarrow \langle \text{Val } v, (h_1, l_1, sh_1) \rangle;$   
 $P \vdash C \text{ has } F, \text{Static}:t \text{ in } D;$   
 $\nexists sfs. sh_1 D = \text{Some}(sfs, \text{Done}); P \vdash_1 \langle \text{INIT } D ([D], \text{False}) \leftarrow \text{unit}, (h_1, l_1, sh_1) \rangle \Rightarrow \langle \text{Val } v', (h', l', sh') \rangle;$   
 $sh' D = \text{Some}(sfs, i);$   
 $sfs' = sfs(F \mapsto v); sh'' = sh'(D \mapsto (sfs', i)) \rrbracket$   
 $\implies P \vdash_1 \langle C \cdot_s F\{D\} := e_2, s_0 \rangle \Rightarrow \langle \text{unit}, (h', l', sh'') \rangle$

$|$  *SFAssInitThrow*<sub>1</sub>:  
 $\llbracket P \vdash_1 \langle e_2, s_0 \rangle \Rightarrow \langle \text{Val } v, (h_1, l_1, sh_1) \rangle;$   
 $P \vdash C \text{ has } F, \text{Static}:t \text{ in } D;$   
 $\nexists sfs. sh_1 D = \text{Some}(sfs, \text{Done}); P \vdash_1 \langle \text{INIT } D ([D], \text{False}) \leftarrow \text{unit}, (h_1, l_1, sh_1) \rangle \Rightarrow \langle \text{throw } a, s^\wedge \rangle \rrbracket$   
 $\implies P \vdash_1 \langle C \cdot_s F\{D\} := e_2, s_0 \rangle \Rightarrow \langle \text{throw } a, s^\wedge \rangle$

$|$  *SFAssThrow*<sub>1</sub>:  
 $P \vdash_1 \langle e_2, s_0 \rangle \Rightarrow \langle \text{throw } e', s_2 \rangle$   
 $\implies P \vdash_1 \langle C \cdot_s F\{D\} := e_2, s_0 \rangle \Rightarrow \langle \text{throw } e', s_2 \rangle$

$|$  *SFAssNone*<sub>1</sub>:  
 $\llbracket P \vdash_1 \langle e_2, s_0 \rangle \Rightarrow \langle \text{Val } v, (h_2, l_2, sh_2) \rangle;$   
 $\neg(\exists b t. P \vdash C \text{ has } F, b:t \text{ in } D) \rrbracket$

$\Rightarrow P \vdash_1 \langle C \cdot_s F\{D\} := e_2, s_0 \rangle \Rightarrow \langle \text{THROW NoSuchFieldError}, (h_2, l_2, sh_2) \rangle$   
| *SFAssNonStatic*<sub>1</sub>:  
 $\llbracket P \vdash_1 \langle e_2, s_0 \rangle \Rightarrow \langle \text{Val } v, (h_2, l_2, sh_2) \rangle; \quad P \vdash C \text{ has } F, \text{NonStatic}: t \text{ in } D \rrbracket$   
 $\Rightarrow P \vdash_1 \langle C \cdot_s F\{D\} := e_2, s_0 \rangle \Rightarrow \langle \text{THROW IncompatibleClassChangeError}, (h_2, l_2, sh_2) \rangle$

| *CallObjThrow*<sub>1</sub>:  
 $P \vdash_1 \langle e, s_0 \rangle \Rightarrow \langle \text{throw } e', s_1 \rangle \Rightarrow$   
 $P \vdash_1 \langle e \cdot M(es), s_0 \rangle \Rightarrow \langle \text{throw } e', s_1 \rangle$

| *CallNull*<sub>1</sub>:  
 $\llbracket P \vdash_1 \langle e, s_0 \rangle \Rightarrow \langle \text{null}, s_1 \rangle; P \vdash_1 \langle es, s_1 \rangle [\Rightarrow] \langle \text{map Val } vs, s_2 \rangle \rrbracket$   
 $\Rightarrow P \vdash_1 \langle e \cdot M(es), s_0 \rangle \Rightarrow \langle \text{THROW NullPointer}, s_2 \rangle$

| *Call*<sub>1</sub>:  
 $\llbracket P \vdash_1 \langle e, s_0 \rangle \Rightarrow \langle \text{addr } a, s_1 \rangle; P \vdash_1 \langle es, s_1 \rangle [\Rightarrow] \langle \text{map Val } vs, (h_2, ls_2, sh_2) \rangle;$   
 $h_2 \ a = \text{Some}(C.fs); P \vdash C \text{ sees } M, \text{NonStatic}: Ts \rightarrow T = \text{body in } D;$   
 $\text{size } vs = \text{size } Ts; ls_2' = (\text{Addr } a) \# vs \ @ \ \text{replicate } (\text{max-vars body}) \ \text{undefined};$   
 $P \vdash_1 \langle \text{body}, (h_2, ls_2', sh_2) \rangle \Rightarrow \langle e', (h_3, ls_3, sh_3) \rangle \rrbracket$   
 $\Rightarrow P \vdash_1 \langle e \cdot M(es), s_0 \rangle \Rightarrow \langle e', (h_3, ls_2, sh_3) \rangle$

| *CallParamsThrow*<sub>1</sub>:  
 $\llbracket P \vdash_1 \langle e, s_0 \rangle \Rightarrow \langle \text{Val } v, s_1 \rangle; P \vdash_1 \langle es, s_1 \rangle [\Rightarrow] \langle es', s_2 \rangle;$   
 $es' = \text{map Val } vs \ @ \ \text{throw } ex \ \# \ es_2 \rrbracket$   
 $\Rightarrow P \vdash_1 \langle e \cdot M(es), s_0 \rangle \Rightarrow \langle \text{throw } ex, s_2 \rangle$

| *CallNone*<sub>1</sub>:  
 $\llbracket P \vdash_1 \langle e, s_0 \rangle \Rightarrow \langle \text{addr } a, s_1 \rangle; P \vdash_1 \langle ps, s_1 \rangle [\Rightarrow] \langle \text{map Val } vs, (h_2, ls_2, sh_2) \rangle;$   
 $h_2 \ a = \text{Some}(C.fs); \neg(\exists b \ Ts \ T \ \text{body } D. P \vdash C \text{ sees } M, b: Ts \rightarrow T = \text{body in } D) \rrbracket$   
 $\Rightarrow P \vdash_1 \langle e \cdot M(ps), s_0 \rangle \Rightarrow \langle \text{THROW NoSuchMethodError}, (h_2, ls_2, sh_2) \rangle$

| *CallStatic*<sub>1</sub>:  
 $\llbracket P \vdash_1 \langle e, s_0 \rangle \Rightarrow \langle \text{addr } a, s_1 \rangle; P \vdash_1 \langle ps, s_1 \rangle [\Rightarrow] \langle \text{map Val } vs, (h_2, ls_2, sh_2) \rangle;$   
 $h_2 \ a = \text{Some}(C.fs); P \vdash C \text{ sees } M, \text{Static}: Ts \rightarrow T = \text{body in } D \rrbracket$   
 $\Rightarrow P \vdash_1 \langle e \cdot M(ps), s_0 \rangle \Rightarrow \langle \text{THROW IncompatibleClassChangeError}, (h_2, ls_2, sh_2) \rangle$

| *SCallParamsThrow*<sub>1</sub>:  
 $\llbracket P \vdash_1 \langle es, s_0 \rangle [\Rightarrow] \langle es', s_2 \rangle; es' = \text{map Val } vs \ @ \ \text{throw } ex \ \# \ es_2 \rrbracket$   
 $\Rightarrow P \vdash_1 \langle C \cdot_s M(es), s_0 \rangle \Rightarrow \langle \text{throw } ex, s_2 \rangle$

| *SCallNone*<sub>1</sub>:  
 $\llbracket P \vdash_1 \langle ps, s_0 \rangle [\Rightarrow] \langle \text{map Val } vs, s_2 \rangle;$   
 $\neg(\exists b \ Ts \ T \ \text{body } D. P \vdash C \text{ sees } M, b: Ts \rightarrow T = \text{body in } D) \rrbracket$   
 $\Rightarrow P \vdash_1 \langle C \cdot_s M(ps), s_0 \rangle \Rightarrow \langle \text{THROW NoSuchMethodError}, s_2 \rangle$

| *SCallNonStatic*<sub>1</sub>:  
 $\llbracket P \vdash_1 \langle ps, s_0 \rangle [\Rightarrow] \langle \text{map Val } vs, s_2 \rangle;$   
 $P \vdash C \text{ sees } M, \text{NonStatic}: Ts \rightarrow T = \text{body in } D \rrbracket$   
 $\Rightarrow P \vdash_1 \langle C \cdot_s M(ps), s_0 \rangle \Rightarrow \langle \text{THROW IncompatibleClassChangeError}, s_2 \rangle$

| *SCallInitThrow*<sub>1</sub>:  
 $\llbracket P \vdash_1 \langle ps, s_0 \rangle [\Rightarrow] \langle \text{map Val } vs, (h_1, ls_1, sh_1) \rangle;$   
 $P \vdash C \text{ sees } M, \text{Static}: Ts \rightarrow T = \text{body in } D;$   
 $\nexists \text{sfs. } sh_1 \ D = \text{Some}(\text{sfs}, \text{Done}); M \neq \text{clinit};$   
 $P \vdash_1 \langle \text{INIT } D \ ([D], \text{False}) \leftarrow \text{unit}, (h_1, ls_1, sh_1) \rangle \Rightarrow \langle \text{throw } a, s^\wedge \rangle \rrbracket$   
 $\Rightarrow P \vdash_1 \langle C \cdot_s M(ps), s_0 \rangle \Rightarrow \langle \text{throw } a, s^\wedge \rangle$

| *SCallInit*<sub>1</sub>:  
 $\llbracket P \vdash_1 \langle ps, s_0 \rangle [\Rightarrow] \langle \text{map Val } vs, (h_1, ls_1, sh_1) \rangle;$   
 $P \vdash C \text{ sees } M, \text{Static}: Ts \rightarrow T = \text{body in } D;$   
 $\nexists \text{sfs. } sh_1 \ D = \text{Some}(\text{sfs}, \text{Done}); M \neq \text{clinit};$   
 $P \vdash_1 \langle \text{INIT } D \ ([D], \text{False}) \leftarrow \text{unit}, (h_1, ls_1, sh_1) \rangle \Rightarrow \langle \text{Val } v', (h_2, ls_2, sh_2) \rangle;$   
 $\text{size } vs = \text{size } Ts; ls_2' = vs \ @ \ \text{replicate } (\text{max-vars body}) \ \text{undefined};$

$$\begin{aligned}
& P \vdash_1 \langle \text{body}, (h_2, ls_2', sh_2) \rangle \Rightarrow \langle e', (h_3, ls_3, sh_3) \rangle \text{ ]} \\
& \Longrightarrow P \vdash_1 \langle C \cdot_s M(ps), s_0 \rangle \Rightarrow \langle e', (h_3, ls_2, sh_3) \rangle \\
| \text{SCall}_1: \\
& \text{[ } P \vdash_1 \langle ps, s_0 \rangle \text{ [}\Rightarrow\text{]} \langle \text{map Val } vs, (h_2, ls_2, sh_2) \rangle; \\
& P \vdash C \text{ sees } M, \text{Static: } Ts \rightarrow T = \text{body in } D; \\
& sh_2 D = \text{Some}(sfs, Done) \vee (M = \text{clinit} \wedge sh_2 D = \text{[(sfs, Processing)]}); \\
& \text{size } vs = \text{size } Ts; ls_2' = vs @ \text{replicate (max-vars body) undefined}; \\
& P \vdash_1 \langle \text{body}, (h_2, ls_2', sh_2) \rangle \Rightarrow \langle e', (h_3, ls_3, sh_3) \rangle \text{ ]} \\
& \Longrightarrow P \vdash_1 \langle C \cdot_s M(ps), s_0 \rangle \Rightarrow \langle e', (h_3, ls_2, sh_3) \rangle \\
| \text{Block}_1: \\
& P \vdash_1 \langle e, s_0 \rangle \Rightarrow \langle e', s_1 \rangle \Longrightarrow P \vdash_1 \langle \text{Block } i \ T \ e, s_0 \rangle \Rightarrow \langle e', s_1 \rangle \\
| \text{Seq}_1: \\
& \text{[ } P \vdash_1 \langle e_0, s_0 \rangle \Rightarrow \langle \text{Val } v, s_1 \rangle; P \vdash_1 \langle e_1, s_1 \rangle \Rightarrow \langle e_2, s_2 \rangle \text{ ]} \\
& \Longrightarrow P \vdash_1 \langle e_0;;e_1, s_0 \rangle \Rightarrow \langle e_2, s_2 \rangle \\
| \text{SeqThrow}_1: \\
& P \vdash_1 \langle e_0, s_0 \rangle \Rightarrow \langle \text{throw } e, s_1 \rangle \Longrightarrow \\
& P \vdash_1 \langle e_0;;e_1, s_0 \rangle \Rightarrow \langle \text{throw } e, s_1 \rangle \\
| \text{CondT}_1: \\
& \text{[ } P \vdash_1 \langle e, s_0 \rangle \Rightarrow \langle \text{true}, s_1 \rangle; P \vdash_1 \langle e_1, s_1 \rangle \Rightarrow \langle e', s_2 \rangle \text{ ]} \\
& \Longrightarrow P \vdash_1 \langle \text{if } (e) \ e_1 \ \text{else } e_2, s_0 \rangle \Rightarrow \langle e', s_2 \rangle \\
| \text{CondF}_1: \\
& \text{[ } P \vdash_1 \langle e, s_0 \rangle \Rightarrow \langle \text{false}, s_1 \rangle; P \vdash_1 \langle e_2, s_1 \rangle \Rightarrow \langle e', s_2 \rangle \text{ ]} \\
& \Longrightarrow P \vdash_1 \langle \text{if } (e) \ e_1 \ \text{else } e_2, s_0 \rangle \Rightarrow \langle e', s_2 \rangle \\
| \text{CondThrow}_1: \\
& P \vdash_1 \langle e, s_0 \rangle \Rightarrow \langle \text{throw } e', s_1 \rangle \Longrightarrow \\
& P \vdash_1 \langle \text{if } (e) \ e_1 \ \text{else } e_2, s_0 \rangle \Rightarrow \langle \text{throw } e', s_1 \rangle \\
| \text{WhileF}_1: \\
& P \vdash_1 \langle e, s_0 \rangle \Rightarrow \langle \text{false}, s_1 \rangle \Longrightarrow \\
& P \vdash_1 \langle \text{while } (e) \ c, s_0 \rangle \Rightarrow \langle \text{unit}, s_1 \rangle \\
| \text{WhileT}_1: \\
& \text{[ } P \vdash_1 \langle e, s_0 \rangle \Rightarrow \langle \text{true}, s_1 \rangle; P \vdash_1 \langle c, s_1 \rangle \Rightarrow \langle \text{Val } v_1, s_2 \rangle; \\
& P \vdash_1 \langle \text{while } (e) \ c, s_2 \rangle \Rightarrow \langle e_3, s_3 \rangle \text{ ]} \\
& \Longrightarrow P \vdash_1 \langle \text{while } (e) \ c, s_0 \rangle \Rightarrow \langle e_3, s_3 \rangle \\
| \text{WhileCondThrow}_1: \\
& P \vdash_1 \langle e, s_0 \rangle \Rightarrow \langle \text{throw } e', s_1 \rangle \Longrightarrow \\
& P \vdash_1 \langle \text{while } (e) \ c, s_0 \rangle \Rightarrow \langle \text{throw } e', s_1 \rangle \\
| \text{WhileBodyThrow}_1: \\
& \text{[ } P \vdash_1 \langle e, s_0 \rangle \Rightarrow \langle \text{true}, s_1 \rangle; P \vdash_1 \langle c, s_1 \rangle \Rightarrow \langle \text{throw } e', s_2 \rangle \text{ ]} \\
& \Longrightarrow P \vdash_1 \langle \text{while } (e) \ c, s_0 \rangle \Rightarrow \langle \text{throw } e', s_2 \rangle \\
| \text{Throw}_1: \\
& P \vdash_1 \langle e, s_0 \rangle \Rightarrow \langle \text{addr } a, s_1 \rangle \Longrightarrow \\
& P \vdash_1 \langle \text{throw } e, s_0 \rangle \Rightarrow \langle \text{Throw } a, s_1 \rangle \\
| \text{ThrowNull}_1: \\
& P \vdash_1 \langle e, s_0 \rangle \Rightarrow \langle \text{null}, s_1 \rangle \Longrightarrow \\
& P \vdash_1 \langle \text{throw } e, s_0 \rangle \Rightarrow \langle \text{THROW NullPointer}, s_1 \rangle \\
| \text{ThrowThrow}_1: \\
& P \vdash_1 \langle e, s_0 \rangle \Rightarrow \langle \text{throw } e', s_1 \rangle \Longrightarrow \\
& P \vdash_1 \langle \text{throw } e, s_0 \rangle \Rightarrow \langle \text{throw } e', s_1 \rangle
\end{aligned}$$

| *Try*<sub>1</sub>:  
 $P \vdash_1 \langle e_1, s_0 \rangle \Rightarrow \langle \text{Val } v_1, s_1 \rangle \Longrightarrow$   
 $P \vdash_1 \langle \text{try } e_1 \text{ catch}(C \ i) \ e_2, s_0 \rangle \Rightarrow \langle \text{Val } v_1, s_1 \rangle$

| *TryCatch*<sub>1</sub>:  
 $\llbracket P \vdash_1 \langle e_1, s_0 \rangle \Rightarrow \langle \text{Throw } a, (h_1, ls_1, sh_1) \rangle;$   
 $h_1 \ a = \text{Some}(D, fs); P \vdash D \preceq^* C; i < \text{length } ls_1;$   
 $P \vdash_1 \langle e_2, (h_1, ls_1[i := \text{Addr } a], sh_1) \rangle \Rightarrow \langle e_2', (h_2, ls_2, sh_2) \rangle \rrbracket$   
 $\Longrightarrow P \vdash_1 \langle \text{try } e_1 \text{ catch}(C \ i) \ e_2, s_0 \rangle \Rightarrow \langle e_2', (h_2, ls_2, sh_2) \rangle$

| *TryThrow*<sub>1</sub>:  
 $\llbracket P \vdash_1 \langle e_1, s_0 \rangle \Rightarrow \langle \text{Throw } a, (h_1, ls_1, sh_1) \rangle; h_1 \ a = \text{Some}(D, fs); \neg P \vdash D \preceq^* C \rrbracket$   
 $\Longrightarrow P \vdash_1 \langle \text{try } e_1 \text{ catch}(C \ i) \ e_2, s_0 \rangle \Rightarrow \langle \text{Throw } a, (h_1, ls_1, sh_1) \rangle$

| *Nil*<sub>1</sub>:  
 $P \vdash_1 \langle [], s \rangle [\Rightarrow] \langle [], s \rangle$

| *Cons*<sub>1</sub>:  
 $\llbracket P \vdash_1 \langle e, s_0 \rangle \Rightarrow \langle \text{Val } v, s_1 \rangle; P \vdash_1 \langle es, s_1 \rangle [\Rightarrow] \langle es', s_2 \rangle \rrbracket$   
 $\Longrightarrow P \vdash_1 \langle e \# es, s_0 \rangle [\Rightarrow] \langle \text{Val } v \# es', s_2 \rangle$

| *ConsThrow*<sub>1</sub>:  
 $P \vdash_1 \langle e, s_0 \rangle \Rightarrow \langle \text{throw } e', s_1 \rangle \Longrightarrow$   
 $P \vdash_1 \langle e \# es, s_0 \rangle [\Rightarrow] \langle \text{throw } e' \# es, s_1 \rangle$

— init rules

| *InitFinal*<sub>1</sub>:  
 $P \vdash_1 \langle e, s \rangle \Rightarrow \langle e', s' \rangle \Longrightarrow P \vdash_1 \langle \text{INIT } C \ (\text{Nil}, b) \leftarrow e, s \rangle \Rightarrow \langle e', s' \rangle$

| *InitNone*<sub>1</sub>:  
 $\llbracket sh \ C = \text{None}; P \vdash_1 \langle \text{INIT } C' \ (C \# Cs, \text{False}) \leftarrow e, (h, l, sh(C \mapsto (\text{sblank } P \ C, \text{Prepared}))) \rangle \Rightarrow \langle e', s' \rangle \rrbracket$   
 $\Longrightarrow P \vdash_1 \langle \text{INIT } C' \ (C \# Cs, \text{False}) \leftarrow e, (h, l, sh) \rangle \Rightarrow \langle e', s' \rangle$

| *InitDone*<sub>1</sub>:  
 $\llbracket sh \ C = \text{Some}(sfs, \text{Done}); P \vdash_1 \langle \text{INIT } C' \ (Cs, \text{True}) \leftarrow e, (h, l, sh) \rangle \Rightarrow \langle e', s' \rangle \rrbracket$   
 $\Longrightarrow P \vdash_1 \langle \text{INIT } C' \ (C \# Cs, \text{False}) \leftarrow e, (h, l, sh) \rangle \Rightarrow \langle e', s' \rangle$

| *InitProcessing*<sub>1</sub>:  
 $\llbracket sh \ C = \text{Some}(sfs, \text{Processing}); P \vdash_1 \langle \text{INIT } C' \ (Cs, \text{True}) \leftarrow e, (h, l, sh) \rangle \Rightarrow \langle e', s' \rangle \rrbracket$   
 $\Longrightarrow P \vdash_1 \langle \text{INIT } C' \ (C \# Cs, \text{False}) \leftarrow e, (h, l, sh) \rangle \Rightarrow \langle e', s' \rangle$

| *InitError*<sub>1</sub>:  
 $\llbracket sh \ C = \text{Some}(sfs, \text{Error});$   
 $P \vdash_1 \langle \text{RI } (C, \text{THROW } \text{NoClassDefFoundError}); Cs \leftarrow e, (h, l, sh) \rangle \Rightarrow \langle e', s' \rangle \rrbracket$   
 $\Longrightarrow P \vdash_1 \langle \text{INIT } C' \ (C \# Cs, \text{False}) \leftarrow e, (h, l, sh) \rangle \Rightarrow \langle e', s' \rangle$

| *InitObject*<sub>1</sub>:  
 $\llbracket sh \ C = \text{Some}(sfs, \text{Prepared});$   
 $C = \text{Object};$   
 $sh' = sh(C \mapsto (sfs, \text{Processing}));$   
 $P \vdash_1 \langle \text{INIT } C' \ (C \# Cs, \text{True}) \leftarrow e, (h, l, sh') \rangle \Rightarrow \langle e', s' \rangle \rrbracket$   
 $\Longrightarrow P \vdash_1 \langle \text{INIT } C' \ (C \# Cs, \text{False}) \leftarrow e, (h, l, sh) \rangle \Rightarrow \langle e', s' \rangle$

| *InitNonObject*<sub>1</sub>:  
 $\llbracket sh \ C = \text{Some}(sfs, \text{Prepared});$   
 $C \neq \text{Object};$   
 $\text{class } P \ C = \text{Some } (D, r);$   
 $sh' = sh(C \mapsto (sfs, \text{Processing}));$   
 $P \vdash_1 \langle \text{INIT } C' \ (D \# C \# Cs, \text{False}) \leftarrow e, (h, l, sh') \rangle \Rightarrow \langle e', s' \rangle \rrbracket$   
 $\Longrightarrow P \vdash_1 \langle \text{INIT } C' \ (C \# Cs, \text{False}) \leftarrow e, (h, l, sh) \rangle \Rightarrow \langle e', s' \rangle$

| *InitRInit*<sub>1</sub>:

$$P \vdash_1 \langle RI (C, C \cdot_s \text{clinit}(\square)); Cs \leftarrow e, (h, l, sh) \rangle \Rightarrow \langle e', s' \rangle \\ \Longrightarrow P \vdash_1 \langle INIT C' (C \# Cs, True) \leftarrow e, (h, l, sh) \rangle \Rightarrow \langle e', s' \rangle$$

| *RInit*<sub>1</sub>:

$$\llbracket P \vdash_1 \langle e, s \rangle \Rightarrow \langle Val v, (h', l', sh') \rangle; \\ sh' C = Some(sfs, i); sh'' = sh'(C \mapsto (sfs, Done)); \\ C' = last(C \# Cs); \\ P \vdash_1 \langle INIT C' (Cs, True) \leftarrow e', (h', l', sh'') \rangle \Rightarrow \langle e_1, s_1 \rangle \rrbracket \\ \Longrightarrow P \vdash_1 \langle RI (C, e); Cs \leftarrow e', s \rangle \Rightarrow \langle e_1, s_1 \rangle$$

| *RInitInitFail*<sub>1</sub>:

$$\llbracket P \vdash_1 \langle e, s \rangle \Rightarrow \langle throw a, (h', l', sh') \rangle; \\ sh' C = Some(sfs, i); sh'' = sh'(C \mapsto (sfs, Error)); \\ P \vdash_1 \langle RI (D, throw a); Cs \leftarrow e', (h', l', sh'') \rangle \Rightarrow \langle e_1, s_1 \rangle \rrbracket \\ \Longrightarrow P \vdash_1 \langle RI (C, e); D \# Cs \leftarrow e', s \rangle \Rightarrow \langle e_1, s_1 \rangle$$

| *RInitFailFinal*<sub>1</sub>:

$$\llbracket P \vdash_1 \langle e, s \rangle \Rightarrow \langle throw a, (h', l', sh') \rangle; \\ sh' C = Some(sfs, i); sh'' = sh'(C \mapsto (sfs, Error)) \rrbracket \\ \Longrightarrow P \vdash_1 \langle RI (C, e); Nil \leftarrow e', s \rangle \Rightarrow \langle throw a, (h', l', sh'') \rangle$$

**inductive-cases** *eval*<sub>1</sub>-cases [cases set]:

$$P \vdash_1 \langle new C, s \rangle \Rightarrow \langle e', s' \rangle \\ P \vdash_1 \langle Cast C e, s \rangle \Rightarrow \langle e', s' \rangle \\ P \vdash_1 \langle Val v, s \rangle \Rightarrow \langle e', s' \rangle \\ P \vdash_1 \langle e_1 \ll \text{bop} \gg e_2, s \rangle \Rightarrow \langle e', s' \rangle \\ P \vdash_1 \langle Var v, s \rangle \Rightarrow \langle e', s' \rangle \\ P \vdash_1 \langle V := e, s \rangle \Rightarrow \langle e', s' \rangle \\ P \vdash_1 \langle e \cdot F \{D\}, s \rangle \Rightarrow \langle e', s' \rangle \\ P \vdash_1 \langle C \cdot_s F \{D\}, s \rangle \Rightarrow \langle e', s' \rangle \\ P \vdash_1 \langle e_1 \cdot F \{D\} := e_2, s \rangle \Rightarrow \langle e', s' \rangle \\ P \vdash_1 \langle C \cdot_s F \{D\} := e_2, s \rangle \Rightarrow \langle e', s' \rangle \\ P \vdash_1 \langle e \cdot M(es), s \rangle \Rightarrow \langle e', s' \rangle \\ P \vdash_1 \langle C \cdot_s M(es), s \rangle \Rightarrow \langle e', s' \rangle \\ P \vdash_1 \langle \{V:T; e_1\}, s \rangle \Rightarrow \langle e', s' \rangle \\ P \vdash_1 \langle e_1 ;; e_2, s \rangle \Rightarrow \langle e', s' \rangle \\ P \vdash_1 \langle if (e) e_1 else e_2, s \rangle \Rightarrow \langle e', s' \rangle \\ P \vdash_1 \langle while (b) c, s \rangle \Rightarrow \langle e', s' \rangle \\ P \vdash_1 \langle throw e, s \rangle \Rightarrow \langle e', s' \rangle \\ P \vdash_1 \langle try e_1 catch(C V) e_2, s \rangle \Rightarrow \langle e', s' \rangle \\ P \vdash_1 \langle INIT C (Cs, b) \leftarrow e, s \rangle \Rightarrow \langle e', s' \rangle \\ P \vdash_1 \langle RI (C, e); Cs \leftarrow e_0, s \rangle \Rightarrow \langle e', s' \rangle$$

**inductive-cases** *evals*<sub>1</sub>-cases [cases set]:

$$P \vdash_1 \langle \square, s \rangle [\Rightarrow] \langle e', s' \rangle \\ P \vdash_1 \langle e \# es, s \rangle [\Rightarrow] \langle e', s' \rangle$$

**lemma** *eval*<sub>1</sub>-final:  $P \vdash_1 \langle e, s \rangle \Rightarrow \langle e', s' \rangle \Longrightarrow \text{final } e'$

**and** *evals*<sub>1</sub>-final:  $P \vdash_1 \langle es, s \rangle [\Rightarrow] \langle es', s' \rangle \Longrightarrow \text{finals } es' \langle \text{proof} \rangle$

**lemma** *eval*<sub>1</sub>-final-same:

**assumes** *eval*:  $P \vdash_1 \langle e, s \rangle \Rightarrow \langle e', s' \rangle$  **shows** *final*  $e \Longrightarrow e = e' \wedge s = s' \langle \text{proof} \rangle$



### 3.1.1 Property preservation

**lemma** *eval<sub>1</sub>-preserves-len*:

$$P \vdash_1 \langle e_0, (h_0, ls_0, sh_0) \rangle \Rightarrow \langle e_1, (h_1, ls_1, sh_1) \rangle \Longrightarrow \text{length } ls_0 = \text{length } ls_1$$

**and** *evals<sub>1</sub>-preserves-len*:

$$P \vdash_1 \langle es_0, (h_0, ls_0, sh_0) \rangle [\Rightarrow] \langle es_1, (h_1, ls_1, sh_1) \rangle \Longrightarrow \text{length } ls_0 = \text{length } ls_1 \langle \text{proof} \rangle$$

**lemma** *evals<sub>1</sub>-preserves-elen*:

$$\bigwedge es' s s'. P \vdash_1 \langle es, s \rangle [\Rightarrow] \langle es', s' \rangle \Longrightarrow \text{length } es = \text{length } es' \langle \text{proof} \rangle$$

**lemma** *clinit<sub>1</sub>-loc-pres*:

$$P \vdash_1 \langle C_0 \cdot_s \text{clinit}(\square), (h, l, sh) \rangle \Rightarrow \langle e', (h', l', sh') \rangle \Longrightarrow l = l' \langle \text{proof} \rangle$$

**lemma**

**shows** *init<sub>1</sub>-ri<sub>1</sub>-same-loc*:  $P \vdash_1 \langle e, (h, l, sh) \rangle \Rightarrow \langle e', (h', l', sh') \rangle$

$$\begin{aligned} \Longrightarrow & (\bigwedge C Cs b M a. e = \text{INIT } C (Cs, b) \leftarrow \text{unit} \vee e = C \cdot_s M(\square) \vee e = \text{RI } (C, \text{Throw } a) ; Cs \leftarrow \text{unit} \\ & \vee e = \text{RI } (C, C \cdot_s \text{clinit}(\square)) ; Cs \leftarrow \text{unit} \\ \Longrightarrow & l = l' \end{aligned}$$

**and**  $P \vdash_1 \langle es, (h, l, sh) \rangle [\Rightarrow] \langle es', (h', l', sh') \rangle \Longrightarrow \text{True}$

$\langle \text{proof} \rangle$

**lemma** *init<sub>1</sub>-same-loc*:  $P \vdash_1 \langle \text{INIT } C (Cs, b) \leftarrow \text{unit}, (h, l, sh) \rangle \Rightarrow \langle e', (h', l', sh') \rangle \Longrightarrow l = l'$

$\langle \text{proof} \rangle$

**theorem** *eval<sub>1</sub>-hext*:  $P \vdash_1 \langle e, (h, l, sh) \rangle \Rightarrow \langle e', (h', l', sh') \rangle \Longrightarrow h \sqsubseteq h'$

**and** *evals<sub>1</sub>-hext*:  $P \vdash_1 \langle es, (h, l, sh) \rangle [\Rightarrow] \langle es', (h', l', sh') \rangle \Longrightarrow h \sqsubseteq h' \langle \text{proof} \rangle$

### 3.1.2 Initialization

**lemma** *rinit<sub>1</sub>-throw*:

$$P_1 \vdash_1 \langle \text{RI } (D, \text{Throw } xa) ; Cs \leftarrow e, (h, l, sh) \rangle \Rightarrow \langle e', (h', l', sh') \rangle \Longrightarrow e' = \text{Throw } xa$$

$\langle \text{proof} \rangle$

**lemma** *rinit<sub>1</sub>-throwE*:

$$P \vdash_1 \langle \text{RI } (C, \text{throw } e) ; Cs \leftarrow e_0, s \rangle \Rightarrow \langle e', s' \rangle \Longrightarrow \exists a s_t. e' = \text{throw } a \wedge P \vdash_1 \langle \text{throw } e, s \rangle \Rightarrow \langle \text{throw } a, s_t \rangle$$

$\langle \text{proof} \rangle$

**end**

## 3.2 Well-Formedness of Intermediate Language

**theory** *J1WellForm*

**imports** *../J/JWellForm J1*

**begin**

### 3.2.1 Well-Typedness

**type-synonym**

$env_1 = \text{ty list}$  — type environment indexed by variable number

**inductive**

$WT_1 :: [J_1\text{-prog}, env_1, expr_1, ty] \Rightarrow bool$   
 $\langle \langle -, \vdash_1 / - :: - \rangle \rangle [51, 51, 51] 50$   
**and**  $WTs_1 :: [J_1\text{-prog}, env_1, expr_1\ list, ty\ list] \Rightarrow bool$   
 $\langle \langle -, \vdash_1 / - [::] - \rangle \rangle [51, 51, 51] 50$   
**for**  $P :: J_1\text{-prog}$

**where**

$WTNew_1:$   
 $is\text{-class } P\ C \Longrightarrow$   
 $P, E \vdash_1 new\ C :: Class\ C$

$| WTCast_1:$   
 $\llbracket P, E \vdash_1 e :: Class\ D; is\text{-class } P\ C; P \vdash C \preceq^* D \vee P \vdash D \preceq^* C \rrbracket$   
 $\Longrightarrow P, E \vdash_1 Cast\ C\ e :: Class\ C$

$| WTVVal_1:$   
 $typeof\ v = Some\ T \Longrightarrow$   
 $P, E \vdash_1 Val\ v :: T$

$| WTVar_1:$   
 $\llbracket E!i = T; i < size\ E \rrbracket$   
 $\Longrightarrow P, E \vdash_1 Var\ i :: T$

$| WTBinOp_1:$   
 $\llbracket P, E \vdash_1 e_1 :: T_1; P, E \vdash_1 e_2 :: T_2;$   
 $case\ bop\ of\ Eq \Rightarrow (P \vdash T_1 \leq T_2 \vee P \vdash T_2 \leq T_1) \wedge T = Boolean$   
 $\quad | Add \Rightarrow T_1 = Integer \wedge T_2 = Integer \wedge T = Integer \rrbracket$   
 $\Longrightarrow P, E \vdash_1 e_1 \llbracket bop \rrbracket e_2 :: T$

$| WTLAss_1:$   
 $\llbracket E!i = T; i < size\ E; P, E \vdash_1 e :: T'; P \vdash T' \leq T \rrbracket$   
 $\Longrightarrow P, E \vdash_1 i := e :: Void$

$| WTFAcc_1:$   
 $\llbracket P, E \vdash_1 e :: Class\ C; P \vdash C\ sees\ F, NonStatic:T\ in\ D \rrbracket$   
 $\Longrightarrow P, E \vdash_1 e \cdot F\{D\} :: T$

$| WTSFAcc_1:$   
 $\llbracket P \vdash C\ sees\ F, Static:T\ in\ D \rrbracket$   
 $\Longrightarrow P, E \vdash_1 C \cdot_s F\{D\} :: T$

$| WTFAss_1:$   
 $\llbracket P, E \vdash_1 e_1 :: Class\ C; P \vdash C\ sees\ F, NonStatic:T\ in\ D; P, E \vdash_1 e_2 :: T'; P \vdash T' \leq T \rrbracket$   
 $\Longrightarrow P, E \vdash_1 e_1 \cdot F\{D\} := e_2 :: Void$

$| WTSFAss_1:$   
 $\llbracket P \vdash C\ sees\ F, Static:T\ in\ D; P, E \vdash_1 e_2 :: T'; P \vdash T' \leq T \rrbracket$   
 $\Longrightarrow P, E \vdash_1 C \cdot_s F\{D\} := e_2 :: Void$

$| WTCall_1:$   
 $\llbracket P, E \vdash_1 e :: Class\ C; P \vdash C\ sees\ M, NonStatic:T_s' \rightarrow T = m\ in\ D;$   
 $P, E \vdash_1 es [::] Ts; P \vdash Ts [\leq] Ts' \rrbracket$   
 $\Longrightarrow P, E \vdash_1 e \cdot M(es) :: T$

| *WTSCall*<sub>1</sub>:

$\llbracket P \vdash C \text{ sees } M, \text{Static: } Ts \rightarrow T = m \text{ in } D;$   
 $P, E \vdash_1 es \llbracket :: Ts' ; P \vdash Ts' \leq Ts ; M \neq \text{clinit} \rrbracket$   
 $\implies P, E \vdash_1 C \cdot_s M(es) \llbracket :: T$

| *WTBlock*<sub>1</sub>:

$\llbracket \text{is-type } P T ; P, E@[T] \vdash_1 e \llbracket :: T' \rrbracket$   
 $\implies P, E \vdash_1 \{i:T; e\} \llbracket :: T'$

| *WTSeq*<sub>1</sub>:

$\llbracket P, E \vdash_1 e_1 \llbracket :: T_1 ; P, E \vdash_1 e_2 \llbracket :: T_2 \rrbracket$   
 $\implies P, E \vdash_1 e_1 ; e_2 \llbracket :: T_2$

| *WTCond*<sub>1</sub>:

$\llbracket P, E \vdash_1 e \llbracket :: \text{Boolean} ; P, E \vdash_1 e_1 \llbracket :: T_1 ; P, E \vdash_1 e_2 \llbracket :: T_2 ;$   
 $P \vdash T_1 \leq T_2 \vee P \vdash T_2 \leq T_1 ; P \vdash T_1 \leq T_2 \implies T = T_2 ; P \vdash T_2 \leq T_1 \implies T = T_1 \rrbracket$   
 $\implies P, E \vdash_1 \text{if } (e) e_1 \text{ else } e_2 \llbracket :: T$

| *WTWhile*<sub>1</sub>:

$\llbracket P, E \vdash_1 e \llbracket :: \text{Boolean} ; P, E \vdash_1 c \llbracket :: T \rrbracket$   
 $\implies P, E \vdash_1 \text{while } (e) c \llbracket :: \text{Void}$

| *WTThrow*<sub>1</sub>:

$P, E \vdash_1 e \llbracket :: \text{Class } C \implies$   
 $P, E \vdash_1 \text{throw } e \llbracket :: \text{Void}$

| *WTTry*<sub>1</sub>:

$\llbracket P, E \vdash_1 e_1 \llbracket :: T ; P, E@[ \text{Class } C ] \vdash_1 e_2 \llbracket :: T ; \text{is-class } P C \rrbracket$   
 $\implies P, E \vdash_1 \text{try } e_1 \text{ catch}(C i) e_2 \llbracket :: T$

| *WTNil*<sub>1</sub>:

$P, E \vdash_1 [] \llbracket :: []$

| *WTCons*<sub>1</sub>:

$\llbracket P, E \vdash_1 e \llbracket :: T ; P, E \vdash_1 es \llbracket :: Ts \rrbracket$   
 $\implies P, E \vdash_1 e \# es \llbracket :: T \# Ts$

**lemma** *init-nWT*<sub>1</sub> [simp]:  $\neg P, E \vdash_1 \text{INIT } C (Cs, b) \leftarrow e \llbracket :: T$   
 $\langle \text{proof} \rangle$

**lemma** *rinit-nWT*<sub>1</sub> [simp]:  $\neg P, E \vdash_1 \text{RI}(C, e); Cs \leftarrow e' \llbracket :: T$   
 $\langle \text{proof} \rangle$

**lemma** *WTs<sub>1</sub>-same-size*:  $\bigwedge Ts. P, E \vdash_1 es \llbracket :: Ts \implies \text{size } es = \text{size } Ts \langle \text{proof} \rangle$

**lemma** *WT<sub>1</sub>-unique*:

$P, E \vdash_1 e \llbracket :: T_1 \implies (\bigwedge T_2. P, E \vdash_1 e \llbracket :: T_2 \implies T_1 = T_2) \text{ and}$   
 $\text{WTs}_1\text{-unique: } P, E \vdash_1 es \llbracket :: Ts_1 \implies (\bigwedge Ts_2. P, E \vdash_1 es \llbracket :: Ts_2 \implies Ts_1 = Ts_2) \langle \text{proof} \rangle$

**lemma** *assumes wf*: *wf-prog*  $p P$

**shows** *WT<sub>1</sub>-is-type*:  $P, E \vdash_1 e \llbracket :: T \implies \text{set } E \subseteq \text{types } P \implies \text{is-type } P T$

**and**  $P, E \vdash_1 es \llbracket :: Ts \implies \text{True} \langle \text{proof} \rangle$

**lemma** *WT<sub>1</sub>-nsub-RI*:  $P, E \vdash_1 e \llbracket :: T \implies \neg \text{sub-RI } e$

**and** *WTs<sub>1</sub>-nsub-RIs*:  $P, E \vdash_1 es \llbracket :: Ts \implies \neg \text{sub-RIs } es$

*<proof>*

### 3.2.2 Runtime Well-Typedness

**inductive**

$WTrt_1 :: J_1\text{-prog} \Rightarrow \text{heap} \Rightarrow \text{sheap} \Rightarrow \text{env}_1 \Rightarrow \text{expr}_1 \Rightarrow \text{ty} \Rightarrow \text{bool}$   
**and**  $WTrts_1 :: J_1\text{-prog} \Rightarrow \text{heap} \Rightarrow \text{sheap} \Rightarrow \text{env}_1 \Rightarrow \text{expr}_1 \text{ list} \Rightarrow \text{ty list} \Rightarrow \text{bool}$   
**and**  $WTrt2_1 :: [J_1\text{-prog}, \text{env}_1, \text{heap}, \text{sheap}, \text{expr}_1, \text{ty}] \Rightarrow \text{bool}$   
 $(\langle -, -, -, - \vdash_1 - : - \rangle [51, 51, 51, 51] 50)$   
**and**  $WTrts2_1 :: [J_1\text{-prog}, \text{env}_1, \text{heap}, \text{sheap}, \text{expr}_1 \text{ list}, \text{ty list}] \Rightarrow \text{bool}$   
 $(\langle -, -, -, - \vdash_1 - [:] - \rangle [51, 51, 51, 51] 50)$   
**for**  $P :: J_1\text{-prog}$  **and**  $h :: \text{heap}$  **and**  $sh :: \text{sheap}$

**where**

$P, E, h, sh \vdash_1 e : T \equiv WTrt_1 P h sh E e T$   
 $| P, E, h, sh \vdash_1 es[:] Ts \equiv WTrts_1 P h sh E es Ts$

$| WTrtNew_1:$   
 $is\text{-class } P C \implies$   
 $P, E, h, sh \vdash_1 \text{new } C : \text{Class } C$

$| WTrtCast_1:$   
 $\llbracket P, E, h, sh \vdash_1 e : T; is\text{-refT } T; is\text{-class } P C \rrbracket$   
 $\implies P, E, h, sh \vdash_1 \text{Cast } C e : \text{Class } C$

$| WTrtVal_1:$   
 $typeof_h v = \text{Some } T \implies$   
 $P, E, h, sh \vdash_1 \text{Val } v : T$

$| WTrtVar_1:$   
 $\llbracket E!i = T; i < \text{size } E \rrbracket \implies$   
 $P, E, h, sh \vdash_1 \text{Var } i : T$

$| WTrtBinOpEq_1:$   
 $\llbracket P, E, h, sh \vdash_1 e_1 : T_1; P, E, h, sh \vdash_1 e_2 : T_2 \rrbracket$   
 $\implies P, E, h, sh \vdash_1 e_1 \ll\text{Eq}\gg e_2 : \text{Boolean}$

$| WTrtBinOpAdd_1:$   
 $\llbracket P, E, h, sh \vdash_1 e_1 : \text{Integer}; P, E, h, sh \vdash_1 e_2 : \text{Integer} \rrbracket$   
 $\implies P, E, h, sh \vdash_1 e_1 \ll\text{Add}\gg e_2 : \text{Integer}$

$| WTrtLAss_1:$   
 $\llbracket E!i = T; i < \text{size } E; P, E, h, sh \vdash_1 e : T'; P \vdash T' \leq T \rrbracket$   
 $\implies P, E, h, sh \vdash_1 i := e : \text{Void}$

$| WTrtFAcc_1:$   
 $\llbracket P, E, h, sh \vdash_1 e : \text{Class } C; P \vdash C \text{ has } F, \text{NonStatic}:T \text{ in } D \rrbracket \implies$   
 $P, E, h, sh \vdash_1 e \cdot F\{D\} : T$

$| WTrtFAccNT_1:$   
 $P, E, h, sh \vdash_1 e : NT \implies$   
 $P, E, h, sh \vdash_1 e \cdot F\{D\} : T$

$| WTrtSFAcc_1:$

$$\llbracket P \vdash C \text{ has } F, \text{Static}:T \text{ in } D \rrbracket \implies \\ P, E, h, sh \vdash_1 C \cdot_s F\{D\} : T$$

| *WTrtFAss<sub>1</sub>*:  

$$\llbracket P, E, h, sh \vdash_1 e_1 : \text{Class } C; P \vdash C \text{ has } F, \text{NonStatic}:T \text{ in } D; P, E, h, sh \vdash_1 e_2 : T_2; P \vdash T_2 \leq T \rrbracket \\ \implies P, E, h, sh \vdash_1 e_1 \cdot F\{D\} := e_2 : \text{Void}$$

| *WTrtFAssNT<sub>1</sub>*:  

$$\llbracket P, E, h, sh \vdash_1 e_1 : NT; P, E, h, sh \vdash_1 e_2 : T_2 \rrbracket \\ \implies P, E, h, sh \vdash_1 e_1 \cdot F\{D\} := e_2 : \text{Void}$$

| *WTrtSFAss<sub>1</sub>*:  

$$\llbracket P, E, h, sh \vdash_1 e_2 : T_2; P \vdash C \text{ has } F, \text{Static}:T \text{ in } D; P \vdash T_2 \leq T \rrbracket \\ \implies P, E, h, sh \vdash_1 C \cdot_s F\{D\} := e_2 : \text{Void}$$

| *WTrtCall<sub>1</sub>*:  

$$\llbracket P, E, h, sh \vdash_1 e : \text{Class } C; P \vdash C \text{ sees } M, \text{NonStatic}:Ts \rightarrow T = m \text{ in } D; \\ P, E, h, sh \vdash_1 es \text{ } [ : ] Ts'; P \vdash Ts' \leq Ts \rrbracket \\ \implies P, E, h, sh \vdash_1 e \cdot M(es) : T$$

| *WTrtCallNT<sub>1</sub>*:  

$$\llbracket P, E, h, sh \vdash_1 e : NT; P, E, h, sh \vdash_1 es \text{ } [ : ] Ts \rrbracket \\ \implies P, E, h, sh \vdash_1 e \cdot M(es) : T$$

| *WTrtSCall<sub>1</sub>*:  

$$\llbracket P \vdash C \text{ sees } M, \text{Static}:Ts \rightarrow T = m \text{ in } D; \\ P, E, h, sh \vdash_1 es \text{ } [ : ] Ts'; P \vdash Ts' \leq Ts; \\ M = \text{clinit} \longrightarrow sh D = [(sfs, Processing)] \wedge es = \text{map Val vs} \rrbracket \\ \implies P, E, h, sh \vdash_1 C \cdot_s M(es) : T$$

| *WTrtBlock<sub>1</sub>*:  

$$P, E@[T], h, sh \vdash_1 e : T' \implies \\ P, E, h, sh \vdash_1 \{i:T; e\} : T'$$

| *WTrtSeq<sub>1</sub>*:  

$$\llbracket P, E, h, sh \vdash_1 e_1 : T_1; P, E, h, sh \vdash_1 e_2 : T_2 \rrbracket \\ \implies P, E, h, sh \vdash_1 e_1 ;; e_2 : T_2$$

| *WTrtCond<sub>1</sub>*:  

$$\llbracket P, E, h, sh \vdash_1 e : \text{Boolean}; P, E, h, sh \vdash_1 e_1 : T_1; P, E, h, sh \vdash_1 e_2 : T_2; \\ P \vdash T_1 \leq T_2 \vee P \vdash T_2 \leq T_1; P \vdash T_1 \leq T_2 \longrightarrow T = T_2; P \vdash T_2 \leq T_1 \longrightarrow T = T_1 \rrbracket \\ \implies P, E, h, sh \vdash_1 \text{if } (e) e_1 \text{ else } e_2 : T$$

| *WTrtWhile<sub>1</sub>*:  

$$\llbracket P, E, h, sh \vdash_1 e : \text{Boolean}; P, E, h, sh \vdash_1 c : T \rrbracket \\ \implies P, E, h, sh \vdash_1 \text{while}(e) c : \text{Void}$$

| *WTrtThrow<sub>1</sub>*:  

$$\llbracket P, E, h, sh \vdash_1 e : T_r; \text{is-refT } T_r \rrbracket \implies \\ P, E, h, sh \vdash_1 \text{throw } e : T$$

| *WTrtTry<sub>1</sub>*:  

$$\llbracket P, E, h, sh \vdash_1 e_1 : T_1; P, E@[Class C], h, sh \vdash_1 e_2 : T_2; P \vdash T_1 \leq T_2 \rrbracket \\ \implies P, E, h, sh \vdash_1 \text{try } e_1 \text{ catch}(C i) e_2 : T_2$$

| *WTrtInit*<sub>1</sub>:  
 $\llbracket P, E, h, sh \vdash_1 e : T; \forall C' \in \text{set } (C \# Cs). \text{is-class } P \ C'; \neg \text{sub-RI } e;$   
 $\quad \forall C' \in \text{set } (tl \ Cs). \exists \text{sfs. sh } C' = \llbracket (sfs, Processing) \rrbracket;$   
 $\quad b \longrightarrow (\forall C' \in \text{set } Cs. \exists \text{sfs. sh } C' = \llbracket (sfs, Processing) \rrbracket);$   
 $\quad \text{distinct } Cs; \text{supercls-lst } P \ Cs \rrbracket$   
 $\implies P, E, h, sh \vdash_1 \text{INIT } C \ (Cs, b) \leftarrow e : T$

| *WTrtRI*<sub>1</sub>:  
 $\llbracket P, E, h, sh \vdash_1 e : T; P, E, h, sh \vdash_1 e' : T'; \forall C' \in \text{set } (C \# Cs). \text{is-class } P \ C'; \neg \text{sub-RI } e';$   
 $\quad \forall C' \in \text{set } (C \# Cs). \text{not-init } C' \ e';$   
 $\quad \forall C' \in \text{set } Cs. \exists \text{sfs. sh } C' = \llbracket (sfs, Processing) \rrbracket;$   
 $\quad \exists \text{sfs. sh } C = \llbracket (sfs, Processing) \rrbracket \vee (sh \ C = \llbracket (sfs, Error) \rrbracket \wedge e = \text{THROW NoClassDefFoundError});$   
 $\quad \text{distinct } (C \# Cs); \text{supercls-lst } P \ (C \# Cs) \rrbracket$   
 $\implies P, E, h, sh \vdash_1 \text{RI}(C, e); Cs \leftarrow e' : T'$

— well-typed expression lists

| *WTrtNil*<sub>1</sub>:  
 $P, E, h, sh \vdash_1 [] \ [:] \ []$

| *WTrtCons*<sub>1</sub>:  
 $\llbracket P, E, h, sh \vdash_1 e : T; P, E, h, sh \vdash_1 es \ [:] \ Ts \rrbracket$   
 $\implies P, E, h, sh \vdash_1 e \# es \ [:] \ T \# Ts$

**lemma** *WT*<sub>1</sub>-*implies-WTrt*<sub>1</sub>:  $P, E \vdash_1 e :: T \implies P, E, h, sh \vdash_1 e : T$   
**and** *WT*<sub>1</sub>-*implies-WTrts*<sub>1</sub>:  $P, E \vdash_1 es \ [::] \ Ts \implies P, E, h, sh \vdash_1 es \ [:] \ Ts$  *<proof>*

### 3.2.3 Well-formedness

**primrec**  $\mathcal{B} :: \text{expr}_1 \Rightarrow \text{nat} \Rightarrow \text{bool}$   
**and**  $\mathcal{B}s :: \text{expr}_1 \ \text{list} \Rightarrow \text{nat} \Rightarrow \text{bool}$  **where**

$\mathcal{B} \ (\text{new } C) \ i = \text{True} \mid$   
 $\mathcal{B} \ (\text{Cast } C \ e) \ i = \mathcal{B} \ e \ i \mid$   
 $\mathcal{B} \ (\text{Val } v) \ i = \text{True} \mid$   
 $\mathcal{B} \ (e_1 \ \ll\text{bop}\ e_2) \ i = (\mathcal{B} \ e_1 \ i \wedge \mathcal{B} \ e_2 \ i) \mid$   
 $\mathcal{B} \ (\text{Var } j) \ i = \text{True} \mid$   
 $\mathcal{B} \ (e \cdot F\{D\}) \ i = \mathcal{B} \ e \ i \mid$   
 $\mathcal{B} \ (C \cdot_s F\{D\}) \ i = \text{True} \mid$   
 $\mathcal{B} \ (j := e) \ i = \mathcal{B} \ e \ i \mid$   
 $\mathcal{B} \ (e_1 \cdot F\{D\} := e_2) \ i = (\mathcal{B} \ e_1 \ i \wedge \mathcal{B} \ e_2 \ i) \mid$   
 $\mathcal{B} \ (C \cdot_s F\{D\} := e_2) \ i = \mathcal{B} \ e_2 \ i \mid$   
 $\mathcal{B} \ (e \cdot M(es)) \ i = (\mathcal{B} \ e \ i \wedge \mathcal{B}s \ es \ i) \mid$   
 $\mathcal{B} \ (C \cdot_s M(es)) \ i = \mathcal{B}s \ es \ i \mid$   
 $\mathcal{B} \ (\{j:T ; e\}) \ i = (i = j \wedge \mathcal{B} \ e \ (i+1)) \mid$   
 $\mathcal{B} \ (e_1 ;; e_2) \ i = (\mathcal{B} \ e_1 \ i \wedge \mathcal{B} \ e_2 \ i) \mid$   
 $\mathcal{B} \ (\text{if } (e) \ e_1 \ \text{else } e_2) \ i = (\mathcal{B} \ e \ i \wedge \mathcal{B} \ e_1 \ i \wedge \mathcal{B} \ e_2 \ i) \mid$   
 $\mathcal{B} \ (\text{throw } e) \ i = \mathcal{B} \ e \ i \mid$   
 $\mathcal{B} \ (\text{while } (e) \ c) \ i = (\mathcal{B} \ e \ i \wedge \mathcal{B} \ c \ i) \mid$   
 $\mathcal{B} \ (\text{try } e_1 \ \text{catch}(C \ j) \ e_2) \ i = (\mathcal{B} \ e_1 \ i \wedge i=j \wedge \mathcal{B} \ e_2 \ (i+1)) \mid$   
 $\mathcal{B} \ (\text{INIT } C \ (Cs, b) \leftarrow e) \ i = \mathcal{B} \ e \ i \mid$   
 $\mathcal{B} \ (\text{RI}(C, e); Cs \leftarrow e') \ i = (\mathcal{B} \ e \ i \wedge \mathcal{B} \ e' \ i) \mid$

$\mathcal{B}s \ [] \ i = \text{True} \mid$

$\mathcal{B}s (e\#es) i = (\mathcal{B} e i \wedge \mathcal{B}s es i)$

**definition**  $wf\text{-}J_1\text{-mdecl} :: J_1\text{-prog} \Rightarrow cname \Rightarrow expr_1\ mdecl \Rightarrow bool$

**where**

$wf\text{-}J_1\text{-mdecl} P C \equiv \lambda(M,b,Ts,T,body).$

$\neg sub\text{-}RI\ body \wedge$

(case b of

NonStatic  $\Rightarrow$

$(\exists T'. P, Class C\#Ts \vdash_1 body :: T' \wedge P \vdash T' \leq T) \wedge$

$\mathcal{D} body [\{..size\ Ts\}] \wedge \mathcal{B} body (size\ Ts + 1)$

| Static  $\Rightarrow (\exists T'. P, Ts \vdash_1 body :: T' \wedge P \vdash T' \leq T) \wedge$

$\mathcal{D} body [\{..<size\ Ts\}] \wedge \mathcal{B} body (size\ Ts)$ )

**lemma**  $wf\text{-}J_1\text{-mdecl}\text{-}NonStatic[simp]$ :

$wf\text{-}J_1\text{-mdecl} P C (M, NonStatic, Ts, T, body) \equiv$

$(\neg sub\text{-}RI\ body \wedge$

$(\exists T'. P, Class C\#Ts \vdash_1 body :: T' \wedge P \vdash T' \leq T) \wedge$

$\mathcal{D} body [\{..size\ Ts\}] \wedge \mathcal{B} body (size\ Ts + 1)) \langle proof \rangle$

**lemma**  $wf\text{-}J_1\text{-mdecl}\text{-}Static[simp]$ :

$wf\text{-}J_1\text{-mdecl} P C (M, Static, Ts, T, body) \equiv$

$(\neg sub\text{-}RI\ body \wedge$

$(\exists T'. P, Ts \vdash_1 body :: T' \wedge P \vdash T' \leq T) \wedge$

$\mathcal{D} body [\{..<size\ Ts\}] \wedge \mathcal{B} body (size\ Ts)) \langle proof \rangle$

**abbreviation**  $wf\text{-}J_1\text{-prog} == wf\text{-}prog\ wf\text{-}J_1\text{-mdecl}$

**lemma**  $sees\text{-}wf_1\text{-}nsub\text{-}RI$ :

**assumes**  $wf: wf\text{-}J_1\text{-prog} P$  **and**  $cM: P \vdash C\ sees\ M, b : Ts \rightarrow T = body$  in  $D$

**shows**  $\neg sub\text{-}RI\ body$

$\langle proof \rangle$

**lemma**  $wf_1\text{-}types\text{-}clinit$ :

**assumes**  $wf: wf\text{-}prog\ wf\text{-}md\ P$  **and**  $ex: class\ P\ C = Some\ a$  **and**  $proc: sh\ C = [(sfs, Processing)]$

**shows**  $P, E, h, sh \vdash_1 C\cdot_s\ clinit(\[]) : Void$

$\langle proof \rangle$

**lemma** **assumes**  $wf: wf\text{-}J_1\text{-prog} P$

**shows**  $eval_1\text{-}proc\text{-}pres: P \vdash_1 \langle e, (h, l, sh) \rangle \Rightarrow \langle e', (h', l', sh') \rangle$

$\Rightarrow not\text{-}init\ C\ e \Rightarrow \exists sfs. sh\ C = [(sfs, Processing)] \Rightarrow \exists sfs'. sh'\ C = [(sfs', Processing)]$

**and**  $evals_1\text{-}proc\text{-}pres: P \vdash_1 \langle es, (h, l, sh) \rangle [\Rightarrow] \langle es', (h', l', sh') \rangle$

$\Rightarrow not\text{-}inits\ C\ es \Rightarrow \exists sfs. sh\ C = [(sfs, Processing)] \Rightarrow \exists sfs'. sh'\ C = [(sfs', Processing)] \langle proof \rangle$

**lemma**  $clinit_1\text{-}proc\text{-}pres$ :

$\llbracket wf\text{-}J_1\text{-prog} P; P \vdash_1 \langle C_0\cdot_s\ clinit(\[]), (h, l, sh) \rangle \Rightarrow \langle e', (h', l', sh') \rangle;$

$sh\ C' = [(sfs, Processing)] \rrbracket$

$\Rightarrow \exists sfs. sh'\ C' = [(sfs, Processing)]$

$\langle proof \rangle$

**end**

### 3.3 Program Compilation

**theory** PCompiler

**imports** ../Common/WellForm  
**begin**

**definition** *compM* :: (staticb ⇒ 'a ⇒ 'b) ⇒ 'a mdecl ⇒ 'b mdecl  
**where**

$$\text{compM } f \equiv \lambda(M, b, Ts, T, m). (M, b, Ts, T, f \text{ b } m)$$

**definition** *compC* :: (staticb ⇒ 'a ⇒ 'b) ⇒ 'a cdecl ⇒ 'b cdecl  
**where**

$$\text{compC } f \equiv \lambda(C, D, Fdecls, Mdecls). (C, D, Fdecls, \text{map } (\text{compM } f) \text{ Mdecls})$$

**definition** *compP* :: (staticb ⇒ 'a ⇒ 'b) ⇒ 'a prog ⇒ 'b prog  
**where**

$$\text{compP } f \equiv \text{map } (\text{compC } f)$$

Compilation preserves the program structure. Therefore lookup functions either commute with compilation (like method lookup) or are preserved by it (like the subclass relation).

**lemma** *map-of-map4*:

$$\begin{aligned} \text{map-of } (\text{map } (\lambda(x, a, b, c). (x, a, b, f \text{ c})) \text{ ts}) &= \\ \text{map-option } (\lambda(a, b, c). (a, b, f \text{ c})) \circ (\text{map-of ts}) &\langle \text{proof} \rangle \end{aligned}$$

**lemma** *map-of-map245*:

$$\begin{aligned} \text{map-of } (\text{map } (\lambda(x, a, b, c, d). (x, a, b, c, f \text{ a } c \text{ d})) \text{ ts}) &= \\ \text{map-option } (\lambda(a, b, c, d). (a, b, c, f \text{ a } c \text{ d})) \circ (\text{map-of ts}) &\langle \text{proof} \rangle \end{aligned}$$

**lemma** *class-compP*:

$$\begin{aligned} \text{class } P \text{ } C &= \text{Some } (D, fs, ms) \\ \implies \text{class } (\text{compP } f \text{ } P) \text{ } C &= \text{Some } (D, fs, \text{map } (\text{compM } f) \text{ ms}) \langle \text{proof} \rangle \end{aligned}$$

**lemma** *class-compPD*:

$$\begin{aligned} \text{class } (\text{compP } f \text{ } P) \text{ } C &= \text{Some } (D, fs, cms) \\ \implies \exists ms. \text{class } P \text{ } C &= \text{Some}(D, fs, ms) \wedge cms = \text{map } (\text{compM } f) \text{ ms} \langle \text{proof} \rangle \end{aligned}$$

**lemma** [*simp*]: *is-class* (compP f P) C = *is-class* P C ⟨proof⟩

**lemma** [*simp*]: *class* (compP f P) C = *map-option* (λc. *snd*(compC f (C, c))) (class P C) ⟨proof⟩

**lemma** *sees-methods-compP*:

$$\begin{aligned} P \vdash C \text{ sees-methods } Mm &\implies \\ \text{compP } f \text{ } P \vdash C \text{ sees-methods } &(\text{map-option } (\lambda((b, Ts, T, m), D). ((b, Ts, T, f \text{ b } m), D))) \circ Mm \langle \text{proof} \rangle \end{aligned}$$

**lemma** *sees-method-compP*:

$$\begin{aligned} P \vdash C \text{ sees } M, b: Ts \rightarrow T = m \text{ in } D &\implies \\ \text{compP } f \text{ } P \vdash C \text{ sees } M, b: Ts \rightarrow T = &(f \text{ b } m) \text{ in } D \langle \text{proof} \rangle \end{aligned}$$

**lemma** [*simp*]:

$$\begin{aligned} P \vdash C \text{ sees } M, b: Ts \rightarrow T = m \text{ in } D &\implies \\ \text{method } (\text{compP } f \text{ } P) \text{ } C \text{ } M &= (D, b, Ts, T, f \text{ b } m) \langle \text{proof} \rangle \end{aligned}$$

**lemma** *sees-methods-compPD*:

$$\begin{aligned} \llbracket cP \vdash C \text{ sees-methods } Mm'; cP = \text{compP } f \text{ } P \rrbracket &\implies \\ \exists Mm. P \vdash C \text{ sees-methods } Mm \wedge & \\ Mm' = (\text{map-option } (\lambda((b, Ts, T, m), D). &((b, Ts, T, f \text{ b } m), D))) \circ Mm \langle \text{proof} \rangle \end{aligned}$$

**lemma** *sees-method-compPD*:



$compP f P \vdash C \text{ sees } M, b: Ts \rightarrow T = fm \text{ in } D \implies$   
 $\exists m. P \vdash C \text{ sees } M, b: Ts \rightarrow T = m \text{ in } D \wedge f b m = fm \langle proof \rangle$

**lemma**  $[simp]: subcls1 (compP f P) = subcls1 P \langle proof \rangle$

**lemma**  $compP\text{-widen}[simp]: (compP f P \vdash T \leq T') = (P \vdash T \leq T') \langle proof \rangle$

**lemma**  $[simp]: (compP f P \vdash Ts [\leq] Ts') = (P \vdash Ts [\leq] Ts') \langle proof \rangle$

**lemma**  $[simp]: is\text{-type} (compP f P) T = is\text{-type} P T \langle proof \rangle$

**lemma**  $[simp]: (compP (f::staticb \Rightarrow 'a \Rightarrow 'b) P \vdash C \text{ has-fields } FDTs) = (P \vdash C \text{ has-fields } FDTs) \langle proof \rangle$

**lemma**  $fields\text{-compP} [simp]: fields (compP f P) C = fields P C \langle proof \rangle$

**lemma**  $ifields\text{-compP} [simp]: ifields (compP f P) C = ifields P C \langle proof \rangle$

**lemma**  $blank\text{-compP} [simp]: blank (compP f P) C = blank P C \langle proof \rangle$

**lemma**  $isfields\text{-compP} [simp]: isfields (compP f P) C = isfields P C \langle proof \rangle$

**lemma**  $sblank\text{-compP} [simp]: sblank (compP f P) C = sblank P C \langle proof \rangle$

**lemma**  $sees\text{-fields}\text{-compP} [simp]: (compP f P \vdash C \text{ sees } F, b: T \text{ in } D) = (P \vdash C \text{ sees } F, b: T \text{ in } D) \langle proof \rangle$

**lemma**  $has\text{-field}\text{-compP} [simp]: (compP f P \vdash C \text{ has } F, b: T \text{ in } D) = (P \vdash C \text{ has } F, b: T \text{ in } D) \langle proof \rangle$

**lemma**  $field\text{-compP} [simp]: field (compP f P) F D = field P F D \langle proof \rangle$

### 3.3.1 Invariance of *wf-prog* under compilation

**lemma**  $[iff]: distinct\text{-fst} (compP f P) = distinct\text{-fst} P \langle proof \rangle$

**lemma**  $[iff]: distinct\text{-fst} (map (compM f) ms) = distinct\text{-fst} ms \langle proof \rangle$

**lemma**  $[iff]: wf\text{-syscls} (compP f P) = wf\text{-syscls} P \langle proof \rangle$

**lemma**  $[iff]: wf\text{-fdecl} (compP f P) = wf\text{-fdecl} P \langle proof \rangle$

**lemma**  $wf\text{-clinit}\text{-compM} [iff]: wf\text{-clinit} (map (compM f) ms) = wf\text{-clinit} ms \langle proof \rangle$

**lemma**  $set\text{-compP}$ :

$((C, D, fs, ms') \in set (compP f P)) =$   
 $(\exists ms. (C, D, fs, ms) \in set P \wedge ms' = map (compM f) ms) \langle proof \rangle$

**lemma**  $wf\text{-cdecl}\text{-compPI}$ :

$\llbracket \bigwedge C M b Ts T m.$   
 $\llbracket wf\text{-mdecl} wf_1 P C (M, b, Ts, T, m); P \vdash C \text{ sees } M, b: Ts \rightarrow T = m \text{ in } C \rrbracket$   
 $\implies wf\text{-mdecl} wf_2 (compP f P) C (M, b, Ts, T, f b m);$   
 $\forall x \in set P. wf\text{-cdecl} wf_1 P x; x \in set (compP f P); wf\text{-prog} p P \rrbracket$   
 $\implies wf\text{-cdecl} wf_2 (compP f P) x \langle proof \rangle$

**lemma**  $wf\text{-prog}\text{-compPI}$ :

**assumes**  $lift$ :

$\bigwedge C M b Ts T m.$   
 $\llbracket P \vdash C \text{ sees } M, b: Ts \rightarrow T = m \text{ in } C; wf\text{-mdecl} wf_1 P C (M, b, Ts, T, m) \rrbracket$   
 $\implies wf\text{-mdecl} wf_2 (compP f P) C (M, b, Ts, T, f b m)$

**and**  $wf: wf\text{-prog} wf_1 P$

**shows**  $wf\text{-prog} wf_2 (compP f P) \langle proof \rangle$

**end**

**theory** *Hidden*

**imports** *List-Index.List-Index*

**begin**

**definition** *hidden* :: 'a list  $\Rightarrow$  nat  $\Rightarrow$  bool **where**  
*hidden* xs i  $\equiv$  i < size xs  $\wedge$  xs!i  $\in$  set(drop (i+1) xs)

**lemma** *hidden-last-index*: x  $\in$  set xs  $\Longrightarrow$  *hidden* (xs @ [x]) (last-index xs x)  
 <proof>

**lemma** *hidden-inacc*: *hidden* xs i  $\Longrightarrow$  last-index xs x  $\neq$  i  
 <proof>

**lemma** [*simp*]: *hidden* xs i  $\Longrightarrow$  *hidden* (xs@[x]) i  
 <proof>

**lemma** *fun-upds-apply*:  
 (m(xs[ $\mapsto$ ]ys)) x =  
 (let xs' = take (size ys) xs  
 in if x  $\in$  set xs' then Some(ys ! last-index xs' x) else m x)  
 <proof>

**lemma** *map-upds-apply-eq-Some*:  
 ((m(xs[ $\mapsto$ ]ys)) x = Some y) =  
 (let xs' = take (size ys) xs  
 in if x  $\in$  set xs' then ys ! last-index xs' x = y else m x = Some y)  
 <proof>

**lemma** *map-upds-upd-conv-last-index*:  
 [x  $\in$  set xs; size xs  $\leq$  size ys ]  
 $\Longrightarrow$  m(xs[ $\mapsto$ ]ys, x $\mapsto$ y) = m(xs[ $\mapsto$ ]ys[last-index xs x := y])  
 <proof>

**end**

### 3.4 Compilation Stage 1

**theory** *Compiler1* **imports** *PCompiler J1 Hidden* **begin**

Replacing variable names by indices.

**primrec** *compE<sub>1</sub>* :: vname list  $\Rightarrow$  expr  $\Rightarrow$  expr<sub>1</sub>  
**and** *compEs<sub>1</sub>* :: vname list  $\Rightarrow$  expr list  $\Rightarrow$  expr<sub>1</sub> list **where**  
*compE<sub>1</sub>* Vs (new C) = new C  
 | *compE<sub>1</sub>* Vs (Cast C e) = Cast C (compE<sub>1</sub> Vs e)  
 | *compE<sub>1</sub>* Vs (Val v) = Val v  
 | *compE<sub>1</sub>* Vs (e<sub>1</sub> «bop» e<sub>2</sub>) = (compE<sub>1</sub> Vs e<sub>1</sub>) «bop» (compE<sub>1</sub> Vs e<sub>2</sub>)  
 | *compE<sub>1</sub>* Vs (Var V) = Var(last-index Vs V)  
 | *compE<sub>1</sub>* Vs (V:=e) = (last-index Vs V):= (compE<sub>1</sub> Vs e)  
 | *compE<sub>1</sub>* Vs (e.F{D}) = (compE<sub>1</sub> Vs e).F{D}  
 | *compE<sub>1</sub>* Vs (C.sF{D}) = C.sF{D}

```

| compE1 Vs (e1·F{D}:=e2) = (compE1 Vs e1)·F{D} := (compE1 Vs e2)
| compE1 Vs (C·sF{D}:=e2) = C·sF{D} := (compE1 Vs e2)
| compE1 Vs (e·M(es)) = (compE1 Vs e)·M(compEs1 Vs es)
| compE1 Vs (C·sM(es)) = C·sM(compEs1 Vs es)
| compE1 Vs {V:T; e} = {(size Vs):T; compE1 (Vs@[V]) e}
| compE1 Vs (e1;;e2) = (compE1 Vs e1);;(compE1 Vs e2)
| compE1 Vs (if (e) e1 else e2) = if (compE1 Vs e) (compE1 Vs e1) else (compE1 Vs e2)
| compE1 Vs (while (e) c) = while (compE1 Vs e) (compE1 Vs c)
| compE1 Vs (throw e) = throw (compE1 Vs e)
| compE1 Vs (try e1 catch(C V) e2) =
  try(compE1 Vs e1) catch(C (size Vs)) (compE1 (Vs@[V]) e2)
| compE1 Vs (INIT C (Cs,b) ← e) = INIT C (Cs,b) ← (compE1 Vs e)
| compE1 Vs (RI(C,e);Cs ← e') = RI(C,(compE1 Vs e));Cs ← (compE1 Vs e')

| compEs1 Vs [] = []
| compEs1 Vs (e#es) = compE1 Vs e # compEs1 Vs es

```

**lemma** [simp]:  $\text{compEs1 Vs es} = \text{map (compE1 Vs) es} \langle \text{proof} \rangle$

**lemma** [simp]:  $\bigwedge \text{Vs. sub-RI (compE1 Vs e)} = \text{sub-RI e}$   
**and** [simp]:  $\bigwedge \text{Vs. sub-RIs (compEs1 Vs es)} = \text{sub-RIs es}$   
 $\langle \text{proof} \rangle$

**primrec**  $\text{fin}_1 :: \text{expr} \Rightarrow \text{expr}_1$  **where**

```

  fin1 (Val v) = Val v
| fin1 (throw e) = throw (fin1 e)

```

**lemma**  $\text{comp-final}$ :  $\text{final e} \Longrightarrow \text{compE1 Vs e} = \text{fin}_1 e \langle \text{proof} \rangle$

**lemma** [simp]:

$\bigwedge \text{Vs. max-vars (compE1 Vs e)} = \text{max-vars e}$   
**and**  $\bigwedge \text{Vs. max-varss (compEs1 Vs es)} = \text{max-varss es} \langle \text{proof} \rangle$

Compiling programs:

**definition**  $\text{compP}_1 :: J\text{-prog} \Rightarrow J_1\text{-prog}$

**where**

$\text{compP}_1 \equiv \text{compP } (\lambda b \text{ (pns,body). compE1 (case b of NonStatic} \Rightarrow \text{this\#pns} \mid \text{Static} \Rightarrow \text{pns}) \text{ body})$

**end**

## 3.5 Correctness of Stage 1

**theory** *Correctness1*

**imports** *J1WellForm Compiler1*

**begin**

### 3.5.1 Correctness of program compilation

**primrec**  $\text{unmod} :: \text{expr}_1 \Rightarrow \text{nat} \Rightarrow \text{bool}$

**and**  $\text{unmods} :: \text{expr}_1 \text{ list} \Rightarrow \text{nat} \Rightarrow \text{bool}$  **where**

```

unmod (new C) i = True |
unmod (Cast C e) i = unmod e i |
unmod (Val v) i = True |
unmod (e1 «bop» e2) i = (unmod e1 i ∧ unmod e2 i) |
unmod (Var i) j = True |

```

$unmod (i:=e) j = (i \neq j \wedge unmod e j) \mid$   
 $unmod (e \cdot F\{D\}) i = unmod e i \mid$   
 $unmod (C \cdot_s F\{D\}) i = True \mid$   
 $unmod (e_1 \cdot F\{D\} := e_2) i = (unmod e_1 i \wedge unmod e_2 i) \mid$   
 $unmod (C \cdot_s F\{D\} := e_2) i = unmod e_2 i \mid$   
 $unmod (e \cdot M(es)) i = (unmod e i \wedge unmods es i) \mid$   
 $unmod (C \cdot_s M(es)) i = unmods es i \mid$   
 $unmod \{j:T; e\} i = unmod e i \mid$   
 $unmod (e_1;;e_2) i = (unmod e_1 i \wedge unmod e_2 i) \mid$   
 $unmod (if (e) e_1 else e_2) i = (unmod e i \wedge unmod e_1 i \wedge unmod e_2 i) \mid$   
 $unmod (while (e) c) i = (unmod e i \wedge unmod c i) \mid$   
 $unmod (throw e) i = unmod e i \mid$   
 $unmod (try e_1 catch (C i) e_2) j = (unmod e_1 j \wedge (if i=j then False else unmod e_2 j)) \mid$   
 $unmod (INIT C (Cs,b) \leftarrow e) i = unmod e i \mid$   
 $unmod (RI(C,e);Cs \leftarrow e') i = (unmod e i \wedge unmod e' i) \mid$

$unmods (\Box) i = True \mid$   
 $unmods (e\#es) i = (unmod e i \wedge unmods es i)$

**lemma** *hidden-unmod*:  $\bigwedge Vs. hidden\ Vs\ i \implies unmod (compE_1\ Vs\ e)\ i$  and  $\bigwedge Vs. hidden\ Vs\ i \implies unmods (compEs_1\ Vs\ es)\ i$  *proof*

**lemma** *eval<sub>1</sub>-preserves-unmod*:

$\llbracket P \vdash_1 \langle e, (h, ls, sh) \rangle \Rightarrow \langle e', (h', ls', sh') \rangle; unmod\ e\ i; i < size\ ls \rrbracket$   
 $\implies ls\ !\ i = ls'\ !\ i$

**and**  $\llbracket P \vdash_1 \langle es, (h, ls, sh) \rangle [\Rightarrow] \langle es', (h', ls', sh') \rangle; unmods\ es\ i; i < size\ ls \rrbracket$   
 $\implies ls\ !\ i = ls'\ !\ i$  *proof*

**lemma** *LAss-lem*:

$\llbracket x \in set\ xs; size\ xs \leq size\ ys \rrbracket$

$\implies m_1 \subseteq_m m_2 (xs \mapsto ys) \implies m_1 (x \mapsto y) \subseteq_m m_2 (xs \mapsto ys [last-index\ xs\ x := y])$  *proof*

**lemma** *Block-lem*:

**fixes**  $l :: 'a \rightarrow 'b$

**assumes**  $0: l \subseteq_m [Vs\ [\mapsto]\ ls]$

**and**  $1: l' \subseteq_m [Vs\ [\mapsto]\ ls', V \mapsto v]$

**and** *hidden*:  $V \in set\ Vs \implies ls\ !\ last-index\ Vs\ V = ls'\ !\ last-index\ Vs\ V$

**and** *size*:  $size\ ls = size\ ls' \quad size\ Vs < size\ ls'$

**shows**  $l'(V := l\ V) \subseteq_m [Vs\ [\mapsto]\ ls']$  *proof*

The main theorem:

**theorem** *assumes* *wf*: *wuf-J-prog*  $P$

**shows** *eval<sub>1</sub>-eval*:  $P \vdash \langle e, (h, l, sh) \rangle \Rightarrow \langle e', (h', l', sh') \rangle$

$\implies (\bigwedge Vs\ ls. \llbracket fv\ e \subseteq set\ Vs; l \subseteq_m [Vs[\mapsto]ls]; size\ Vs + max-vars\ e \leq size\ ls \rrbracket$

$\implies \exists ls'. compP_1\ P \vdash_1 \langle compE_1\ Vs\ e, (h, ls, sh) \rangle \Rightarrow \langle fin_1\ e', (h', ls', sh') \rangle \wedge l' \subseteq_m [Vs[\mapsto]ls']$ )

**and** *evals<sub>1</sub>-evals*:  $P \vdash \langle es, (h, l, sh) \rangle [\Rightarrow] \langle es', (h', l', sh') \rangle$

$\implies (\bigwedge Vs\ ls. \llbracket fvs\ es \subseteq set\ Vs; l \subseteq_m [Vs[\mapsto]ls]; size\ Vs + max-varss\ es \leq size\ ls \rrbracket$

$\implies \exists ls'. compP_1\ P \vdash_1 \langle compEs_1\ Vs\ es, (h, ls, sh) \rangle [\Rightarrow] \langle compEs_1\ Vs\ es', (h', ls', sh') \rangle \wedge l' \subseteq_m [Vs[\mapsto]ls']$ ) *proof*

### 3.5.2 Preservation of well-formedness

The compiler preserves well-formedness. Is less trivial than it may appear. We start with two simple properties: preservation of well-typedness

**lemma** *compE<sub>1</sub>-pres-wt*:  $\bigwedge Vs\ Ts\ U.$

$\llbracket P, [Vs[\mapsto]Ts] \vdash e :: U; size\ Ts = size\ Vs \rrbracket$

$\implies \text{comp}P f P, Ts \vdash_1 \text{comp}E_1 Vs e :: U$   
**and**  $\bigwedge Vs Ts Us.$   
 $\llbracket P, [Vs \mapsto Ts] \vdash es [::] Us; \text{size } Ts = \text{size } Vs \rrbracket$   
 $\implies \text{comp}P f P, Ts \vdash_1 \text{comp}Es_1 Vs es [::] Us \langle \text{proof} \rangle$

and the correct block numbering:

**lemma**  $\mathcal{B}$ :  $\bigwedge Vs n. \text{size } Vs = n \implies \mathcal{B} (\text{comp}E_1 Vs e) n$   
**and**  $\mathcal{B}s$ :  $\bigwedge Vs n. \text{size } Vs = n \implies \mathcal{B}s (\text{comp}Es_1 Vs es) n \langle \text{proof} \rangle$

The main complication is preservation of definite assignment  $\mathcal{D}$ .

**lemma** *image-last-index*:  $A \subseteq \text{set}(xs@[x]) \implies \text{last-index } (xs @ [x]) \text{ ' } A =$   
*(if*  $x \in A$  *then*  $\text{insert } (\text{size } xs) (\text{last-index } xs \text{ ' } (A - \{x\}))$  *else*  $\text{last-index } xs \text{ ' } A$ *)*  $\langle \text{proof} \rangle$

**lemma** *A-compE1-None[simp]*:  
 $\bigwedge Vs. \mathcal{A} e = \text{None} \implies \mathcal{A} (\text{comp}E_1 Vs e) = \text{None}$   
**and**  $\bigwedge Vs. \mathcal{A}s es = \text{None} \implies \mathcal{A}s (\text{comp}Es_1 Vs es) = \text{None} \langle \text{proof} \rangle$

**lemma** *A-compE1*:  
 $\bigwedge A Vs. \llbracket \mathcal{A} e = [A]; \text{fv } e \subseteq \text{set } Vs \rrbracket \implies \mathcal{A} (\text{comp}E_1 Vs e) = \lfloor \text{last-index } Vs \text{ ' } A \rfloor$   
**and**  $\bigwedge A Vs. \llbracket \mathcal{A}s es = [A]; \text{fvs } es \subseteq \text{set } Vs \rrbracket \implies \mathcal{A}s (\text{comp}Es_1 Vs es) = \lfloor \text{last-index } Vs \text{ ' } A \rfloor \langle \text{proof} \rangle$

**lemma** *D-None[iff]*:  $\mathcal{D} (e::'a \text{ exp}) \text{ None}$  **and**  $[\text{iff}]$ :  $\mathcal{D}s (es::'a \text{ exp list}) \text{ None} \langle \text{proof} \rangle$

**lemma** *D-last-index-compE1*:  
 $\bigwedge A Vs. \llbracket A \subseteq \text{set } Vs; \text{fv } e \subseteq \text{set } Vs \rrbracket \implies$   
 $\mathcal{D} e [A] \implies \mathcal{D} (\text{comp}E_1 Vs e) \lfloor \text{last-index } Vs \text{ ' } A \rfloor$   
**and**  $\bigwedge A Vs. \llbracket A \subseteq \text{set } Vs; \text{fvs } es \subseteq \text{set } Vs \rrbracket \implies$   
 $\mathcal{D}s es [A] \implies \mathcal{D}s (\text{comp}Es_1 Vs es) \lfloor \text{last-index } Vs \text{ ' } A \rfloor \langle \text{proof} \rangle$

**lemma** *last-index-image-set*:  $\text{distinct } xs \implies \text{last-index } xs \text{ ' set } xs = \{..<\text{size } xs\} \langle \text{proof} \rangle$

**lemma** *D-compE1*:  
 $\llbracket \mathcal{D} e \lfloor \text{set } Vs \rfloor; \text{fv } e \subseteq \text{set } Vs; \text{distinct } Vs \rrbracket \implies \mathcal{D} (\text{comp}E_1 Vs e) \lfloor \{..<\text{length } Vs\} \rfloor \langle \text{proof} \rangle$

**lemma** *D-compE1'*:  
**assumes**  $\mathcal{D} e \lfloor \text{set}(V\#Vs) \rfloor$  **and**  $\text{fv } e \subseteq \text{set}(V\#Vs)$  **and**  $\text{distinct}(V\#Vs)$   
**shows**  $\mathcal{D} (\text{comp}E_1 (V\#Vs) e) \lfloor \{.. \text{length } Vs \} \rfloor \langle \text{proof} \rangle$   
**lemma** *compP1-pres-wf*:  $\text{wf-J-prog } P \implies \text{wf-J}_1\text{-prog } (\text{comp}P_1 P) \langle \text{proof} \rangle$

end

## 3.6 Compilation Stage 2

**theory** *Compiler2*  
**imports** *PCompiler J1 ../JVM/JVMExec*  
**begin**

**lemma** *bop-expr-length-aux* [simp]:  
 $\text{length } (\text{case bop of } Eq \Rightarrow [CmpEq] \mid Add \Rightarrow [IAdd]) = \text{Suc } 0$   
 $\langle \text{proof} \rangle$

**primrec**  $\text{comp}E_2 :: \text{expr}_1 \Rightarrow \text{instr list}$   
**and**  $\text{comp}Es_2 :: \text{expr}_1 \text{ list} \Rightarrow \text{instr list}$  **where**  
 $\text{comp}E_2 (\text{new } C) = [\text{New } C]$

```

| compE2 (Cast C e) = compE2 e @ [Checkcast C]
| compE2 (Val v) = [Push v]
| compE2 (e1 «bop» e2) = compE2 e1 @ compE2 e2 @
  (case bop of Eq ⇒ [CmpEq]
   | Add ⇒ [IAdd])
| compE2 (Var i) = [Load i]
| compE2 (i:=e) = compE2 e @ [Store i, Push Unit]
| compE2 (e·F{D}) = compE2 e @ [Getfield F D]
| compE2 (C·sF{D}) = [Getstatic C F D]
| compE2 (e1·F{D} := e2) =
  compE2 e1 @ compE2 e2 @ [Putfield F D, Push Unit]
| compE2 (C·sF{D} := e2) =
  compE2 e2 @ [Putstatic C F D, Push Unit]
| compE2 (e·M(es)) = compE2 e @ compEs2 es @ [Invoke M (size es)]
| compE2 (C·sM(es)) = compEs2 es @ [Invokestatic C M (size es)]
| compE2 ({i:T; e}) = compE2 e
| compE2 (e1;;e2) = compE2 e1 @ [Pop] @ compE2 e2
| compE2 (if (e) e1 else e2) =
  (let cnd = compE2 e;
   thn = compE2 e1;
   els = compE2 e2;
   test = IfFalse (int(size thn + 2));
   thnex = Goto (int(size els + 1))
  in cnd @ [test] @ thn @ [thnex] @ els)
| compE2 (while (e) c) =
  (let cnd = compE2 e;
   bdy = compE2 c;
   test = IfFalse (int(size bdy + 3));
   loop = Goto (-int(size bdy + size cnd + 2))
  in cnd @ [test] @ bdy @ [Pop] @ [loop] @ [Push Unit])
| compE2 (throw e) = compE2 e @ [instr.Throw]
| compE2 (try e1 catch(C i) e2) =
  (let catch = compE2 e2
  in compE2 e1 @ [Goto (int(size catch)+2), Store i] @ catch)
| compE2 (INIT C (Cs,b) ← e) = []
| compE2 (RI(C,e);Cs ← e') = []

| compEs2 [] = []
| compEs2 (e#es) = compE2 e @ compEs2 es

```

Compilation of exception table. Is given start address of code to compute absolute addresses necessary in exception table.

```

primrec compxE2 :: expr1 ⇒ pc ⇒ nat ⇒ ex-table
and compxEs2 :: expr1 list ⇒ pc ⇒ nat ⇒ ex-table where
  compxE2 (new C) pc d = []
| compxE2 (Cast C e) pc d = compxE2 e pc d
| compxE2 (Val v) pc d = []
| compxE2 (e1 «bop» e2) pc d =
  compxE2 e1 pc d @ compxE2 e2 (pc + size(compE2 e1)) (d+1)
| compxE2 (Var i) pc d = []
| compxE2 (i:=e) pc d = compxE2 e pc d
| compxE2 (e·F{D}) pc d = compxE2 e pc d
| compxE2 (C·sF{D}) pc d = []
| compxE2 (e1·F{D} := e2) pc d =

```

$\text{compxE}_2 \ e_1 \ pc \ d \ @ \ \text{compxE}_2 \ e_2 \ (pc + \text{size}(\text{compE}_2 \ e_1)) \ (d+1)$   
 $|\ \text{compxE}_2 \ (C \cdot_s F\{D\} := e_2) \ pc \ d = \text{compxE}_2 \ e_2 \ pc \ d$   
 $|\ \text{compxE}_2 \ (e \cdot M(es)) \ pc \ d =$   
 $\quad \text{compxE}_2 \ e \ pc \ d \ @ \ \text{compxEs}_2 \ es \ (pc + \text{size}(\text{compE}_2 \ e)) \ (d+1)$   
 $|\ \text{compxE}_2 \ (C \cdot_s M(es)) \ pc \ d = \text{compxEs}_2 \ es \ pc \ d$   
 $|\ \text{compxE}_2 \ (\{i:T; e\}) \ pc \ d = \text{compxE}_2 \ e \ pc \ d$   
 $|\ \text{compxE}_2 \ (e_1;;e_2) \ pc \ d =$   
 $\quad \text{compxE}_2 \ e_1 \ pc \ d \ @ \ \text{compxE}_2 \ e_2 \ (pc + \text{size}(\text{compE}_2 \ e_1) + 1) \ d$   
 $|\ \text{compxE}_2 \ (\text{if } (e) \ e_1 \ \text{else } e_2) \ pc \ d =$   
 $\quad (\text{let } pc_1 = pc + \text{size}(\text{compE}_2 \ e) + 1;$   
 $\quad \quad pc_2 = pc_1 + \text{size}(\text{compE}_2 \ e_1) + 1$   
 $\quad \text{in } \text{compxE}_2 \ e \ pc \ d \ @ \ \text{compxE}_2 \ e_1 \ pc_1 \ d \ @ \ \text{compxE}_2 \ e_2 \ pc_2 \ d)$   
 $|\ \text{compxE}_2 \ (\text{while } (b) \ e) \ pc \ d =$   
 $\quad \text{compxE}_2 \ b \ pc \ d \ @ \ \text{compxE}_2 \ e \ (pc + \text{size}(\text{compE}_2 \ b) + 1) \ d$   
 $|\ \text{compxE}_2 \ (\text{throw } e) \ pc \ d = \text{compxE}_2 \ e \ pc \ d$   
 $|\ \text{compxE}_2 \ (\text{try } e_1 \ \text{catch}(C \ i) \ e_2) \ pc \ d =$   
 $\quad (\text{let } pc_1 = pc + \text{size}(\text{compE}_2 \ e_1)$   
 $\quad \text{in } \text{compxE}_2 \ e_1 \ pc \ d \ @ \ \text{compxE}_2 \ e_2 \ (pc_1 + 2) \ d \ @ \ [(pc, pc_1, C, pc_1 + 1, d)])$   
 $|\ \text{compxE}_2 \ (\text{INIT } C \ (Cs, b) \leftarrow e) \ pc \ d = []$   
 $|\ \text{compxE}_2 \ (\text{RI}(C, e); Cs \leftarrow e^\wedge) \ pc \ d = []$   
  
 $|\ \text{compxEs}_2 \ [] \ pc \ d = []$   
 $|\ \text{compxEs}_2 \ (e\#es) \ pc \ d = \text{compxE}_2 \ e \ pc \ d \ @ \ \text{compxEs}_2 \ es \ (pc + \text{size}(\text{compE}_2 \ e)) \ (d+1)$

**primrec** *max-stack* :: *expr*<sub>1</sub> ⇒ *nat*

**and** *max-stacks* :: *expr*<sub>1</sub> *list* ⇒ *nat* **where**

$\text{max-stack } (\text{new } C) = 1$   
 $|\ \text{max-stack } (\text{Cast } C \ e) = \text{max-stack } e$   
 $|\ \text{max-stack } (\text{Val } v) = 1$   
 $|\ \text{max-stack } (e_1 \ll\text{bop}\gg e_2) = \max (\text{max-stack } e_1) (\text{max-stack } e_2) + 1$   
 $|\ \text{max-stack } (\text{Var } i) = 1$   
 $|\ \text{max-stack } (i:=e) = \text{max-stack } e$   
 $|\ \text{max-stack } (e \cdot F\{D\}) = \text{max-stack } e$   
 $|\ \text{max-stack } (C \cdot_s F\{D\}) = 1$   
 $|\ \text{max-stack } (e_1 \cdot F\{D\} := e_2) = \max (\text{max-stack } e_1) (\text{max-stack } e_2) + 1$   
 $|\ \text{max-stack } (C \cdot_s F\{D\} := e_2) = \text{max-stack } e_2$   
 $|\ \text{max-stack } (e \cdot M(es)) = \max (\text{max-stack } e) (\text{max-stacks } es) + 1$   
 $|\ \text{max-stack } (C \cdot_s M(es)) = \text{max-stacks } es + 1$   
 $|\ \text{max-stack } (\{i:T; e\}) = \text{max-stack } e$   
 $|\ \text{max-stack } (e_1;;e_2) = \max (\text{max-stack } e_1) (\text{max-stack } e_2)$   
 $|\ \text{max-stack } (\text{if } (e) \ e_1 \ \text{else } e_2) =$   
 $\quad \max (\text{max-stack } e) (\max (\text{max-stack } e_1) (\text{max-stack } e_2))$   
 $|\ \text{max-stack } (\text{while } (e) \ c) = \max (\text{max-stack } e) (\text{max-stack } c)$   
 $|\ \text{max-stack } (\text{throw } e) = \text{max-stack } e$   
 $|\ \text{max-stack } (\text{try } e_1 \ \text{catch}(C \ i) \ e_2) = \max (\text{max-stack } e_1) (\text{max-stack } e_2)$   
  
 $|\ \text{max-stacks } [] = 0$   
 $|\ \text{max-stacks } (e\#es) = \max (\text{max-stack } e) (1 + \text{max-stacks } es)$

**lemma** *max-stack1'*:  $\neg \text{sub-RI } e \implies 1 \leq \text{max-stack } e$ ⟨*proof*⟩

**lemma** *compE<sub>2</sub>-not-Nil'*:  $\neg \text{sub-RI } e \implies \text{compE}_2 \ e \neq []$ ⟨*proof*⟩

**lemma** *compE<sub>2</sub>-nRet*:  $\bigwedge i. i \in \text{set } (\text{compE}_2 \ e_1) \implies i \neq \text{Return}$

**and**  $\bigwedge i. i \in \text{set } (\text{compEs}_2 \ es_1) \implies i \neq \text{Return}$

⟨*proof*⟩

**definition**  $compMb_2 :: staticb \Rightarrow expr_1 \Rightarrow jvm-method$

**where**

$compMb_2 \equiv \lambda b \text{ body.}$   
 $let \ ins = compE_2 \ body \ @ \ [Return];$   
 $xt = compxE_2 \ body \ 0 \ 0$   
 $in \ (max-stack \ body, \ max-vars \ body, \ ins, \ xt)$

**definition**  $compP_2 :: J_1-prog \Rightarrow jvm-prog$

**where**

$compP_2 \equiv compP \ compMb_2$

**lemma**  $compMb_2 \ [simp]:$

$compMb_2 \ b \ e = (max-stack \ e, \ max-vars \ e,$   
 $compE_2 \ e \ @ \ [Return], \ compxE_2 \ e \ 0 \ 0) \langle proof \rangle$

**end**

## 3.7 Correctness of Stage 2

**theory** *Correctness2*

**imports** *HOL-Library.Sublist Compiler2 J1WellForm ../J/EConform*

**begin**

### 3.7.1 Instruction sequences

How to select individual instructions and subsequences of instructions from a program given the class, method and program counter.

**definition**  $before :: jvm-prog \Rightarrow cname \Rightarrow mname \Rightarrow nat \Rightarrow instr \ list \Rightarrow bool$

$\langle \langle -, -, - / \triangleright - \rangle \rangle \ [51, 0, 0, 0, 51] \ 50) \ \mathbf{where}$   
 $P, C, M, pc \triangleright is \longleftrightarrow prefix \ is \ (drop \ pc \ (instrs-of \ P \ C \ M))$

**definition**  $at :: jvm-prog \Rightarrow cname \Rightarrow mname \Rightarrow nat \Rightarrow instr \Rightarrow bool$

$\langle \langle -, -, - / \triangleright - \rangle \rangle \ [51, 0, 0, 0, 51] \ 50) \ \mathbf{where}$   
 $P, C, M, pc \triangleright i \longleftrightarrow (\exists \ is. \ drop \ pc \ (instrs-of \ P \ C \ M) = i \# is)$

**lemma**  $[simp]: P, C, M, pc \triangleright [] \langle proof \rangle$

**lemma**  $[simp]: P, C, M, pc \triangleright (i \# is) = (P, C, M, pc \triangleright i \wedge P, C, M, pc + 1 \triangleright is) \langle proof \rangle$

**lemma**  $[simp]: P, C, M, pc \triangleright (is_1 \ @ \ is_2) = (P, C, M, pc \triangleright is_1 \wedge P, C, M, pc + size \ is_1 \triangleright is_2) \langle proof \rangle$

**lemma**  $[simp]: P, C, M, pc \triangleright i \Longrightarrow instrs-of \ P \ C \ M \ ! \ pc = i \langle proof \rangle$

**lemma**  $beforeM:$

$P \vdash C \ sees \ M, b: Ts \rightarrow T = body \ in \ D \Longrightarrow$   
 $compP_2 \ P, D, M, 0 \triangleright compE_2 \ body \ @ \ [Return] \langle proof \rangle$

This lemma executes a single instruction by rewriting:

**lemma**  $[simp]:$

$P, C, M, pc \triangleright instr \Longrightarrow$   
 $(P \vdash (None, h, (vs, ls, C, M, pc, ics) \# frs, sh) -jvm \rightarrow \sigma) =$   
 $((None, h, (vs, ls, C, M, pc, ics) \# frs, sh) = \sigma' \vee$



$(\exists \sigma. \text{exec}(P, (\text{None}, h, (vs, ls, C, M, pc, ics) \# \text{frs}, sh)) = \text{Some } \sigma \wedge P \vdash \sigma - \text{jvm} \rightarrow \sigma') \langle \text{proof} \rangle$

### 3.7.2 Exception tables

**definition**  $\text{pcs} :: \text{ex-table} \Rightarrow \text{nat set}$

**where**

$\text{pcs } xt \equiv \bigcup (f, t, C, h, d) \in \text{set } xt. \{f ..< t\}$

**lemma**  $\text{pcs-subset}$ :

**shows**  $(\bigwedge pc \ d. \text{pcs}(\text{comp}x E_2 \ e \ pc \ d) \subseteq \{pc..<pc+\text{size}(\text{comp}E_2 \ e)\})$

**and**  $(\bigwedge pc \ d. \text{pcs}(\text{comp}x E_{s_2} \ es \ pc \ d) \subseteq \{pc..<pc+\text{size}(\text{comp}E_{s_2} \ es)\}) \langle \text{proof} \rangle$

**lemma**  $[\text{simp}]$ :  $\text{pcs } [] = \{\}$   $\langle \text{proof} \rangle$

**lemma**  $[\text{simp}]$ :  $\text{pcs } (x \# xt) = \{\text{fst } x ..< \text{fst}(\text{snd } x)\} \cup \text{pcs } xt \langle \text{proof} \rangle$

**lemma**  $[\text{simp}]$ :  $\text{pcs}(xt_1 @ xt_2) = \text{pcs } xt_1 \cup \text{pcs } xt_2 \langle \text{proof} \rangle$

**lemma**  $[\text{simp}]$ :  $pc < pc_0 \vee pc_0 + \text{size}(\text{comp}E_2 \ e) \leq pc \implies pc \notin \text{pcs}(\text{comp}x E_2 \ e \ pc_0 \ d) \langle \text{proof} \rangle$

**lemma**  $[\text{simp}]$ :  $pc < pc_0 \vee pc_0 + \text{size}(\text{comp}E_{s_2} \ es) \leq pc \implies pc \notin \text{pcs}(\text{comp}x E_{s_2} \ es \ pc_0 \ d) \langle \text{proof} \rangle$

**lemma**  $[\text{simp}]$ :  $pc_1 + \text{size}(\text{comp}E_2 \ e_1) \leq pc_2 \implies \text{pcs}(\text{comp}x E_2 \ e_1 \ pc_1 \ d_1) \cap \text{pcs}(\text{comp}x E_2 \ e_2 \ pc_2 \ d_2) = \{\} \langle \text{proof} \rangle$

**lemma**  $[\text{simp}]$ :  $pc_1 + \text{size}(\text{comp}E_2 \ e) \leq pc_2 \implies \text{pcs}(\text{comp}x E_2 \ e \ pc_1 \ d_1) \cap \text{pcs}(\text{comp}x E_{s_2} \ es \ pc_2 \ d_2) = \{\} \langle \text{proof} \rangle$

**lemma**  $[\text{simp}]$ :

$pc \notin \text{pcs } xt_0 \implies \text{match-ex-table } P \ C \ pc \ (xt_0 @ xt_1) = \text{match-ex-table } P \ C \ pc \ xt_1 \langle \text{proof} \rangle$

**lemma**  $[\text{simp}]$ :  $\llbracket x \in \text{set } xt; pc \notin \text{pcs } xt \rrbracket \implies \neg \text{matches-ex-entry } P \ D \ pc \ x \langle \text{proof} \rangle$

**lemma**  $[\text{simp}]$ :

**assumes**  $xe$ :  $xe \in \text{set}(\text{comp}x E_2 \ e \ pc \ d)$  **and** **outside**:  $pc' < pc \vee pc + \text{size}(\text{comp}E_2 \ e) \leq pc'$

**shows**  $\neg \text{matches-ex-entry } P \ C \ pc' \ xe \langle \text{proof} \rangle$

**lemma**  $[\text{simp}]$ :

**assumes**  $xe$ :  $xe \in \text{set}(\text{comp}x E_{s_2} \ es \ pc \ d)$  **and** **outside**:  $pc' < pc \vee pc + \text{size}(\text{comp}E_{s_2} \ es) \leq pc'$

**shows**  $\neg \text{matches-ex-entry } P \ C \ pc' \ xe \langle \text{proof} \rangle$

**lemma**  $\text{match-ex-table-app}[\text{simp}]$ :

$\forall xte \in \text{set } xt_1. \neg \text{matches-ex-entry } P \ D \ pc \ xte \implies$

$\text{match-ex-table } P \ D \ pc \ (xt_1 @ xt) = \text{match-ex-table } P \ D \ pc \ xt \langle \text{proof} \rangle$

**lemma**  $[\text{simp}]$ :

$\forall x \in \text{set } xtab. \neg \text{matches-ex-entry } P \ C \ pc \ x \implies$

$\text{match-ex-table } P \ C \ pc \ xtab = \text{None} \langle \text{proof} \rangle$

**lemma**  $\text{match-ex-entry}$ :

$\text{matches-ex-entry } P \ C \ pc \ (\text{start}, \text{end}, \text{catch-type}, \text{handler}) =$

$(\text{start} \leq pc \wedge pc < \text{end} \wedge P \vdash C \preceq^* \text{catch-type}) \langle \text{proof} \rangle$

**definition**  $\text{caught} :: \text{jvm-prog} \Rightarrow \text{pc} \Rightarrow \text{heap} \Rightarrow \text{addr} \Rightarrow \text{ex-table} \Rightarrow \text{bool}$  **where**

*caught*  $P$   $pc$   $h$   $a$   $xt$   $\longleftrightarrow$   
 $(\exists entry \in set\ xt.\ matches\text{-}ex\text{-}entry\ P\ (cname\text{-}of\ h\ a)\ pc\ entry)$

**definition** *beforex* ::  $jvm\text{-}prog \Rightarrow cname \Rightarrow mname \Rightarrow ex\text{-}table \Rightarrow nat\ set \Rightarrow nat \Rightarrow bool$   
 $(\langle(2,-,-,-) \triangleright / - /' / -,-\rangle [51,0,0,0,0,51] 50)$  **where**  
 $P, C, M \triangleright xt / I, d \longleftrightarrow$   
 $(\exists xt_0\ xt_1.\ ex\text{-}table\text{-}of\ P\ C\ M = xt_0 @ xt @ xt_1 \wedge pcs\ xt_0 \cap I = \{\}$   
 $\wedge pcs\ xt \subseteq I \wedge$   
 $(\forall pc \in I.\ \forall C\ pc'\ d'.\ match\text{-}ex\text{-}table\ P\ C\ pc\ xt_1 = [(pc', d')] \longrightarrow d' \leq d))$

**definition** *dummyx* ::  $jvm\text{-}prog \Rightarrow cname \Rightarrow mname \Rightarrow ex\text{-}table \Rightarrow nat\ set \Rightarrow nat \Rightarrow bool$   $(\langle(2,-,-,-) \triangleright / - /' / -,-\rangle [51,0,0,0,0,51] 50)$  **where**  
 $P, C, M \triangleright xt / I, d \longleftrightarrow P, C, M \triangleright xt / I, d$

### abbreviation

*beforex*<sub>0</sub>  $P\ C\ M\ d\ I\ xt\ xt_0\ xt_1$   
 $\equiv ex\text{-}table\text{-}of\ P\ C\ M = xt_0 @ xt @ xt_1 \wedge pcs\ xt_0 \cap I = \{\}$   
 $\wedge pcs\ xt \subseteq I \wedge (\forall pc \in I.\ \forall C\ pc'\ d'.\ match\text{-}ex\text{-}table\ P\ C\ pc\ xt_1 = [(pc', d')] \longrightarrow d' \leq d)$

**lemma** *beforex-beforex<sub>0</sub>-eq*:

$P, C, M \triangleright xt / I, d \equiv \exists xt_0\ xt_1.\ beforex_0\ P\ C\ M\ d\ I\ xt\ xt_0\ xt_1$   
 $\langle proof \rangle$

**lemma** *beforexD1*:  $P, C, M \triangleright xt / I, d \implies pcs\ xt \subseteq I \langle proof \rangle$

**lemma** *beforex-mono*:  $\llbracket P, C, M \triangleright xt / I, d'; d' \leq d \rrbracket \implies P, C, M \triangleright xt / I, d \langle proof \rangle$

**lemma** [*simp*]:  $P, C, M \triangleright xt / I, d \implies P, C, M \triangleright xt / I, Suc\ d \langle proof \rangle$

**lemma** *beforex-append*[*simp*]:

$pcs\ xt_1 \cap pcs\ xt_2 = \{\} \implies$   
 $P, C, M \triangleright xt_1 @ xt_2 / I, d =$   
 $(P, C, M \triangleright xt_1 / I - pcs\ xt_2, d \wedge P, C, M \triangleright xt_2 / I - pcs\ xt_1, d \wedge P, C, M \triangleright xt_1 @ xt_2 / I, d) \langle proof \rangle$

**lemma** *beforex-appendD1*:

**assumes**  $bx: P, C, M \triangleright xt_1 @ xt_2 @ [(f, t, D, h, d)] / I, d$   
**and**  $pcs: pcs\ xt_1 \subseteq J$  **and**  $JI: J \subseteq I$  **and**  $Jpcs: J \cap pcs\ xt_2 = \{\}$   
**shows**  $P, C, M \triangleright xt_1 / J, d \langle proof \rangle$

**lemma** *beforex-appendD2*:

**assumes**  $bx: P, C, M \triangleright xt_1 @ xt_2 @ [(f, t, D, h, d)] / I, d$   
**and**  $pcs: pcs\ xt_2 \subseteq J$  **and**  $JI: J \subseteq I$  **and**  $Jpcs: J \cap pcs\ xt_1 = \{\}$   
**shows**  $P, C, M \triangleright xt_2 / J, d \langle proof \rangle$

**lemma** *beforexM*:

$P \vdash C\ sees\ M, b: Ts \rightarrow T = body\ in\ D \implies compP_2\ P, D, M \triangleright compxE_2\ body\ 0\ 0 / \{.. < size(compE_2\ body)\}, 0 \langle proof \rangle$

**lemma** *match-ex-table-SomeD2*:

**assumes**  $met: match\text{-}ex\text{-}table\ P\ D\ pc\ (ex\text{-}table\text{-}of\ P\ C\ M) = [(pc', d')]$   
**and**  $bx: P, C, M \triangleright xt / I, d$   
**and**  $nmet: \forall x \in set\ xt.\ \neg matches\text{-}ex\text{-}entry\ P\ D\ pc\ x$  **and**  $pcI: pc \in I$   
**shows**  $d' \leq d \langle proof \rangle$

**lemma** *match-ex-table-SomeD1*:

$$\llbracket \text{match-ex-table } P \ D \ pc \ (\text{ex-table-of } P \ C \ M) = \llbracket (pc', d') \rrbracket;$$

$$P, C, M \triangleright xt / I, d; pc \in I; pc \notin pcs \ xt \rrbracket \implies d' \leq d \langle \text{proof} \rangle$$

### 3.7.3 The correctness proof

#### definition

*handle* :: *jvm-prog*  $\Rightarrow$  *cname*  $\Rightarrow$  *mname*  $\Rightarrow$  *addr*  $\Rightarrow$  *heap*  $\Rightarrow$  *val list*  $\Rightarrow$  *val list*  $\Rightarrow$  *nat*  $\Rightarrow$  *init-call-status*  
 $\Rightarrow$  *frame list*  $\Rightarrow$  *sheap*  
 $\Rightarrow$  *jvm-state* **where**  
*handle* *P C M a h vs ls pc ics frs sh* = *find-handler* *P a h* ((*vs, ls, C, M, pc, ics*) # *frs*) *sh*

**lemma** *aux-isin[simp]*:  $\llbracket B \subseteq A; a \in B \rrbracket \implies a \in A \langle \text{proof} \rangle$

**lemma** *handle-frs-tl-neq*:

*ics-of f*  $\neq$  *No-ics*  
 $\implies (xp, h, f \# frs, sh) \neq \text{handle } P \ C \ M \ xa \ h' \ vs \ l \ pc \ ics \ frs \ sh'$   
 $\langle \text{proof} \rangle$

#### Correctness proof inductive hypothesis

**fun** *calling-to-called* :: *frame*  $\Rightarrow$  *frame* **where**

*calling-to-called* (*stk, loc, D, M, pc, ics*) = (*stk, loc, D, M, pc, case ics of Calling C Cs*  $\Rightarrow$  *Called (C # Cs)*)

**fun** *calling-to-scalled* :: *frame*  $\Rightarrow$  *frame* **where**

*calling-to-scalled* (*stk, loc, D, M, pc, ics*) = (*stk, loc, D, M, pc, case ics of Calling C Cs*  $\Rightarrow$  *Called Cs*)

**fun** *calling-to-calling* :: *frame*  $\Rightarrow$  *cname*  $\Rightarrow$  *frame* **where**

*calling-to-calling* (*stk, loc, D, M, pc, ics*) *C'* = (*stk, loc, D, M, pc, case ics of Calling C Cs*  $\Rightarrow$  *Calling C'*  
(*C # Cs*))

**fun** *calling-to-throwing* :: *frame*  $\Rightarrow$  *addr*  $\Rightarrow$  *frame* **where**

*calling-to-throwing* (*stk, loc, D, M, pc, ics*) *a* = (*stk, loc, D, M, pc, case ics of Calling C Cs*  $\Rightarrow$  *Throwing*  
(*C # Cs*) *a*)

**fun** *calling-to-sthrowing* :: *frame*  $\Rightarrow$  *addr*  $\Rightarrow$  *frame* **where**

*calling-to-sthrowing* (*stk, loc, D, M, pc, ics*) *a* = (*stk, loc, D, M, pc, case ics of Calling C Cs*  $\Rightarrow$  *Throwing Cs*  
*a*)

— pieces of the correctness proof's inductive hypothesis, which depend on the expression being compiled)

**fun** *Jcc-cond* :: *J<sub>1</sub>-prog*  $\Rightarrow$  *ty list*  $\Rightarrow$  *cname*  $\Rightarrow$  *mname*  $\Rightarrow$  *val list*  $\Rightarrow$  *pc*  $\Rightarrow$  *init-call-status*

$\Rightarrow$  *nat set*  $\Rightarrow$  *heap*  $\Rightarrow$  *sheap*  $\Rightarrow$  *expr<sub>1</sub>*  $\Rightarrow$  *bool* **where**  
*Jcc-cond* *P E C M vs pc ics I h sh* (*INIT C<sub>0</sub> (Cs, b)*  $\leftarrow$  *e'*)  
= (( $\exists T. P, E, h, sh \vdash_1 \text{INIT } C_0 (Cs, b) \leftarrow e' : T$ )  $\wedge$  *unit* = *e'*  $\wedge$  *ics* = *No-ics*) |  
*Jcc-cond* *P E C M vs pc ics I h sh* (*RI(C', e<sub>0</sub>); Cs*  $\leftarrow$  *e'*)  
= (((*e<sub>0</sub>* = *C'.<sub>s</sub>clinit*( $\square$ )  $\wedge$  ( $\exists T. P, E, h, sh \vdash_1 \text{RI}(C', e_0); Cs \leftarrow e' : T$ ))  
 $\vee$  (( $\exists a. e_0 = \text{Throw } a$ )  $\wedge$  ( $\forall C \in \text{set}(C' \# Cs). \text{is-class } P \ C$ )))  
 $\wedge$  *unit* = *e'*  $\wedge$  *ics* = *No-ics*) |  
*Jcc-cond* *P E C M vs pc ics I h sh* (*C'.<sub>s</sub>M'(es)*)  
= (*let e* = (*C'.<sub>s</sub>M'(es)*)  
in if *M' = clinit*  $\wedge$  *es* =  $\square$  then ( $\exists T. P, E, h, sh \vdash_1 e : T$ )  $\wedge$  ( $\exists Cs. \text{ics} = \text{Called } Cs$ )  
else (*compP<sub>2</sub> P, C, M, pc*  $\triangleright$  *compE<sub>2</sub> e*  $\wedge$  *compP<sub>2</sub> P, C, M*  $\triangleright$  *compxE<sub>2</sub> e pc (size vs)/I, size vs*)

$$\wedge \{pc..<pc+size(compE_2 e)\} \subseteq I \wedge \neg sub-RI e \wedge ics = No-ics)$$

$$) |$$

*Jcc-cond*  $P E C M vs pc ics I h sh e$   
 $= (compP_2 P, C, M, pc \triangleright compE_2 e \wedge compP_2 P, C, M \triangleright compxE_2 e pc (size vs)/I, size vs$   
 $\wedge \{pc..<pc+size(compE_2 e)\} \subseteq I \wedge \neg sub-RI e \wedge ics = No-ics)$

**fun** *Jcc-frames* :: *jvm-prog*  $\Rightarrow$  *cname*  $\Rightarrow$  *mname*  $\Rightarrow$  *val list*  $\Rightarrow$  *val list*  $\Rightarrow$  *pc*  $\Rightarrow$  *init-call-status*  
 $\Rightarrow$  *frame list*  $\Rightarrow$  *expr<sub>1</sub>*  $\Rightarrow$  *frame list* **where**

*Jcc-frames*  $P C M vs ls pc ics frs (INIT C_0 (C' \# Cs, b) \leftarrow e')$   
 $= (case b of False \Rightarrow (vs, ls, C, M, pc, Calling C' Cs) \# frs$   
 $| True \Rightarrow (vs, ls, C, M, pc, Called (C' \# Cs)) \# frs$

) |  
*Jcc-frames*  $P C M vs ls pc ics frs (INIT C_0 (Nil, b) \leftarrow e')$   
 $= (vs, ls, C, M, pc, Called []) \# frs |$

*Jcc-frames*  $P C M vs ls pc ics frs (RI(C', e_0); Cs \leftarrow e')$   
 $= (case e_0 of Throw a \Rightarrow (vs, ls, C, M, pc, Throwing (C' \# Cs) a) \# frs$   
 $| - \Rightarrow (vs, ls, C, M, pc, Called (C' \# Cs)) \# frs ) |$

*Jcc-frames*  $P C M vs ls pc ics frs (C' \cdot_s M'(es))$   
 $= (if M' = clinit \wedge es = []$   
 $then create-init-frame P C' \# (vs, ls, C, M, pc, ics) \# frs$   
 $else (vs, ls, C, M, pc, ics) \# frs$

) |  
*Jcc-frames*  $P C M vs ls pc ics frs e$   
 $= (vs, ls, C, M, pc, ics) \# frs$

**fun** *Jcc-rhs* :: *J<sub>1</sub>-prog*  $\Rightarrow$  *ty list*  $\Rightarrow$  *cname*  $\Rightarrow$  *mname*  $\Rightarrow$  *val list*  $\Rightarrow$  *val list*  $\Rightarrow$  *pc*  $\Rightarrow$  *init-call-status*  
 $\Rightarrow$  *frame list*  $\Rightarrow$  *heap*  $\Rightarrow$  *val list*  $\Rightarrow$  *sheap*  $\Rightarrow$  *val*  $\Rightarrow$  *expr<sub>1</sub>*  $\Rightarrow$  *jvm-state* **where**

*Jcc-rhs*  $P E C M vs ls pc ics frs h' ls' sh' v (INIT C_0 (Cs, b) \leftarrow e')$   
 $= (None, h', (vs, ls, C, M, pc, Called []) \# frs, sh') |$

*Jcc-rhs*  $P E C M vs ls pc ics frs h' ls' sh' v (RI(C', e_0); Cs \leftarrow e')$   
 $= (None, h', (vs, ls, C, M, pc, Called []) \# frs, sh') |$

*Jcc-rhs*  $P E C M vs ls pc ics frs h' ls' sh' v (C' \cdot_s M'(es))$   
 $= (let e = (C' \cdot_s M'(es))$   
 $in if M' = clinit \wedge es = []$   
 $then (None, h', (vs, ls, C, M, pc, ics) \# frs, sh' (C' \mapsto (fst(the(sh' C')), Done)))$   
 $else (None, h', (v \# vs, ls', C, M, pc + size(compE_2 e), ics) \# frs, sh')$

) |  
*Jcc-rhs*  $P E C M vs ls pc ics frs h' ls' sh' v e$   
 $= (None, h', (v \# vs, ls', C, M, pc + size(compE_2 e), ics) \# frs, sh')$

**fun** *Jcc-err* :: *jvm-prog*  $\Rightarrow$  *cname*  $\Rightarrow$  *mname*  $\Rightarrow$  *heap*  $\Rightarrow$  *val list*  $\Rightarrow$  *val list*  $\Rightarrow$  *pc*  $\Rightarrow$  *init-call-status*  
 $\Rightarrow$  *frame list*  $\Rightarrow$  *sheap*  $\Rightarrow$  *nat set*  $\Rightarrow$  *heap*  $\Rightarrow$  *val list*  $\Rightarrow$  *sheap*  $\Rightarrow$  *addr*  $\Rightarrow$  *expr<sub>1</sub>*  
 $\Rightarrow$  *bool* **where**

*Jcc-err*  $P C M h vs ls pc ics frs sh I h' ls' sh' xa (INIT C_0 (Cs, b) \leftarrow e')$   
 $= (\exists vs'. P \vdash (None, h, Jcc-frames P C M vs ls pc ics frs (INIT C_0 (Cs, b) \leftarrow e'), sh)$   
 $-jvm \rightarrow handle P C M xa h' (vs' @ vs) ls pc ics frs sh') |$

*Jcc-err*  $P C M h vs ls pc ics frs sh I h' ls' sh' xa (RI(C', e_0); Cs \leftarrow e')$   
 $= (\exists vs'. P \vdash (None, h, Jcc-frames P C M vs ls pc ics frs (RI(C', e_0); Cs \leftarrow e'), sh)$   
 $-jvm \rightarrow handle P C M xa h' (vs' @ vs) ls pc ics frs sh') |$

*Jcc-err*  $P C M h vs ls pc ics frs sh I h' ls' sh' xa (C' \cdot_s M'(es))$   
 $= (let e = (C' \cdot_s M'(es))$   
 $in if M' = clinit \wedge es = []$   
 $then case ics of$

Called  $Cs \Rightarrow P \vdash (None, h, Jcc\text{-frames } P \ C \ M \ vs \ ls \ pc \ ics \ frs \ e, sh)$   
 $-jvm \rightarrow (None, h', (vs, ls, C, M, pc, Throwing \ Cs \ xa) \# frs, (sh'(C' \mapsto (fst(the(sh' \ C')), Error))))$   
 else  $(\exists pc_1. pc \leq pc_1 \wedge pc_1 < pc + size(compE_2 \ e) \wedge$   
 $\neg caught \ P \ pc_1 \ h' \ xa \ (compxE_2 \ e \ pc \ (size \ vs)) \wedge$   
 $(\exists vs'. P \vdash (None, h, Jcc\text{-frames } P \ C \ M \ vs \ ls \ pc \ ics \ frs \ e, sh)$   
 $-jvm \rightarrow handle \ P \ C \ M \ xa \ h' \ (vs'@vs) \ ls' \ pc_1 \ ics \ frs \ sh')$   
 $) \mid$   
 $Jcc\text{-err } P \ C \ M \ h \ vs \ ls \ pc \ ics \ frs \ sh \ I \ h' \ ls' \ sh' \ xa \ e$   
 $= (\exists pc_1. pc \leq pc_1 \wedge pc_1 < pc + size(compE_2 \ e) \wedge$   
 $\neg caught \ P \ pc_1 \ h' \ xa \ (compxE_2 \ e \ pc \ (size \ vs)) \wedge$   
 $(\exists vs'. P \vdash (None, h, Jcc\text{-frames } P \ C \ M \ vs \ ls \ pc \ ics \ frs \ e, sh)$   
 $-jvm \rightarrow handle \ P \ C \ M \ xa \ h' \ (vs'@vs) \ ls' \ pc_1 \ ics \ frs \ sh')$

**fun**  $Jcc\text{-pieces} :: J_1\text{-prog} \Rightarrow ty \ list \Rightarrow cname \Rightarrow mname \Rightarrow heap \Rightarrow val \ list \Rightarrow val \ list \Rightarrow pc \Rightarrow$   
 $init\text{-call}\text{-status}$   
 $\Rightarrow frame \ list \Rightarrow sheap \Rightarrow nat \ set \Rightarrow heap \Rightarrow val \ list \Rightarrow sheap \Rightarrow val \Rightarrow addr \Rightarrow expr_1$   
 $\Rightarrow bool \times frame \ list \times jvm\text{-state} \times bool$  **where**  
 $Jcc\text{-pieces } P \ E \ C \ M \ h \ vs \ ls \ pc \ ics \ frs \ sh \ I \ h' \ ls' \ sh' \ v \ xa \ e$   
 $= (Jcc\text{-cond } P \ E \ C \ M \ vs \ pc \ ics \ I \ h \ sh \ e, Jcc\text{-frames } (compP_2 \ P) \ C \ M \ vs \ ls \ pc \ ics \ frs \ e,$   
 $Jcc\text{-rhs } P \ E \ C \ M \ vs \ ls \ pc \ ics \ frs \ h' \ ls' \ sh' \ v \ e,$   
 $Jcc\text{-err } (compP_2 \ P) \ C \ M \ h \ vs \ ls \ pc \ ics \ frs \ sh \ I \ h' \ ls' \ sh' \ xa \ e)$

—  $Jcc\text{-pieces}$  lemmas

**lemma**  $nsub\text{-RI}\text{-}Jcc\text{-pieces}$ :

**assumes**  $[simp]: P \equiv compP_2 \ P_1$

**and**  $nsub: \neg sub\text{-RI} \ e$

**shows**  $Jcc\text{-pieces } P_1 \ E \ C \ M \ h \ vs \ ls \ pc \ ics \ frs \ sh \ I \ h' \ ls' \ sh' \ v \ xa \ e$

$= (let \ cond = P, C, M, pc \triangleright compE_2 \ e \wedge P, C, M \triangleright compxE_2 \ e \ pc \ (size \ vs) / I, size \ vs$   
 $\wedge \{pc.. < pc + size(compE_2 \ e)\} \subseteq I \wedge ics = No\text{-ics};$

$frs' = (vs, ls, C, M, pc, ics) \# frs;$

$rhs = (None, h', (v \# vs, ls', C, M, pc + size(compE_2 \ e), ics) \# frs, sh');$

$err = (\exists pc_1. pc \leq pc_1 \wedge pc_1 < pc + size(compE_2 \ e) \wedge$

$\neg caught \ P \ pc_1 \ h' \ xa \ (compxE_2 \ e \ pc \ (size \ vs)) \wedge$

$(\exists vs'. P \vdash (None, h, frs', sh) -jvm \rightarrow handle \ P \ C \ M \ xa \ h' \ (vs'@vs) \ ls' \ pc_1 \ ics \ frs \ sh'))$

$in \ (cond, frs', rhs, err)$

)

$\langle proof \rangle$

**lemma**  $Jcc\text{-pieces}\text{-Cast}$ :

**assumes**  $[simp]: P \equiv compP_2 \ P_1$

**and**  $Jcc\text{-pieces } P_1 \ E \ C \ M \ h_0 \ vs \ ls_0 \ pc \ ics \ frs \ sh_0 \ I \ h_1 \ ls_1 \ sh_1 \ v \ xa \ (Cast \ C' \ e)$

$= (True, frs_0, (xp', h', (v \# vs', ls', C_0, M', pc', ics') \# frs', sh'), err)$

**shows**  $Jcc\text{-pieces } P_1 \ E \ C \ M \ h_0 \ vs \ ls_0 \ pc \ ics \ frs \ sh_0 \ I \ h_1 \ ls_1 \ sh_1 \ v' \ xa \ e$

$= (True, frs_0, (xp', h', (v' \# vs', ls', C_0, M', pc' - 1, ics') \# frs', sh'),$

$(\exists pc_1. pc \leq pc_1 \wedge pc_1 < pc + size(compE_2 \ e) \wedge$

$\neg caught \ P \ pc_1 \ h_1 \ xa \ (compxE_2 \ e \ pc \ (size \ vs)) \wedge$

$(\exists vs'. P \vdash (None, h_0, frs_0, sh_0) -jvm \rightarrow handle \ P \ C \ M \ xa \ h_1 \ (vs'@vs) \ ls_1 \ pc_1 \ ics \ frs \ sh_1)))$

$\langle proof \rangle$

**lemma**  $Jcc\text{-pieces}\text{-BinOp}1$ :

**assumes**

$Jcc\text{-pieces } P \ E \ C \ M \ h_0 \ vs \ ls_0 \ pc \ ics \ frs \ sh_0 \ I \ h_2 \ ls_2 \ sh_2 \ v \ xa \ (e \ll bop \ e')$

$= (True, frs_0, (xp', h', (v \# vs', ls', C_0, M', pc', ics') \# frs', sh'), err)$   
**shows**  $\exists err. Jcc\text{-pieces } P E C M h_0 vs ls_0 pc ics frs sh_0$   
 $(I - pcs (compxE_2 e' (pc + length (compE_2 e))) (Suc (length vs')))) h_1 ls_1 sh_1 v' xa e$   
 $= (True, frs_0, (xp', h_1, (v \# vs', ls_1, C_0, M', pc' - size (compE_2 e') - 1, ics') \# frs', sh_1), err)$   
 $\langle proof \rangle$

**lemma** *Jcc-pieces-BinOp2*:

**assumes**  $[simp]: P \equiv compP_2 P_1$

**and** *Jcc-pieces*  $P_1 E C M h_0 vs ls_0 pc ics frs sh_0 I h_2 ls_2 sh_2 v xa (e \ll bop \gg e')$   
 $= (True, frs_0, (xp', h', (v \# vs', ls', C_0, M', pc', ics') \# frs', sh'), err)$

**shows**  $\exists err. Jcc\text{-pieces } P_1 E C M h_1 (v_1 \# vs) ls_1 (pc + size (compE_2 e)) ics frs sh_1$

$(I - pcs (compxE_2 e pc (length vs'))) h_2 ls_2 sh_2 v' xa e'$

$= (True, (v_1 \# vs, ls_1, C, M, pc + size (compE_2 e), ics) \# frs,$

$(xp', h', (v \# v_1 \# vs', ls', C_0, M', pc' - 1, ics') \# frs', sh'),$

$(\exists pc_1. pc + size (compE_2 e) \leq pc_1 \wedge pc_1 < pc + size (compE_2 e) + length (compE_2 e') \wedge$

$\neg caught P pc_1 h_2 xa (compxE_2 e' (pc + size (compE_2 e)) (Suc (length vs))) \wedge$

$(\exists vs'. P \vdash (None, h_1, (v_1 \# vs, ls_1, C, M, pc + size (compE_2 e), ics) \# frs, sh_1)$

$-jvm \rightarrow handle P C M xa h_2 (vs' @ v_1 \# vs) ls_2 pc_1 ics frs sh_2))$

$\langle proof \rangle$

**lemma** *Jcc-pieces-FAcc*:

**assumes**

*Jcc-pieces*  $P E C M h_0 vs ls_0 pc ics frs sh_0 I h_1 ls_1 sh_1 v xa (e \cdot F\{D\})$

$= (True, frs_0, (xp', h', (v \# vs', ls', C_0, M', pc', ics') \# frs', sh'), err)$

**shows**  $\exists err. Jcc\text{-pieces } P E C M h_0 vs ls_0 pc ics frs sh_0 I h_1 ls_1 sh_1 v' xa e$

$= (True, frs_0, (xp', h', (v \# vs', ls', C_0, M', pc' - 1, ics') \# frs', sh'), err)$

$\langle proof \rangle$

**lemma** *Jcc-pieces-LAss*:

**assumes**  $[simp]: P \equiv compP_2 P_1$

**and** *Jcc-pieces*  $P_1 E C M h_0 vs ls_0 pc ics frs sh_0 I h_1 ls_1 sh_1 v xa (i := e)$

$= (True, frs_0, (xp', h', (v \# vs', ls', C_0, M', pc', ics') \# frs', sh'), err)$

**shows** *Jcc-pieces*  $P_1 E C M h_0 vs ls_0 pc ics frs sh_0 I h_1 ls_1 sh_1 v' xa e$

$= (True, frs_0, (xp', h', (v \# vs', ls', C_0, M', pc' - 2, ics') \# frs', sh'),$

$(\exists pc_1. pc \leq pc_1 \wedge pc_1 < pc + size (compE_2 e) \wedge$

$\neg caught P pc_1 h_1 xa (compxE_2 e pc (size vs)) \wedge$

$(\exists vs'. P \vdash (None, h_0, frs_0, sh_0) -jvm \rightarrow handle P C M xa h_1 (vs' @ vs) ls_1 pc_1 ics frs sh_1))$

$\langle proof \rangle$

**lemma** *Jcc-pieces-FAss1*:

**assumes**

*Jcc-pieces*  $P E C M h_0 vs ls_0 pc ics frs sh_0 I h_2 ls_2 sh_2 v xa (e \cdot F\{D\} := e')$

$= (True, frs_0, (xp', h', (v \# vs', ls', C_0, M', pc', ics') \# frs', sh'), err)$

**shows**  $\exists err. Jcc\text{-pieces } P E C M h_0 vs ls_0 pc ics frs sh_0$

$(I - pcs (compxE_2 e' (pc + length (compE_2 e)) (Suc (length vs')))) h_1 ls_1 sh_1 v' xa e$

$= (True, frs_0, (xp', h_1, (v \# vs', ls_1, C_0, M', pc' - size (compE_2 e') - 2, ics') \# frs', sh_1), err)$

$\langle proof \rangle$

**lemma** *Jcc-pieces-FAss2*:

**assumes**

*Jcc-pieces*  $P E C M h_0 vs ls_0 pc ics frs sh_0 I h_2 ls_2 sh_2 v xa (e \cdot F\{D\} := e')$

$= (True, frs_0, (xp', h', (v \# vs', ls', C_0, M', pc', ics') \# frs', sh'), err)$

**shows** *Jcc-pieces*  $P E C M h_1 (v_1 \# vs) ls_1 (pc + size (compE_2 e)) ics frs sh_1$

$(I - pcs (compxE_2 e pc (length vs'))) h_2 ls_2 sh_2 v' xa e'$

$$\begin{aligned}
&= (\text{True}, (v_1\#vs, ls_1, C, M, pc + \text{size}(\text{comp}E_2 e), ics)\#frs, \\
&\quad (xp', h', (v\#v_1\#vs', ls', C_0, M', pc' - 2, ics')\#frs', sh'), \\
&\quad (\exists pc_1. (pc + \text{size}(\text{comp}E_2 e)) \leq pc_1 \wedge pc_1 < pc + \text{size}(\text{comp}E_2 e) + \text{size}(\text{comp}E_2 e') \wedge \\
&\quad \quad \neg \text{caught}(\text{comp}P_2 P) pc_1 h_2 xa (\text{comp}xE_2 e' (pc + \text{size}(\text{comp}E_2 e)) (\text{size}(v_1\#vs))) \wedge \\
&\quad (\exists vs'. (\text{comp}P_2 P) \vdash (\text{None}, h_1, (v_1\#vs, ls_1, C, M, pc + \text{size}(\text{comp}E_2 e), ics)\#frs, sh_1) \\
&\quad \quad -jvm \rightarrow \text{handle}(\text{comp}P_2 P) C M xa h_2 (vs'@v_1\#vs) ls_2 pc_1 ics frs sh_2)))
\end{aligned}$$

$\langle \text{proof} \rangle$

**lemma** *Jcc-pieces-SFAss*:

**assumes**

$$\begin{aligned}
&Jcc\text{-pieces } P E C M h_0 vs ls_0 pc ics frs sh_0 I h' ls' sh' v xa (C' \cdot_s F\{D\} := e) \\
&= (\text{True}, frs_0, (xp', h', (v\#vs', ls', C_0, M', pc', ics')\#frs', sh'), err)
\end{aligned}$$

**shows**  $\exists err. Jcc\text{-pieces } P E C M h_0 vs ls_0 pc ics frs sh_0 I h_1 ls_1 sh_1 v' xa e$

$$= (\text{True}, frs_0, (xp', h_1, (v\#vs', ls_1, C_0, M', pc' - 2, ics')\#frs', sh_1), err)$$

$\langle \text{proof} \rangle$

**lemma** *Jcc-pieces-Call1*:

**assumes**

$$\begin{aligned}
&Jcc\text{-pieces } P E C M h_0 vs ls_0 pc ics frs sh_0 I h_3 ls_3 sh_3 v xa (e \cdot M_0(es)) \\
&= (\text{True}, frs_0, (xp', h', (v\#vs', ls', C', M', pc', ics')\#frs', sh'), err)
\end{aligned}$$

**shows**  $\exists err. Jcc\text{-pieces } P E C M h_0 vs ls_0 pc ics frs sh_0$

$$\begin{aligned}
&(I - pcs(\text{comp}xEs_2 es (pc + \text{length}(\text{comp}E_2 e)) (\text{Suc}(\text{length } vs'))) h_1 ls_1 sh_1 v' xa e \\
&= (\text{True}, frs_0,
\end{aligned}$$

$$(xp', h_1, (v\#vs', ls_1, C', M', pc' - \text{size}(\text{comp}Es_2 es) - 1, ics')\#frs', sh_1), err)$$

$\langle \text{proof} \rangle$

**lemma** *Jcc-pieces-clinit*:

**assumes**  $[simp]: P \equiv \text{comp}P_2 P_1$

**and cond:** *Jcc-cond*  $P_1 E C M vs pc ics I h sh (C1 \cdot_s \text{clinit}(\square))$

**shows** *Jcc-pieces*  $P_1 E C M h vs ls pc ics frs sh I h' ls' sh' v xa (C1 \cdot_s \text{clinit}(\square))$

$$\begin{aligned}
&= (\text{True}, \text{create-init-frame } P C1 \# (vs, ls, C, M, pc, ics)\#frs, \\
&\quad (\text{None}, h', (vs, ls, C, M, pc, ics)\#frs, sh'(C1 \mapsto (\text{fst}(\text{the}(sh' C1)), \text{Done}))), \\
&\quad P \vdash (\text{None}, h, \text{create-init-frame } P C1 \# (vs, ls, C, M, pc, ics)\#frs, sh) -jvm \rightarrow \\
&\quad (\text{case } ics \text{ of Called } Cs \Rightarrow (\text{None}, h', (vs, ls, C, M, pc, \text{Throwing } Cs xa)\#frs, (sh'(C1 \mapsto (\text{fst}(\text{the}(sh' \\
&\quad C1)), \text{Error}))))))
\end{aligned}$$

$\langle \text{proof} \rangle$

**lemma** *Jcc-pieces-SCall-clinit-body*:

**assumes**  $[simp]: P \equiv \text{comp}P_2 P_1$  **and** *wf*: *wf-J<sub>1</sub>-prog*  $P_1$

**and** *Jcc-pieces*  $P_1 E C M h_0 vs ls_0 pc ics frs sh_0 I h_3 ls_2 sh_3 v xa (C1 \cdot_s \text{clinit}(\square))$

$$= (\text{True}, frs', rhs', err')$$

**and method:**  $P_1 \vdash C1 \text{ sees } \text{clinit}, \text{Static}: \square \rightarrow \text{Void} = \text{body in } D$

**shows** *Jcc-pieces*  $P_1 \square D \text{clinit } h_2 \square (\text{replicate}(\text{max-vars } \text{body}) \text{undefined}) 0$

*No-ics*  $(tl \text{ frs}') sh_2 \{.. < \text{length}(\text{comp}E_2 \text{body})\} h_3 ls_3 sh_3 v xa \text{body}$

$$\begin{aligned}
&= (\text{True}, frs', \\
&\quad (\text{None}, h_3, ([v], ls_3, D, \text{clinit}, \text{size}(\text{comp}E_2 \text{body}), \text{No-ics})\#tl \text{ frs}', sh_3), \\
&\quad \exists pc_1. 0 \leq pc_1 \wedge pc_1 < \text{size}(\text{comp}E_2 \text{body}) \wedge \\
&\quad \quad \neg \text{caught } P pc_1 h_3 xa (\text{comp}xE_2 \text{body } 0 0) \wedge \\
&\quad (\exists vs'. P \vdash (\text{None}, h_2, frs', sh_2) -jvm \rightarrow \text{handle } P D \text{clinit } xa h_3 vs' ls_3 pc_1 \\
&\quad \quad \text{No-ics } (tl \text{ frs}') sh_3))
\end{aligned}$$

$\langle \text{proof} \rangle$

**lemma** *Jcc-pieces-Cons*:

**assumes**  $[simp]: P \equiv \text{comp}P_2 P_1$

**and**  $P, C, M, pc \triangleright \text{compEs}_2 (e \# es)$  **and**  $P, C, M \triangleright \text{compxEs}_2 (e \# es) pc (size\ vs)/I, size\ vs$   
**and**  $\{pc.. < pc + size(\text{compEs}_2 (e \# es))\} \subseteq I$   
**and**  $ics = No\text{-ics}$   
**and**  $\neg \text{sub-RIs} (e \# es)$   
**shows**  $Jcc\text{-pieces } P_1\ E\ C\ M\ h\ vs\ ls\ pc\ ics\ frs\ sh$   
 $(I - pcs(\text{compxEs}_2\ es\ (pc + length(\text{compE}_2\ e))\ (Suc\ (length\ vs))))\ h'\ ls'\ sh'\ v\ xa\ e$   
 $= (True, (vs, ls, C, M, pc, ics) \# frs,$   
 $(None, h', (v \# vs, ls', C, M, pc + length(\text{compE}_2\ e), ics) \# frs, sh'),$   
 $\exists pc_1 \geq pc. pc_1 < pc + length(\text{compE}_2\ e) \wedge \neg caught\ P\ pc_1\ h'\ xa\ (\text{compxE}_2\ e\ pc\ (length\ vs))$   
 $\wedge (\exists vs'. P \vdash (None, h, (vs, ls, C, M, pc, ics) \# frs, sh)$   
 $\quad -jvm \rightarrow handle\ P\ C\ M\ xa\ h'\ (vs'@vs)\ ls'\ pc_1\ ics\ frs\ sh')$   
 $\langle proof \rangle$

**lemma**  $Jcc\text{-pieces-InitNone}$ :

**assumes**  $[simp]: P \equiv \text{compP}_2\ P_1$   
**and**  $Jcc\text{-pieces } P_1\ E\ C\ M\ h\ vs\ l\ pc\ ics\ frs\ sh\ I\ h'\ l'\ sh'\ v\ xa\ (INIT\ C'\ (C_0 \# Cs, False) \leftarrow e)$   
 $= (True, frs', (None, h', (vs, l, C, M, pc, Called\ [])) \# frs, sh'), err)$   
**shows**  
 $Jcc\text{-pieces } P_1\ E\ C\ M\ h\ vs\ l\ pc\ ics\ frs\ (sh(C_0 \mapsto (sblank\ P\ C_0, Prepared)))$   
 $I\ h'\ l'\ sh'\ v\ xa\ (INIT\ C'\ (C_0 \# Cs, False) \leftarrow e)$   
 $= (True, frs', (None, h', (vs, l, C, M, pc, Called\ [])) \# frs, sh'),$   
 $\exists vs'. P \vdash (None, h, frs', (sh(C_0 \mapsto (sblank\ P_1\ C_0, Prepared))))$   
 $\quad -jvm \rightarrow handle\ P\ C\ M\ xa\ h'\ (vs'@vs)\ l\ pc\ ics\ frs\ sh')$   
 $\langle proof \rangle$

**lemma**  $Jcc\text{-pieces-InitDP}$ :

**assumes**  $[simp]: P \equiv \text{compP}_2\ P_1$   
**and**  $Jcc\text{-pieces } P_1\ E\ C\ M\ h\ vs\ l\ pc\ ics\ frs\ sh\ I\ h'\ l'\ sh'\ v\ xa\ (INIT\ C'\ (C_0 \# Cs, False) \leftarrow e)$   
 $= (True, frs', (None, h', (vs, l, C, M, pc, Called\ [])) \# frs, sh'), err)$   
**shows**  
 $Jcc\text{-pieces } P_1\ E\ C\ M\ h\ vs\ l\ pc\ ics\ frs\ sh\ I\ h'\ l'\ sh'\ v\ xa\ (INIT\ C'\ (Cs, True) \leftarrow e)$   
 $= (True, (calling\text{-to}\text{-scalled}\ (hd\ frs')) \# (tl\ frs'),$   
 $(None, h', (vs, l, C, M, pc, Called\ [])) \# frs, sh'),$   
 $\exists vs'. P \vdash (None, h, calling\text{-to}\text{-scalled}\ (hd\ frs') \# (tl\ frs'), sh)$   
 $\quad -jvm \rightarrow handle\ P\ C\ M\ xa\ h'\ (vs'@vs)\ l\ pc\ ics\ frs\ sh')$   
 $\langle proof \rangle$

**lemma**  $Jcc\text{-pieces-InitError}$ :

**assumes**  $[simp]: P \equiv \text{compP}_2\ P_1$   
**and**  $Jcc\text{-pieces } P_1\ E\ C\ M\ h\ vs\ l\ pc\ ics\ frs\ sh\ I\ h'\ l'\ sh'\ v\ xa\ (INIT\ C'\ (C_0 \# Cs, False) \leftarrow e)$   
 $= (True, frs', (None, h', (vs, l, C, M, pc, Called\ [])) \# frs, sh'), err)$   
**and**  $err: sh\ C_0 = Some(sfs, Error)$   
**shows**  
 $Jcc\text{-pieces } P_1\ E\ C\ M\ h\ vs\ l\ pc\ ics\ frs\ sh\ I\ h'\ l'\ sh'\ v\ xa\ (RI\ (C_0, THROW\ NoClassDefFoundError); Cs$   
 $\leftarrow e)$   
 $= (True, (calling\text{-to}\text{-throwing}\ (hd\ frs')\ (addr\text{-of}\text{-sys}\text{-xcpt}\ NoClassDefFoundError)) \# (tl\ frs'),$   
 $(None, h', (vs, l, C, M, pc, Called\ [])) \# frs, sh'),$   
 $\exists vs'. P \vdash (None, h, (calling\text{-to}\text{-throwing}\ (hd\ frs')\ (addr\text{-of}\text{-sys}\text{-xcpt}\ NoClassDefFoundEr$   
 $ror)) \# (tl\ frs'), sh)$   
 $\quad -jvm \rightarrow handle\ P\ C\ M\ xa\ h'\ (vs'@vs)\ l\ pc\ ics\ frs\ sh')$   
 $\langle proof \rangle$

**lemma**  $Jcc\text{-pieces-InitObj}$ :

**assumes**  $[simp]: P \equiv \text{compP}_2\ P_1$



**and** *Jcc-pieces*  $P_1 E C M h vs l pc ics frs sh I h' l' (sh(C_0 \mapsto (sfs, Processing))) v xa (INIT C' (C_0 \# Cs, False) \leftarrow e)$

$= (True, frs', (None, h', (vs, l, C, M, pc, Called [])) \# frs, sh'), err)$

**shows**

*Jcc-pieces*  $P_1 E C M h vs l pc ics frs (sh(C_0 \mapsto (sfs, Processing))) I h' l' sh'' v xa (INIT C' (C_0 \# Cs, True) \leftarrow e)$

$= (True, calling-to-called (hd frs') \# (tl frs'),$   
 $(None, h', (vs, l, C, M, pc, Called [])) \# frs, sh''),$   
 $\exists vs'. P \vdash (None, h, calling-to-called (hd frs') \# (tl frs'), sh')$   
 $-jvm \rightarrow handle P C M xa h' (vs' @ vs) l pc ics frs sh'')$

$\langle proof \rangle$

**lemma** *Jcc-pieces-InitNonObj*:

**assumes**  $[simp]: P \equiv comp P_2 P_1$

**and** *is-class*  $P_1 D$  **and**  $D \notin set (C_0 \# Cs)$  **and**  $\forall C \in set (C_0 \# Cs). P_1 \vdash C \preceq^* D$

**and** *pcs*: *Jcc-pieces*  $P_1 E C M h vs l pc ics frs sh I h' l' (sh(C_0 \mapsto (sfs, Processing))) v xa (INIT C' (C_0 \# Cs, False) \leftarrow e)$

$= (True, frs', (None, h', (vs, l, C, M, pc, Called [])) \# frs, sh'), err)$

**shows**

*Jcc-pieces*  $P_1 E C M h vs l pc ics frs (sh(C_0 \mapsto (sfs, Processing))) I h' l' sh'' v xa (INIT C' (D \# C_0 \# Cs, False) \leftarrow e)$

$= (True, calling-to-calling (hd frs') D \# (tl frs'),$   
 $(None, h', (vs, l, C, M, pc, Called [])) \# frs, sh''),$   
 $\exists vs'. P \vdash (None, h, calling-to-calling (hd frs') D \# (tl frs'), sh')$   
 $-jvm \rightarrow handle P C M xa h' (vs' @ vs) l pc ics frs sh'')$

$\langle proof \rangle$

**lemma** *Jcc-pieces-InitRInit*:

**assumes**  $[simp]: P \equiv comp P_2 P_1$  **and** *wf*: *wf-J<sub>1</sub>-prog*  $P_1$

**and** *Jcc-pieces*  $P_1 E C M h vs l pc ics frs sh I h' l' sh' v xa (INIT C' (C_0 \# Cs, True) \leftarrow e)$

$= (True, frs', (None, h', (vs, l, C, M, pc, Called [])) \# frs, sh'), err)$

**shows**

*Jcc-pieces*  $P_1 E C M h vs l pc ics frs sh I h' l' sh' v xa (RI (C_0, C_0 \cdot_s clinit([])); Cs \leftarrow e)$

$= (True, frs',$   
 $(None, h', (vs, l, C, M, pc, Called [])) \# frs, sh'),$   
 $\exists vs'. P \vdash (None, h, frs', sh)$   
 $-jvm \rightarrow handle P C M xa h' (vs' @ vs) l pc ics frs sh')$

$\langle proof \rangle$

**lemma** *Jcc-pieces-RInit-clinit*:

**assumes**  $[simp]: P \equiv comp P_2 P_1$  **and** *wf*: *wf-J<sub>1</sub>-prog*  $P_1$

**and** *Jcc-pieces*  $P_1 E C M h vs l pc ics frs sh I h_1 l_1 sh_1 v xa (RI (C_0, C_0 \cdot_s clinit([])); Cs \leftarrow e)$

$= (True, frs',$   
 $(None, h_1, (vs, l, C, M, pc, Called [])) \# frs, sh_1), err)$

**shows**

*Jcc-pieces*  $P_1 E C M h vs l pc (Called Cs) (tl frs') sh I h' l' sh' v xa (C_0 \cdot_s clinit([]))$

$= (True, create-init-frame P C_0 \# (vs, l, C, M, pc, Called Cs) \# tl frs',$   
 $(None, h', (vs, l, C, M, pc, Called Cs) \# tl frs', sh'(C_0 \mapsto (fst(the(sh' C_0)), Done))),$   
 $P \vdash (None, h, create-init-frame P C_0 \# (vs, l, C, M, pc, Called Cs) \# tl frs', sh)$   
 $-jvm \rightarrow (None, h', (vs, l, C, M, pc, Throwing Cs xa) \# tl frs', sh'(C_0 \mapsto (fst(the(sh' C_0)), Error))))$

$\langle proof \rangle$

**lemma** *Jcc-pieces-RInit-Init*:

**assumes**  $[simp]: P \equiv comp P_2 P_1$  **and** *wf*: *wf-J<sub>1</sub>-prog*  $P_1$

**and proc:**  $\forall C' \in \text{set } Cs. \exists \text{sfs. } sh'' C' = \lfloor (\text{sfs}, \text{Processing}) \rfloor$   
**and Jcc-pieces**  $P_1 E C M h \text{ vs } l \text{ pc ics frs sh } I h_1 l_1 sh_1 v xa (RI (C_0, C_0 \cdot s \text{clinit}(\square)); Cs \leftarrow e)$   
 $= (True, frs',$   
 $(None, h_1, (vs, l, C, M, pc, Called \square) \# frs, sh_1), err)$

**shows**

$Jcc\text{-pieces } P_1 E C M h' \text{ vs } l \text{ pc ics frs sh'' } I h_1 l_1 sh_1 v xa (INIT (last (C_0 \# Cs)) (Cs, True) \leftarrow e)$   
 $= (True, (vs, l, C, M, pc, Called Cs) \# frs,$   
 $(None, h_1, (vs, l, C, M, pc, Called \square) \# frs, sh_1),$   
 $\exists vs'. P \vdash (None, h', (vs, l, C, M, pc, Called Cs) \# frs, sh'')$   
 $\quad -jvm \rightarrow \text{handle } P C M xa h_1 (vs' @ vs) l \text{ pc ics frs sh}_1)$

$\langle \text{proof} \rangle$

**lemma Jcc-pieces-RInit-RInit:**

**assumes**  $[simp]: P \equiv \text{comp} P_2 P_1$

**and Jcc-pieces**  $P_1 E C M h \text{ vs } l \text{ pc ics frs sh } I h_1 l_1 sh_1 v xa (RI (C_0, e); D \# Cs \leftarrow e')$   
 $= (True, frs', rhs, err)$

**and hd:**  $hd \text{ frs}' = (vs1, l1, C1, M1, pc1, ics1)$

**shows**

$Jcc\text{-pieces } P_1 E C M h' \text{ vs } l \text{ pc ics frs sh'' } I h_1 l_1 sh_1 v xa (RI (D, Throw xa) ; Cs \leftarrow e')$   
 $= (True, (vs1, l1, C1, M1, pc1, Throwing (D \# Cs) xa) \# tl \text{ frs}',$   
 $(None, h_1, (vs, l, C, M, pc, Called \square) \# frs, sh_1),$   
 $\exists vs'. P \vdash (None, h', (vs1, l1, C1, M1, pc1, Throwing (D \# Cs) xa) \# tl \text{ frs}', sh'')$   
 $\quad -jvm \rightarrow \text{handle } P C M xa h_1 (vs' @ vs) l \text{ pc ics frs sh}_1)$

$\langle \text{proof} \rangle$

## JVM stepping lemmas

**lemma jvm-Invoke:**

**assumes**  $[simp]: P \equiv \text{comp} P_2 P_1$

**and**  $P, C, M, pc \triangleright \text{Invoke } M' (\text{length } Ts)$

**and ha:**  $h_2 a = \lfloor (Ca, fs) \rfloor$  **and method:**  $P_1 \vdash Ca \text{ sees } M', \text{NonStatic} : Ts \rightarrow T = \text{body in } D$

**and len:**  $\text{length } pvs = \text{length } Ts$  **and**  $ls_2' = \text{Addr } a \# pvs @ \text{replicate } (\text{max-vars } \text{body}) \text{ undefined}$

**shows**  $P \vdash (None, h_2, (\text{rev } pvs @ \text{Addr } a \# vs, ls_2, C, M, pc, \text{No-ics}) \# frs, sh_2) -jvm \rightarrow$

$(None, h_2, (\square, ls_2', D, M', 0, \text{No-ics}) \# (\text{rev } pvs @ \text{Addr } a \# vs, ls_2, C, M, pc, \text{No-ics}) \# frs, sh_2)$

$\langle \text{proof} \rangle$

**lemma jvm-Invokestatic:**

**assumes**  $[simp]: P \equiv \text{comp} P_2 P_1$

**and**  $P, C, M, pc \triangleright \text{Invokestatic } C' M' (\text{length } Ts)$

**and sh:**  $sh_2 D = \text{Some}(sfs, Done)$

**and method:**  $P_1 \vdash C' \text{ sees } M', \text{Static} : Ts \rightarrow T = \text{body in } D$

**and len:**  $\text{length } pvs = \text{length } Ts$  **and**  $ls_2' = pvs @ \text{replicate } (\text{max-vars } \text{body}) \text{ undefined}$

**shows**  $P \vdash (None, h_2, (\text{rev } pvs @ vs, ls_2, C, M, pc, \text{No-ics}) \# frs, sh_2) -jvm \rightarrow$

$(None, h_2, (\square, ls_2', D, M', 0, \text{No-ics}) \# (\text{rev } pvs @ vs, ls_2, C, M, pc, \text{No-ics}) \# frs, sh_2)$

$\langle \text{proof} \rangle$

**lemma jvm-Invokestatic-Called:**

**assumes**  $[simp]: P \equiv \text{comp} P_2 P_1$

**and**  $P, C, M, pc \triangleright \text{Invokestatic } C' M' (\text{length } Ts)$

**and sh:**  $sh_2 D = \text{Some}(sfs, i)$

**and method:**  $P_1 \vdash C' \text{ sees } M', \text{Static} : Ts \rightarrow T = \text{body in } D$

**and len:**  $\text{length } pvs = \text{length } Ts$  **and**  $ls_2' = pvs @ \text{replicate } (\text{max-vars } \text{body}) \text{ undefined}$

**shows**  $P \vdash (None, h_2, (\text{rev } pvs @ vs, ls_2, C, M, pc, Called \square) \# frs, sh_2) -jvm \rightarrow$

$(None, h_2, (\square, ls_2', D, M', 0, \text{No-ics}) \# (\text{rev } pvs @ vs, ls_2, C, M, pc, \text{No-ics}) \# frs, sh_2)$

$\langle \text{proof} \rangle$

**lemma** *jvm-Return-Init*:

$P, D, \text{clinit}, 0 \triangleright \text{comp}E_2 \text{ body} @ [\text{Return}]$   
 $\implies P \vdash (\text{None}, h, (vs, ls, D, \text{clinit}, \text{size}(\text{comp}E_2 \text{ body}), \text{No-ics}) \# \text{frs}, sh)$   
 $\quad -\text{jvm} \rightarrow (\text{None}, h, \text{frs}, sh(D \mapsto (\text{fst}(\text{the}(sh D)), \text{Done})))$   
**(is**  $?P \implies P \vdash ?s1 -\text{jvm} \rightarrow ?s2$ )

$\langle \text{proof} \rangle$

**lemma** *jvm-InitNone*:

$\llbracket \text{ics-of } f = \text{Calling } C \text{ Cs};$   
 $\quad sh C = \text{None} \rrbracket$   
 $\implies P \vdash (\text{None}, h, f \# \text{frs}, sh) -\text{jvm} \rightarrow (\text{None}, h, f \# \text{frs}, sh(C \mapsto (\text{sblank } P C, \text{Prepared})))$   
**(is**  $\llbracket ?P; ?Q \rrbracket \implies P \vdash ?s1 -\text{jvm} \rightarrow ?s2$ )

$\langle \text{proof} \rangle$

**lemma** *jvm-InitDP*:

$\llbracket \text{ics-of } f = \text{Calling } C \text{ Cs};$   
 $\quad sh C = \llbracket (sfs, i) \rrbracket; i = \text{Done} \vee i = \text{Processing} \rrbracket$   
 $\implies P \vdash (\text{None}, h, f \# \text{frs}, sh) -\text{jvm} \rightarrow (\text{None}, h, (\text{calling-to-scaled } f) \# \text{frs}, sh)$   
**(is**  $\llbracket ?P; ?Q; ?R \rrbracket \implies P \vdash ?s1 -\text{jvm} \rightarrow ?s2$ )

$\langle \text{proof} \rangle$

**lemma** *jvm-InitError*:

$sh C = \llbracket (sfs, \text{Error}) \rrbracket$   
 $\implies P \vdash (\text{None}, h, (vs, ls, C_0, M, pc, \text{Calling } C \text{ Cs}) \# \text{frs}, sh)$   
 $\quad -\text{jvm} \rightarrow (\text{None}, h, (vs, ls, C_0, M, pc, \text{Throwing } Cs (\text{addr-of-sys-xcpt } \text{NoClassDefFoundError})) \# \text{frs}, sh)$

$\langle \text{proof} \rangle$

**lemma** *exec-ErrorThrowing*:

$sh C = \llbracket (sfs, \text{Error}) \rrbracket$   
 $\implies \text{exec } (P, (\text{None}, h, \text{calling-to-throwing } (stk, loc, D, M, pc, \text{Calling } C \text{ Cs}) a \# \text{frs}, sh))$   
 $\quad = \text{Some } (\text{None}, h, \text{calling-to-sthrowing } (stk, loc, D, M, pc, \text{Calling } C \text{ Cs}) a \# \text{frs}, sh)$

$\langle \text{proof} \rangle$

**lemma** *jvm-InitObj*:

$\llbracket sh C = \text{Some}(sfs, \text{Prepared});$   
 $\quad C = \text{Object};$   
 $\quad sh' = sh(C \mapsto (sfs, \text{Processing})) \rrbracket$   
 $\implies P \vdash (\text{None}, h, (vs, ls, C_0, M, pc, \text{Calling } C \text{ Cs}) \# \text{frs}, sh) -\text{jvm} \rightarrow$   
 $\quad (\text{None}, h, (vs, ls, C_0, M, pc, \text{Called } (C \# Cs)) \# \text{frs}, sh')$   
**(is**  $\llbracket ?P; ?Q; ?R \rrbracket \implies P \vdash ?s1 -\text{jvm} \rightarrow ?s2$ )

$\langle \text{proof} \rangle$

**lemma** *jvm-InitNonObj*:

$\llbracket sh C = \text{Some}(sfs, \text{Prepared});$   
 $\quad C \neq \text{Object};$   
 $\quad \text{class } P C = \text{Some } (D, r);$   
 $\quad sh' = sh(C \mapsto (sfs, \text{Processing})) \rrbracket$   
 $\implies P \vdash (\text{None}, h, (vs, ls, C_0, M, pc, \text{Calling } C \text{ Cs}) \# \text{frs}, sh) -\text{jvm} \rightarrow$   
 $\quad (\text{None}, h, (vs, ls, C_0, M, pc, \text{Calling } D (C \# Cs)) \# \text{frs}, sh')$   
**(is**  $\llbracket ?P; ?Q; ?R; ?S \rrbracket \implies P \vdash ?s1 -\text{jvm} \rightarrow ?s2$ )

$\langle \text{proof} \rangle$

**lemma** *jvm-RInit-throw*:

$P \vdash (\text{None}, h, (vs, l, C, M, pc, Throwing [] xa) \# frs, sh)$   
 $\quad -jvm \rightarrow \text{handle } P \ C \ M \ xa \ h \ vs \ l \ pc \ \text{No-ics } frs \ sh$   
 (is  $P \vdash ?s1 \ -jvm \rightarrow ?s2$ )  
 ⟨proof⟩

**lemma** *jvm-RInit-throw'*:

$P \vdash (\text{None}, h, (vs, l, C, M, pc, Throwing [C'] xa) \# frs, sh)$   
 $\quad -jvm \rightarrow \text{handle } P \ C \ M \ xa \ h \ vs \ l \ pc \ \text{No-ics } frs \ (sh(C' := \text{Some}(\text{fst}(\text{the}(sh \ C')), \text{Error})))$   
 (is  $P \vdash ?s1 \ -jvm \rightarrow ?s2$ )  
 ⟨proof⟩

**lemma** *jvm-Called*:

$P \vdash (\text{None}, h, (vs, l, C, M, pc, Called (C_0 \# Cs)) \# frs, sh) \ -jvm \rightarrow$   
 $(\text{None}, h, \text{create-init-frame } P \ C_0 \ \# (vs, l, C, M, pc, Called \ Cs) \ \# frs, sh)$   
 ⟨proof⟩

**lemma** *jvm-Throwing*:

$P \vdash (\text{None}, h, (vs, l, C, M, pc, Throwing (C_0 \# Cs) xa') \# frs, sh) \ -jvm \rightarrow$   
 $(\text{None}, h, (vs, l, C, M, pc, Throwing \ Cs \ xa') \ \# frs, sh(C_0 \mapsto (\text{fst}(\text{the}(sh \ C_0)), \text{Error})))$   
 ⟨proof⟩

## Other lemmas for correctness proof

**lemma** *assumes wf:wf-prog wf-md P*

**and** *ex: class P C = Some a*

**shows** *create-init-frame-wf-eq: create-init-frame (compP<sub>2</sub> P) C = (stk, loc, D, M, pc, ics)  $\implies$  D=C*  
 ⟨proof⟩

**lemma** *beforex-try*:

**assumes** *pcI: {pc..<pc+size(compE<sub>2</sub>(try e<sub>1</sub> catch(Ci i) e<sub>2</sub>))}  $\subseteq$  I*  
**and** *bx: P, C, M  $\triangleright$  compxE<sub>2</sub> (try e<sub>1</sub> catch(Ci i) e<sub>2</sub>) pc (size vs) / I, size vs*  
**shows** *P, C, M  $\triangleright$  compxE<sub>2</sub> e<sub>1</sub> pc (size vs) / {pc..<pc + length (compE<sub>2</sub> e<sub>1</sub>)}, size vs*  
 ⟨proof⟩

**lemma**

**shows** *eval<sub>1</sub>-init-return: P  $\vdash_1$  ⟨e, s⟩  $\Rightarrow$  ⟨e', s'⟩*  
 $\implies \text{iconf } (shp_1 \ s) \ e$   
 $\implies (\exists Cs \ b. e = \text{INIT } C' (Cs, b) \leftarrow \text{unit}) \vee (\exists C \ e_0 \ Cs \ e_i. e = \text{RI}(C, e_0); Cs @ [C'] \leftarrow \text{unit})$   
 $\quad \vee (\exists e_0. e = \text{RI}(C', e_0); \text{Nil} \leftarrow \text{unit})$   
 $\implies (\text{val-of } e' = \text{Some } v \longrightarrow (\exists sfs \ i. shp_1 \ s' \ C' = [(sfs, i)] \wedge (i = \text{Done} \vee i = \text{Processing})))$   
 $\quad \wedge (\text{throw-of } e' = \text{Some } a \longrightarrow (\exists sfs \ i. shp_1 \ s' \ C' = [(sfs, \text{Error})]))$   
**and** *P  $\vdash_1$  ⟨es, s⟩  $[\Rightarrow]$  ⟨es', s'⟩  $\implies$  True*  
 ⟨proof⟩

**lemma** *init<sub>1</sub>-Val-PD: P  $\vdash_1$  ⟨INIT C' (Cs, b)  $\leftarrow$  unit, s⟩  $\Rightarrow$  ⟨Val v, s'⟩*

$\implies \text{iconf } (shp_1 \ s) \ (\text{INIT } C' (Cs, b) \leftarrow \text{unit})$   
 $\implies \exists sfs \ i. shp_1 \ s' \ C' = [(sfs, i)] \wedge (i = \text{Done} \vee i = \text{Processing})$   
 ⟨proof⟩

**lemma** *init<sub>1</sub>-throw-PD: P  $\vdash_1$  ⟨INIT C' (Cs, b)  $\leftarrow$  unit, s⟩  $\Rightarrow$  ⟨throw a, s'⟩*

$\implies \text{iconf } (shp_1 \ s) \ (\text{INIT } C' (Cs, b) \leftarrow \text{unit})$   
 $\implies \exists sfs \ i. shp_1 \ s' \ C' = [(sfs, \text{Error})]$

$\langle \text{proof} \rangle$

**lemma** *rinit<sub>1</sub>-Val-PD*:

**assumes** *eval*:  $P \vdash_1 \langle RI(C, e_0); Cs \leftarrow \text{unit}, s \rangle \Rightarrow \langle \text{Val } v, s \rangle$

**and** *iconf*: *iconf* (*shp<sub>1</sub>* *s*) ( $RI(C, e_0); Cs \leftarrow \text{unit}$ ) **and** *last*:  $\text{last}(C \# Cs) = C'$

**shows**  $\exists \text{sfs } i. \text{shp}_1 \text{ } s' \text{ } C' = \lfloor (\text{sfs}, i) \rfloor \wedge (i = \text{Done} \vee i = \text{Processing})$

$\langle \text{proof} \rangle$

**lemma** *rinit<sub>1</sub>-throw-PD*:

**assumes** *eval*:  $P \vdash_1 \langle RI(C, e_0); Cs \leftarrow \text{unit}, s \rangle \Rightarrow \langle \text{throw } a, s \rangle$

**and** *iconf*: *iconf* (*shp<sub>1</sub>* *s*) ( $RI(C, e_0); Cs \leftarrow \text{unit}$ ) **and** *last*:  $\text{last}(C \# Cs) = C'$

**shows**  $\exists \text{sfs}. \text{shp}_1 \text{ } s' \text{ } C' = \lfloor (\text{sfs}, \text{Error}) \rfloor$

$\langle \text{proof} \rangle$

## The proof

**lemma** *fixes* *P<sub>1</sub>* **defines** [*simp*]:  $P \equiv \text{comp}P_2 \text{ } P_1$

**assumes** *wf*: *wf-J<sub>1</sub>-prog* *P<sub>1</sub>*

**shows** *Jcc*:  $P_1 \vdash_1 \langle e, (h_0, ls_0, sh_0) \rangle \Rightarrow \langle ef, (h_1, ls_1, sh_1) \rangle \Longrightarrow$

$(\bigwedge E \text{ } C \text{ } M \text{ } pc \text{ } ics \text{ } v \text{ } xa \text{ } vs \text{ } frs \text{ } I.$

$\llbracket Jcc\text{-cond } P_1 \text{ } E \text{ } C \text{ } M \text{ } vs \text{ } pc \text{ } ics \text{ } I \text{ } h_0 \text{ } sh_0 \text{ } e \rrbracket \Longrightarrow$

$(ef = \text{Val } v \longrightarrow$

$P \vdash (\text{None}, h_0, Jcc\text{-frames } P \text{ } C \text{ } M \text{ } vs \text{ } ls_0 \text{ } pc \text{ } ics \text{ } frs \text{ } e, sh_0)$

$\text{-jvm} \rightarrow Jcc\text{-rhs } P_1 \text{ } E \text{ } C \text{ } M \text{ } vs \text{ } ls_0 \text{ } pc \text{ } ics \text{ } frs \text{ } h_1 \text{ } ls_1 \text{ } sh_1 \text{ } v \text{ } e)$

$\wedge$

$(ef = \text{Throw } xa \longrightarrow Jcc\text{-err } P \text{ } C \text{ } M \text{ } h_0 \text{ } vs \text{ } ls_0 \text{ } pc \text{ } ics \text{ } frs \text{ } sh_0 \text{ } I \text{ } h_1 \text{ } ls_1 \text{ } sh_1 \text{ } xa \text{ } e)$

**and**  $P_1 \vdash_1 \langle es, (h_0, ls_0, sh_0) \rangle [\Rightarrow] \langle fs, (h_1, ls_1, sh_1) \rangle \Longrightarrow$

$(\bigwedge C \text{ } M \text{ } pc \text{ } ics \text{ } ws \text{ } xa \text{ } es' \text{ } vs \text{ } frs \text{ } I.$

$\llbracket P, C, M, pc \triangleright \text{comp}Es_2 \text{ } es; P, C, M \triangleright \text{comp}xEs_2 \text{ } es \text{ } pc \text{ } (\text{size } vs) / I, \text{size } vs;$

$\{pc..<pc+\text{size}(\text{comp}Es_2 \text{ } es)\} \subseteq I; ics = \text{No-ics};$

$\neg \text{sub-RIs } es \rrbracket \Longrightarrow$

$(fs = \text{map } \text{Val } ws \longrightarrow$

$P \vdash (\text{None}, h_0, (vs, ls_0, C, M, pc, ics) \# frs, sh_0) \text{-jvm} \rightarrow$

$(\text{None}, h_1, (\text{rev } ws \text{ } @ \text{ } vs, ls_1, C, M, pc + \text{size}(\text{comp}Es_2 \text{ } es), ics) \# frs, sh_1))$

$\wedge$

$(fs = \text{map } \text{Val } ws \text{ } @ \text{ } \text{Throw } xa \text{ } \# \text{ } es' \longrightarrow$

$(\exists pc_1. pc \leq pc_1 \wedge pc_1 < pc + \text{size}(\text{comp}Es_2 \text{ } es) \wedge$

$\neg \text{caught } P \text{ } pc_1 \text{ } h_1 \text{ } xa \text{ } (\text{comp}xEs_2 \text{ } es \text{ } pc \text{ } (\text{size } vs)) \wedge$

$(\exists vs'. P \vdash (\text{None}, h_0, (vs, ls_0, C, M, pc, ics) \# frs, sh_0)$

$\text{-jvm} \rightarrow \text{handle } P \text{ } C \text{ } M \text{ } xa \text{ } h_1 \text{ } (vs' \text{ } @ \text{ } vs) \text{ } ls_1 \text{ } pc_1 \text{ } ics \text{ } frs \text{ } sh_1)))) \langle \text{proof} \rangle$

**lemma** *atLeast0AtMost*[*simp*]:  $\{0::\text{nat}..n\} = \{..n\}$

$\langle \text{proof} \rangle$

**lemma** *atLeast0LessThan*[*simp*]:  $\{0::\text{nat}..<n\} = \{..<n\}$

$\langle \text{proof} \rangle$

**fun** *exception* :: 'a *exp*  $\Rightarrow$  *addr option* **where**

*exception* (*Throw* *a*) = *Some* *a*

| *exception* *e* = *None*

**lemma** *comp<sub>2</sub>-correct*:

**assumes** *wf*: *wf-J<sub>1</sub>-prog* *P<sub>1</sub>*

**and** *method*:  $P_1 \vdash C \text{ sees } M, b: Ts \rightarrow T = \text{body in } C$

**and eval:**  $P_1 \vdash_1 \langle \text{body}, (h, ls, sh) \rangle \Rightarrow \langle e', (h', ls', sh') \rangle$   
**and nclinit:**  $M \neq \text{clinit}$   
**shows**  $\text{comp}P_2 P_1 \vdash (\text{None}, h, [([], ls, C, M, 0, \text{No-ics})], sh) \text{ --jvm--} \rightarrow (\text{exception } e', h', [], sh') \langle \text{proof} \rangle$   
**end**

### 3.8 Combining Stages 1 and 2

**theory** *Compiler*  
**imports** *Correctness1 Correctness2*  
**begin**

**definition**  $J2JVM :: J\text{-prog} \Rightarrow \text{jvm-prog}$   
**where**  
 $J2JVM \equiv \text{comp}P_2 \circ \text{comp}P_1$

**theorem** *comp-correct-NonStatic:*

**assumes** *wf:*  $wf\text{-}J\text{-prog } P$   
**and method:**  $P \vdash C \text{ sees } M, \text{NonStatic}: Ts \rightarrow T = (pns, \text{body}) \text{ in } C$   
**and eval:**  $P \vdash \langle \text{body}, (h, [\text{this}\#pns \text{ --} \mapsto \text{ vs}], sh) \rangle \Rightarrow \langle e', (h', l', sh') \rangle$   
**and sizes:**  $\text{size } vs = \text{size } pns + 1 \quad \text{size } \text{rest} = \text{max-vars } \text{body}$   
**shows**  $J2JVM P \vdash (\text{None}, h, [([], vs@rest, C, M, 0, \text{No-ics})], sh) \text{ --jvm--} \rightarrow (\text{exception } e', h', [], sh') \langle \text{proof} \rangle$

**theorem** *comp-correct-Static:*

**assumes** *wf:*  $wf\text{-}J\text{-prog } P$   
**and method:**  $P \vdash C \text{ sees } M, \text{Static}: Ts \rightarrow T = (pns, \text{body}) \text{ in } C$   
**and eval:**  $P \vdash \langle \text{body}, (h, [pns \text{ --} \mapsto \text{ vs}], sh) \rangle \Rightarrow \langle e', (h', l', sh') \rangle$   
**and sizes:**  $\text{size } vs = \text{size } pns \quad \text{size } \text{rest} = \text{max-vars } \text{body}$   
**and nclinit:**  $M \neq \text{clinit}$   
**shows**  $J2JVM P \vdash (\text{None}, h, [([], vs@rest, C, M, 0, \text{No-ics})], sh) \text{ --jvm--} \rightarrow (\text{exception } e', h', [], sh') \langle \text{proof} \rangle$   
**end**

### 3.9 Preservation of Well-Typedness

**theory** *TypeComp*  
**imports** *Compiler ../BV/BVSpec*  
**begin**

**lemma** *max-stack1:*  $P, E \vdash_1 e :: T \Longrightarrow 1 \leq \text{max-stack } e \langle \text{proof} \rangle$

**locale**  $TC0 =$

**fixes**  $P :: J_1\text{-prog}$  **and**  $\text{maxl} :: \text{nat}$

**begin**

**definition**  $\text{ty } E e = (\text{THE } T. P, E \vdash_1 e :: T)$

**definition**  $\text{ty}_i E A' = \text{map } (\lambda i. \text{if } i \in A' \wedge i < \text{size } E \text{ then } \text{OK}(E!i) \text{ else } \text{Err}) [0..<\text{maxl}]$

**definition**  $\text{ty}_i' ST E A = (\text{case } A \text{ of } \text{None} \Rightarrow \text{None} \mid \_ [A'] \Rightarrow \text{Some}(ST, \text{ty}_i E A))$

**definition**  $\text{after } E A ST e = \text{ty}_i' ( \text{ty } E e \# ST ) E (A \sqcup A e)$

**end**

**lemma** **(in**  $TC0$ ) *ty-def2* [*simp*]:  $P, E \vdash_1 e :: T \Longrightarrow \text{ty } E e = T \langle \text{proof} \rangle$

**lemma** **(in**  $TC0$ ) [*simp*]:  $\text{ty}_i' ST E \text{None} = \text{None} \langle \text{proof} \rangle$

**lemma** (in *TC0*) *ty<sub>l</sub>-app-diff*[*simp*]:

$$ty_l (E@[T]) (A - \{size E\}) = ty_l E A \langle proof \rangle$$

**lemma** (in *TC0*) *ty<sub>i</sub>'-app-diff*[*simp*]:

$$ty_i' ST (E @ [T]) (A \ominus size E) = ty_i' ST E A \langle proof \rangle$$

**lemma** (in *TC0*) *ty<sub>l</sub>-antimono*:

$$A \subseteq A' \implies P \vdash ty_l E A' [\leq_{\top}] ty_l E A \langle proof \rangle$$

**lemma** (in *TC0*) *ty<sub>i</sub>'-antimono*:

$$A \subseteq A' \implies P \vdash ty_i' ST E [A'] \leq' ty_i' ST E [A] \langle proof \rangle$$

**lemma** (in *TC0*) *ty<sub>l</sub>-env-antimono*:

$$P \vdash ty_l (E@[T]) A [\leq_{\top}] ty_l E A \langle proof \rangle$$

**lemma** (in *TC0*) *ty<sub>i</sub>'-env-antimono*:

$$P \vdash ty_i' ST (E@[T]) A \leq' ty_i' ST E A \langle proof \rangle$$

**lemma** (in *TC0*) *ty<sub>i</sub>'-incr*:

$$P \vdash ty_i' ST (E @ [T]) [insert (size E) A] \leq' ty_i' ST E [A] \langle proof \rangle$$

**lemma** (in *TC0*) *ty<sub>l</sub>-incr*:

$$P \vdash ty_l (E @ [T]) (insert (size E) A) [\leq_{\top}] ty_l E A \langle proof \rangle$$

**lemma** (in *TC0*) *ty<sub>l</sub>-in-types*:

$$set E \subseteq types P \implies ty_l E A \in nlists\ max\ (err\ (types\ P)) \langle proof \rangle$$

**locale** *TC1 = TC0*

**begin**

**primrec** *compT* :: *ty list*  $\Rightarrow$  *nat hyperset*  $\Rightarrow$  *ty list*  $\Rightarrow$  *expr<sub>1</sub>*  $\Rightarrow$  *ty<sub>i</sub>' list* **and**

*compTs* :: *ty list*  $\Rightarrow$  *nat hyperset*  $\Rightarrow$  *ty list*  $\Rightarrow$  *expr<sub>1</sub> list*  $\Rightarrow$  *ty<sub>i</sub>' list* **where**

$$compT E A ST (new C) = []$$

$$| compT E A ST (Cast C e) =$$

$$compT E A ST e @ [after E A ST e]$$

$$| compT E A ST (Val v) = []$$

$$| compT E A ST (e_1 \ll bop \gg e_2) =$$

$$(let ST_1 = ty E e_1 \# ST; A_1 = A \sqcup \mathcal{A} e_1 \text{ in}$$

$$compT E A ST e_1 @ [after E A ST e_1] @$$

$$compT E A_1 ST_1 e_2 @ [after E A_1 ST_1 e_2])$$

$$| compT E A ST (Var i) = []$$

$$| compT E A ST (i := e) = compT E A ST e @$$

$$[after E A ST e, ty_i' ST E (A \sqcup \mathcal{A} e \sqcup [\{i\}])]$$

$$| compT E A ST (e \cdot F\{D\}) =$$

$$compT E A ST e @ [after E A ST e]$$

$$| compT E A ST (C \cdot_s F\{D\}) = []$$

$$| compT E A ST (e_1 \cdot F\{D\} := e_2) =$$

$$(let ST_1 = ty E e_1 \# ST; A_1 = A \sqcup \mathcal{A} e_1; A_2 = A_1 \sqcup \mathcal{A} e_2 \text{ in}$$

$$compT E A ST e_1 @ [after E A ST e_1] @$$

$$compT E A_1 ST_1 e_2 @ [after E A_1 ST_1 e_2] @$$

$$[ty_i' ST E A_2])$$

$$| compT E A ST (C \cdot_s F\{D\} := e_2) = compT E A ST e_2 @ [after E A ST e_2] @ [ty_i' ST E (A \sqcup \mathcal{A} e_2)]$$

$$| compT E A ST \{i:T; e\} = compT (E@[T]) (A \ominus i) ST e$$

$$| compT E A ST (e_1 ;; e_2) =$$

$$\begin{aligned}
& (\text{let } A_1 = A \sqcup \mathcal{A} e_1 \text{ in} \\
& \quad \text{compT } E A ST e_1 @ [\text{after } E A ST e_1, ty_i' ST E A_1] @ \\
& \quad \text{compT } E A_1 ST e_2) \\
| \text{compT } E A ST (\text{if } (e) e_1 \text{ else } e_2) = \\
& \quad (\text{let } A_0 = A \sqcup \mathcal{A} e; \tau = ty_i' ST E A_0 \text{ in} \\
& \quad \quad \text{compT } E A ST e @ [\text{after } E A ST e, \tau] @ \\
& \quad \quad \text{compT } E A_0 ST e_1 @ [\text{after } E A_0 ST e_1, \tau] @ \\
& \quad \quad \text{compT } E A_0 ST e_2) \\
| \text{compT } E A ST (\text{while } (e) c) = \\
& \quad (\text{let } A_0 = A \sqcup \mathcal{A} e; A_1 = A_0 \sqcup \mathcal{A} c; \tau = ty_i' ST E A_0 \text{ in} \\
& \quad \quad \text{compT } E A ST e @ [\text{after } E A ST e, \tau] @ \\
& \quad \quad \text{compT } E A_0 ST c @ [\text{after } E A_0 ST c, ty_i' ST E A_1, ty_i' ST E A_0]) \\
| \text{compT } E A ST (\text{throw } e) = \text{compT } E A ST e @ [\text{after } E A ST e] \\
| \text{compT } E A ST (e.M(es)) = \\
& \quad \text{compT } E A ST e @ [\text{after } E A ST e] @ \\
& \quad \text{compTs } E (A \sqcup \mathcal{A} e) (ty \ E e \# ST) es \\
| \text{compT } E A ST (C.sM(es)) = \text{compTs } E A ST es \\
| \text{compT } E A ST (\text{try } e_1 \text{ catch } (C i) e_2) = \\
& \quad \text{compT } E A ST e_1 @ [\text{after } E A ST e_1] @ \\
& \quad [ty_i' (Class C\#ST) E A, ty_i' ST (E@[Class C]) (A \sqcup [\{i\}])] @ \\
& \quad \text{compT } (E@[Class C]) (A \sqcup [\{i\}]) ST e_2 \\
| \text{compT } E A ST (INIT C (Cs,b) \leftarrow e) = [] \\
| \text{compT } E A ST (RI(C,e'); Cs \leftarrow e) = [] \\
| \text{compTs } E A ST [] = [] \\
| \text{compTs } E A ST (e\#es) = \text{compT } E A ST e @ [\text{after } E A ST e] @ \\
\quad \quad \text{compTs } E (A \sqcup (\mathcal{A} e)) (ty \ E e \# ST) es
\end{aligned}$$

**definition**  $\text{compT}_a :: \text{ty list} \Rightarrow \text{nat hyperset} \Rightarrow \text{ty list} \Rightarrow \text{expr}_1 \Rightarrow \text{ty}_i' \text{ list}$  **where**  
 $\text{compT}_a E A ST e = \text{compT } E A ST e @ [\text{after } E A ST e]$

**end**

**lemma**  $\text{compE}_2\text{-not-Nil}[\text{simp}]$ :  $P, E \vdash_1 e :: T \Longrightarrow \text{compE}_2 e \neq [] \langle \text{proof} \rangle$

**lemma** (in TC1)  $\text{compT-sizes}'$ :

**shows**  $\bigwedge E A ST. \neg \text{sub-RI } e \Longrightarrow \text{size}(\text{compT } E A ST e) = \text{size}(\text{compE}_2 e) - 1$

**and**  $\bigwedge E A ST. \neg \text{sub-RIs } es \Longrightarrow \text{size}(\text{compTs } E A ST es) = \text{size}(\text{compEs}_2 es) \langle \text{proof} \rangle$

**lemma** (in TC1)  $\text{compT-sizes}[\text{simp}]$ :

**shows**  $\bigwedge E A ST. P, E \vdash_1 e :: T \Longrightarrow \text{size}(\text{compT } E A ST e) = \text{size}(\text{compE}_2 e) - 1$

**and**  $\bigwedge E A ST. P, E \vdash_1 es [::] Ts \Longrightarrow \text{size}(\text{compTs } E A ST es) = \text{size}(\text{compEs}_2 es) \langle \text{proof} \rangle$

**lemma** (in TC1)  $[\text{simp}]$ :  $\bigwedge ST E. [\tau] \notin \text{set}(\text{compT } E \text{ None } ST e)$

**and**  $[\text{simp}]$ :  $\bigwedge ST E. [\tau] \notin \text{set}(\text{compTs } E \text{ None } ST es) \langle \text{proof} \rangle$

**lemma** (in TC0)  $\text{pair-eq-ty}_i'\text{-conv}$ :

$([(ST, LT)] = ty_i' ST_0 E A) =$

$(\text{case } A \text{ of None} \Rightarrow \text{False} \mid \text{Some } A \Rightarrow (ST = ST_0 \wedge LT = ty_l E A)) \langle \text{proof} \rangle$

**lemma** (in TC0)  $\text{pair-conv-ty}_i'$ :

$[(ST, ty_l E A)] = ty_i' ST E [A] \langle \text{proof} \rangle$

**lemma** (in TC1)  $\text{compT-LT-prefix}$ :

$\bigwedge E A ST_0. \llbracket [(ST, LT)] \in \text{set}(\text{compT } E A ST_0 e); \mathcal{B} e (\text{size } E) \rrbracket$

$\Longrightarrow P \vdash [(ST, LT)] \leq' ty_i' ST E A$

**and**



$\bigwedge E A ST_0. \llbracket [(ST,LT)] \in \text{set}(\text{compTs } E A ST_0 \text{ es}); \mathcal{B}s \text{ es } (\text{size } E) \rrbracket$   
 $\implies P \vdash [(ST,LT)] \leq' \text{ty}_i' ST E A \langle \text{proof} \rangle$

**lemma** [iff]: *OK None*  $\in$  *states* *P mxs mxl*  $\langle \text{proof} \rangle$

**lemma** (in *TC0*) *after-in-states*:

**assumes** *wf*: *wf-prog* *p P* **and** *wt*:  $P, E \vdash_1 e :: T$

**and** *Etypes*:  $\text{set } E \subseteq \text{types } P$  **and** *STtypes*:  $\text{set } ST \subseteq \text{types } P$

**and** *stack*:  $\text{size } ST + \text{max-stack } e \leq \text{mxs}$

**shows** *OK* (after *E A ST e*)  $\in$  *states* *P mxs mxl*  $\langle \text{proof} \rangle$

**lemma** (in *TC0*) *OK-ty<sub>i</sub>'-in-statesI* [simp]:

$\llbracket \text{set } E \subseteq \text{types } P; \text{set } ST \subseteq \text{types } P; \text{size } ST \leq \text{mxs} \rrbracket$

$\implies \text{OK } (\text{ty}_i' ST E A) \in \text{states } P \text{ mxs mxl} \langle \text{proof} \rangle$

**lemma** *is-class-type-aux*: *is-class* *P C*  $\implies$  *is-type* *P* (*Class* *C*)  $\langle \text{proof} \rangle$

**theorem** (in *TC1*) *compT-states*:

**assumes** *wf*: *wf-prog* *p P*

**shows**  $\bigwedge E T A ST.$

$\llbracket P, E \vdash_1 e :: T; \text{set } E \subseteq \text{types } P; \text{set } ST \subseteq \text{types } P;$   
 $\text{size } ST + \text{max-stack } e \leq \text{mxs}; \text{size } E + \text{max-vars } e \leq \text{mxl} \rrbracket$

$\implies \text{OK } \text{'set}(\text{compT } E A ST e) \subseteq \text{states } P \text{ mxs mxl}$

**and**  $\bigwedge E Ts A ST.$

$\llbracket P, E \vdash_1 \text{es}[::] Ts; \text{set } E \subseteq \text{types } P; \text{set } ST \subseteq \text{types } P;$   
 $\text{size } ST + \text{max-stacks } \text{es} \leq \text{mxs}; \text{size } E + \text{max-varss } \text{es} \leq \text{mxl} \rrbracket$

$\implies \text{OK } \text{'set}(\text{compTs } E A ST \text{es}) \subseteq \text{states } P \text{ mxs mxl} \langle \text{proof} \rangle$

**definition** *shift* :: *nat*  $\Rightarrow$  *ex-table*  $\Rightarrow$  *ex-table*

**where**

*shift* *n xt*  $\equiv$  *map* ( $\lambda(\text{from}, \text{to}, C, \text{handler}, \text{depth}). (\text{from} + n, \text{to} + n, C, \text{handler} + n, \text{depth})) \text{xt}$

**lemma** [simp]: *shift* 0 *xt* = *xt*  $\langle \text{proof} \rangle$

**lemma** [simp]: *shift* *n* [] = []  $\langle \text{proof} \rangle$

**lemma** [simp]: *shift* *n* (*xt*<sub>1</sub> @ *xt*<sub>2</sub>) = *shift* *n* *xt*<sub>1</sub> @ *shift* *n* *xt*<sub>2</sub>  $\langle \text{proof} \rangle$

**lemma** [simp]: *shift* *m* (*shift* *n* *xt*) = *shift* (*m* + *n*) *xt*  $\langle \text{proof} \rangle$

**lemma** [simp]: *pcs* (*shift* *n* *xt*) = {*pc* + *n* | *pc*. *pc*  $\in$  *pcs* *xt*}  $\langle \text{proof} \rangle$

**lemma** *shift-compxE<sub>2</sub>*:

**shows**  $\bigwedge pc pc' d. \text{shift } pc (\text{compxE}_2 e pc' d) = \text{compxE}_2 e (pc' + pc) d$

**and**  $\bigwedge pc pc' d. \text{shift } pc (\text{compxEs}_2 \text{es } pc' d) = \text{compxEs}_2 \text{es } (pc' + pc) d \langle \text{proof} \rangle$

**lemma** *compxE<sub>2</sub>-size-convs* [simp]:

**shows**  $n \neq 0 \implies \text{compxE}_2 e n d = \text{shift } n (\text{compxE}_2 e 0 d)$

**and**  $n \neq 0 \implies \text{compxEs}_2 \text{es } n d = \text{shift } n (\text{compxEs}_2 \text{es } 0 d) \langle \text{proof} \rangle$

**locale** *TC2* = *TC1* +

**fixes** *T<sub>r</sub>* :: *ty* **and** *mxs* :: *pc*

**begin**

**definition**

*wt-instrs* :: *instr list*  $\Rightarrow$  *ex-table*  $\Rightarrow$  *ty<sub>i</sub>' list*  $\Rightarrow$  *bool*

( $\langle \vdash -, - /[::] / - \rangle [0, 0, 51] 50$ ) **where**

$\vdash \text{is}, \text{xt} [::] \tau s \iff \text{size } \text{is} < \text{size } \tau s \wedge \text{pcs } \text{xt} \subseteq \{0..<\text{size } \text{is}\} \wedge$

( $\forall pc < \text{size } \text{is}. P, T_r, \text{mxs}, \text{size } \tau s, \text{xt} \vdash \text{is!pc}, pc :: \tau s$ )

end

**notation**  $TC2.wt-instrs$  ( $\langle(-,-,- \vdash / -, - / [::] / -) \rangle$  [50,50,50,50,50,51] 50)

**lemma** (in  $TC2$ ) [simp]:  $\tau s \neq [] \implies \vdash [], [] [::] \tau s \langle proof \rangle$

**lemma** [simp]:  $eff\ i\ P\ pc\ et\ None = [] \langle proof \rangle$

**lemma**  $wt-instr-appR$ :

$\llbracket P, T, m, mpc, xt \vdash is!pc, pc :: \tau s;$   
 $pc < size\ is; size\ is < size\ \tau s; mpc \leq size\ \tau s; mpc \leq mpc' \rrbracket$   
 $\implies P, T, m, mpc', xt \vdash is!pc, pc :: \tau s @ \tau s' \langle proof \rangle$

**lemma**  $relevant-entries-shift$  [simp]:

$relevant-entries\ P\ i\ (pc+n)\ (shift\ n\ xt) = shift\ n\ (relevant-entries\ P\ i\ pc\ xt) \langle proof \rangle$

**lemma** [simp]:

$xcpt-eff\ i\ P\ (pc+n)\ \tau\ (shift\ n\ xt) =$   
 $map\ (\lambda(pc, \tau). (pc + n, \tau))\ (xcpt-eff\ i\ P\ pc\ \tau\ xt) \langle proof \rangle$

**lemma** [simp]:

$app_i\ (i, P, pc, m, T, \tau) \implies$   
 $eff\ i\ P\ (pc+n)\ (shift\ n\ xt)\ (Some\ \tau) =$   
 $map\ (\lambda(pc, \tau). (pc+n, \tau))\ (eff\ i\ P\ pc\ xt\ (Some\ \tau)) \langle proof \rangle$

**lemma** [simp]:

$xcpt-app\ i\ P\ (pc+n)\ mxs\ (shift\ n\ xt)\ \tau = xcpt-app\ i\ P\ pc\ mxs\ xt\ \tau \langle proof \rangle$

**lemma**  $wt-instr-appL$ :

**assumes**  $P, T, m, mpc, xt \vdash i, pc :: \tau s$  **and**  $pc < size\ \tau s$  **and**  $mpc \leq size\ \tau s$   
**shows**  $P, T, m, mpc + size\ \tau s', shift\ (size\ \tau s')\ xt \vdash i, pc + size\ \tau s' :: \tau s' @ \tau s \langle proof \rangle$

**lemma**  $wt-instr-Cons$ :

**assumes**  $wti: P, T, m, mpc - 1, [] \vdash i, pc - 1 :: \tau s$   
**and**  $pcl: 0 < pc$  **and**  $mpcl: 0 < mpc$   
**and**  $pcu: pc < size\ \tau s + 1$  **and**  $mpcu: mpc \leq size\ \tau s + 1$   
**shows**  $P, T, m, mpc, [] \vdash i, pc :: \tau \# \tau s \langle proof \rangle$

**lemma**  $wt-instr-append$ :

**assumes**  $wti: P, T, m, mpc - size\ \tau s', [] \vdash i, pc - size\ \tau s' :: \tau s$   
**and**  $pcl: size\ \tau s' \leq pc$  **and**  $mpcl: size\ \tau s' \leq mpc$   
**and**  $pcu: pc < size\ \tau s + size\ \tau s'$  **and**  $mpcu: mpc \leq size\ \tau s + size\ \tau s'$   
**shows**  $P, T, m, mpc, [] \vdash i, pc :: \tau s' @ \tau s \langle proof \rangle$

**lemma**  $xcpt-app-pcs$ :

$pc \notin pcs\ xt \implies xcpt-app\ i\ P\ pc\ mxs\ xt\ \tau \langle proof \rangle$

**lemma**  $xcpt-eff-pcs$ :

$pc \notin pcs\ xt \implies xcpt-eff\ i\ P\ pc\ \tau\ xt = [] \langle proof \rangle$

**lemma**  $pcs-shift$ :

$pc < n \implies pc \notin pcs\ (shift\ n\ xt) \langle proof \rangle$

**lemma**  $wt-instr-appRx$ :

$\llbracket P, T, m, mpc, xt \vdash is!pc, pc :: \tau s; pc < size\ is; size\ is < size\ \tau s; mpc \leq size\ \tau s \rrbracket$   
 $\implies P, T, m, mpc, xt @ shift\ (size\ is)\ xt' \vdash is!pc, pc :: \tau s \langle proof \rangle$

**lemma** *wt-instr-appLx*:

$\llbracket P, T, m, mpc, xt \vdash i, pc :: \tau s; pc \notin pcs \ xt' \rrbracket$   
 $\implies P, T, m, mpc, xt' @ xt \vdash i, pc :: \tau s \langle proof \rangle$

**lemma** (in *TC2*) *wt-instrs-extR*:

$\vdash is, xt \ [::] \ \tau s \implies \vdash is, xt \ [::] \ \tau s \ @ \ \tau s' \langle proof \rangle$

**lemma** (in *TC2*) *wt-instrs-ext*:

**assumes**  $wt_1: \vdash is_1, xt_1 \ [::] \ \tau s_1 @ \tau s_2$  **and**  $wt_2: \vdash is_2, xt_2 \ [::] \ \tau s_2$   
**and**  $\tau s\text{-size}: size \ \tau s_1 = size \ is_1$   
**shows**  $\vdash is_1 @ is_2, xt_1 \ @ \ shift \ (size \ is_1) \ xt_2 \ [::] \ \tau s_1 @ \tau s_2 \langle proof \rangle$

**corollary** (in *TC2*) *wt-instrs-ext2*:

$\llbracket \vdash is_2, xt_2 \ [::] \ \tau s_2; \vdash is_1, xt_1 \ [::] \ \tau s_1 @ \tau s_2; size \ \tau s_1 = size \ is_1 \rrbracket$   
 $\implies \vdash is_1 @ is_2, xt_1 \ @ \ shift \ (size \ is_1) \ xt_2 \ [::] \ \tau s_1 @ \tau s_2 \langle proof \rangle$

**corollary** (in *TC2*) *wt-instrs-ext-prefix* [*trans*]:

$\llbracket \vdash is_1, xt_1 \ [::] \ \tau s_1 @ \tau s_2; \vdash is_2, xt_2 \ [::] \ \tau s_3;$   
 $size \ \tau s_1 = size \ is_1; prefix \ \tau s_3 \ \tau s_2 \rrbracket$   
 $\implies \vdash is_1 @ is_2, xt_1 \ @ \ shift \ (size \ is_1) \ xt_2 \ [::] \ \tau s_1 @ \tau s_2 \langle proof \rangle$

**corollary** (in *TC2*) *wt-instrs-app*:

**assumes**  $is_1: \vdash is_1, xt_1 \ [::] \ \tau s_1 @ [\tau]$   
**assumes**  $is_2: \vdash is_2, xt_2 \ [::] \ \tau \# \tau s_2$   
**assumes**  $s: size \ \tau s_1 = size \ is_1$   
**shows**  $\vdash is_1 @ is_2, xt_1 @ shift \ (size \ is_1) \ xt_2 \ [::] \ \tau s_1 @ \tau \# \tau s_2 \langle proof \rangle$

**corollary** (in *TC2*) *wt-instrs-app-last* [*trans*]:

**assumes**  $\vdash is_2, xt_2 \ [::] \ \tau \# \tau s_2 \vdash is_1, xt_1 \ [::] \ \tau s_1$   
 $last \ \tau s_1 = \tau \ size \ \tau s_1 = size \ is_1 + 1$   
**shows**  $\vdash is_1 @ is_2, xt_1 @ shift \ (size \ is_1) \ xt_2 \ [::] \ \tau s_1 @ \tau s_2 \langle proof \rangle$

**corollary** (in *TC2*) *wt-instrs-append-last* [*trans*]:

**assumes**  $wtis: \vdash is, xt \ [::] \ \tau s$  **and**  $wti: P, T_r, mxs, mpc, [] \vdash i, pc :: \tau s$   
**and**  $pc: pc = size \ is$  **and**  $mpc: mpc = size \ \tau s$  **and**  $is\text{-}\tau s: size \ is + 1 < size \ \tau s$   
**shows**  $\vdash is @ [i], xt \ [::] \ \tau s \langle proof \rangle$

**corollary** (in *TC2*) *wt-instrs-app2*:

$\llbracket \vdash is_2, xt_2 \ [::] \ \tau' \# \tau s_2; \vdash is_1, xt_1 \ [::] \ \tau \# \tau s_1 @ [\tau'];$   
 $xt' = xt_1 \ @ \ shift \ (size \ is_1) \ xt_2; size \ \tau s_1 + 1 = size \ is_1 \rrbracket$   
 $\implies \vdash is_1 @ is_2, xt' \ [::] \ \tau \# \tau s_1 @ \tau' \# \tau s_2 \langle proof \rangle$

**corollary** (in *TC2*) *wt-instrs-app2-simp* [*trans, simp*]:

$\llbracket \vdash is_2, xt_2 \ [::] \ \tau' \# \tau s_2; \vdash is_1, xt_1 \ [::] \ \tau \# \tau s_1 @ [\tau']; size \ \tau s_1 + 1 = size \ is_1 \rrbracket$   
 $\implies \vdash is_1 @ is_2, xt_1 @ shift \ (size \ is_1) \ xt_2 \ [::] \ \tau \# \tau s_1 @ \tau' \# \tau s_2 \langle proof \rangle$

**corollary** (in *TC2*) *wt-instrs-Cons* [*simp*]:

$\llbracket \tau s \neq []; \vdash [i], [] \ [::] \ [\tau, \tau']; \vdash is, xt \ [::] \ \tau' \# \tau s \rrbracket$   
 $\implies \vdash i \# is, shift \ 1 \ xt \ [::] \ \tau \# \tau' \# \tau s \langle proof \rangle \langle proof \rangle \langle proof \rangle \langle proof \rangle \langle proof \rangle \langle proof \rangle \langle proof \rangle \langle proof \rangle \langle proof \rangle \langle proof \rangle \langle proof \rangle$

**theory** *JinjaDCI*

**imports**

*J/Equivalence*

*J/Annotate*

*JVM/JVMDefensive*

180

*BV/BVExec*

*BV/LBVJVM*

*BV/BVNoTypeError*

*Compiler/TypeComp*

**begin**

**end**

# Bibliography

- [1] S. Mansky and E. L. Gunter. Dynamic class initialization semantics: A jinja extension. In *Proceedings of the 8th ACM SIGPLAN International Conference on Certified Programs and Proofs*, CPP 2019, pages 209–221, New York, NY, USA, 2019. Association for Computing Machinery.