1 Some Sugar

Unless stated otherwise, all names introduced in the right-hand side of a definition are assumed not to appear free in the left-hand side.

1.1 Blocking Calls

Let \( r / \in \text{fn}(w_1 \cdots w_n P) \).

\[
(v \leftarrow \pi(w_1 \cdots w_n); P) = \text{new } r(\pi(w_1 \cdots w_n r).r(v).P)
\]

1.2 Values

Let \( I \) be some finite set. Let \( \bar{w} \) be a list of names that contains one name \( w_j \) for each \( j \in I \). Let one such \( \bar{w} \) be chosen for each finite set \( I \). Let the names in \( \bar{w} \) not be used anywhere else.

Let \( i \in I \).

\[
\{i\} = \text{new } \bar{v}.(\bar{v} w_i) \]

\[
(i : I)P_i = (v)(\text{new } \bar{w}(\pi(\bar{w}) | \sum_{i \in I} w_i.P_i))
\]

The syntax \((i)P_i\) will be used if no confusion is possible.

1.3 Datatypes

Let data \( T = C_1^{m_1} | \cdots | C_n^{m_n} \).

\[
(C_i v_1 \cdots v_{m_i}) = \text{new } v_1(v_1 \cdots v_{m_i}) \pi v_1 \cdots v_{m_i})
\]

Let \( w_1, \ldots, w_n \) not be used in \( P_1, \ldots, P_n \).

\[
\left(\begin{array}{c}
C_1 v_1^1 \cdots v_{m_1}^1 \Rightarrow P_1 \\
\vdots \\
C_n v_n^1 \cdots v_{m_n}^n \Rightarrow P_n 
\end{array}\right)
\]

\[
= \text{new } w_1 \cdots w_n \pi(w_1 \cdots w_n) \sum_i w_i(v_1^i \cdots v_{m_i}^i).P_i
\]
1.3.1 Some Datatypes

\[
data \text{nat} = \text{Zero} \mid \text{Succ nat}\]

1.3.2 Equality on Pure Datatypes

A datatype \(T_i\) is pure if there is a set \(S = \{T_1, \ldots, T_n\}\) of datatypes such that the types of all parameters of all constructors of all datatypes in \(S\) are in \(S\).

\[
P[v_1 = v_2] = \text{new } eq_1 \cdots eq_n (Q_1 | \cdots | Q_n | (b \leftrightarrow \varphi_i(v_1 v_2); P[b]))
\]

with

\[
Q_i = !eq_i(v_1 v_2 r).R_i
\]

We shall define \(R_i\) by example. Suppose \(T_i = \text{nat}\).

\[
\begin{align*}
R_i = \text{case } v_1 \text{ of } &\begin{cases}
\text{Zero} &\Rightarrow \text{case } v_2 \text{ of } \begin{cases}
\text{Zero} &\Rightarrow \varphi(\text{true}) \\
\text{Succ } w_2 &\Rightarrow \varphi(\text{false})
\end{cases} \\
\text{Succ } w_1 &\Rightarrow \text{case } v_2 \text{ of } \begin{cases}
\text{Zero} &\Rightarrow \varphi(\text{false}) \\
\text{Succ } w_2 &\Rightarrow (b \leftrightarrow \varphi_i(w_1 w_2); \varphi(b))
\end{cases}
\end{cases}
\end{align*}
\]

1.4 Structures

Let \(\text{struct } T = \{l_1 \cdots l_n\}\).

\[
\langle \{l_1 \leftarrow v_1, \ldots, l_n \leftarrow v_n\} \rangle = \text{new } v.(\varphi(v_1 \cdots v_n))
\]

\[
P[v.l_i] = v(v_1 \cdots v_n).P[v_i]
\]

1.5 Variables

A variable \(x\) with initial value \(v\) is introduced as follows: \(\text{new } x.(\varphi(v).P)\).

\[
(v \leftarrow [x]; P) = x(v).(\varphi(v) \mid P)
\]

Let \(w \notin \text{fn}(P, v, x)\).

\[
([x] \leftarrow v; P) = x(w).(\varphi(v) \mid P)
\]
2 Memory Model

\[ \text{struct context} = \{\text{context-identity, get-local, lock-locals, unlock-locals}\} \]

\[ \text{data list } \alpha = \text{Nil} \mid \text{Cons } \alpha (\text{list } \alpha) \]

\[ \text{GetTableCell} = !h(i \triangleright r). (n \leftarrow [t]; \text{ case } n \text{ of } \{ \begin{array}{ll}
\text{Nil} & \Rightarrow G_1 \\
\text{Cons } c \ t' & \Rightarrow G_2
\end{array} \} ) \]

\[ G_1 = \text{new } ct' ([t] \leftarrow \text{Cons } c \ t'; (\tau(\text{Nothing}) | \tau(\text{Nil}) | G_2) \]

\[ G_2 = \text{case } i \text{ of } \{ \begin{array}{ll}
\text{Zero} & \Rightarrow \tau(c) \\
\text{Succ } i' & \Rightarrow \tau(i' \ t' \ r)
\end{array} \] \]

\[ \text{Local} = \text{new } a \ r \ g \ p \ e \ d \ (\tau(a \ r) . (l(c.E + c.(v).C + d.(v).D) | \tau | \tau | \tau)) \]

\[ E = l(v).\overline{a}(v) + a.r + r.a + g.(v \leftarrow [m]; \tau(v)) \]

\[ C = l(v).\overline{c}(v) + l(v).\overline{d}(v) + a.r + r.a + g.(v \leftarrow [m]; \tau(v)) \]

\[ D = l(v).\overline{d}(v) + l(v).\overline{d}(v) + a.a + p.(m \leftarrow v; \tau(v)) \]

\[ \text{NewContext}(P) = \text{new } \text{start} (\text{start} (\text{context}), P \mid \text{new } t g x l u C_1) \]

\[ C_1 = \text{identity-server}(n). (\text{start} (\{ \text{context-identity} \leftarrow n, \text{get-local} \leftarrow g, \text{lock-locals} \leftarrow l, \text{unlock-locals} \leftarrow u \}) \mid \overline{t}(\text{Nil}) . (C_2 \mid C_4) \]

\[ C_2 = !g(f \ r) . f(n \ m) . \text{new } h \ (\text{GetTableCell} | (c \leftarrow \overline{h}(n \ t); C_3) \]

\[ C_3 = (v \leftarrow [c]; \text{ case } v \text{ of } \{ \begin{array}{ll}
\text{Nothing} & \Rightarrow \text{new } l (\text{Local} | ([c] \leftarrow \text{Just } l; \tau(l))) \\
\text{Just } l & \Rightarrow \tau(l)
\end{array} \} ) \]

\[ \text{struct lockchannels} = \{ \text{lock, unlock} \} \]

\[ C_4 = \text{new } m \ (l(m) . (x(l \ u) . X + l(r) . L + u(r) . U) \mid \tau(\text{Nil})) \]
\[ X = \overline{m} (\text{Cons} \{ \text{lock} \leftarrow l, \text{unlock} \leftarrow u \} \ n) \]

\[ L = \text{new } i(l(n')).(\text{case } n' \text{ of } \begin{cases} \text{Nil} & \Rightarrow \overline{m}(n) \ \overline{n}.\text{lock}.\overline{i}(t) \mid \overline{i}(n) \\ \text{Cons } h \ t & \Rightarrow \overline{m}(n) \ \overline{n}.\text{unlock}.\overline{i}(t) \mid \overline{i}(n) \end{cases}) \]

\[ U = \text{new } i(l(n')).(\text{case } n' \text{ of } \begin{cases} \text{Nil} & \Rightarrow \overline{m}(n) \ \overline{n}.\text{lock}.\overline{i}(t) \mid \overline{i}(n) \\ \text{Cons } h \ t & \Rightarrow \overline{m}(n) \ \overline{n}.\text{unlock}.\overline{i}(t) \mid \overline{i}(n) \end{cases}) \]

3 Expressions

In this section we define a translation that associates a Java expression \( E \) appearing in a Java program \( P \) with a \( \pi \)-calculus process \( \llbracket E \rrbracket_P \). We will usually omit the subscript.

These are the upcast channels \( U \):

\[ U = \{ \uparrow T \mid T \text{ is a type name} \} \]

These are the class channels \( C \):

\[ C = \{ \text{new, new-public-super, new-internal-super} \} \cup \{ s \mid s \text{ is a signature} \} \cup U \]

These are the program channels \( P \):

\[ P = \{ \text{identity-server} \} \cup \{ T::c \mid T \text{ is a type name}, c \in C \} \]

These are the expression channels \( E \):

\[ E = \{ \text{done, break, return, throw, context, this} \} \cup \{ n \mid n \text{ is a local variable name} \} \cup P \]

The following holds for any expression \( E \):

\[ \text{fn}(\llbracket E \rrbracket) \subseteq E \]

3.1 Some Sugar

The use of the term local in the following definition refers to thread-local working copies of fields, rather than to block-local variables.

\[
\text{struct context} = \{ \text{context-identity, get-local, lock-locals, unlock-locals} \}
\]
struct object = \{identity : nat, lock, lock-owner, private-fields, internal-fields, internal-methods, public-fields, public-methods, super, dynamic-cast\}

data ref = Null | Object object

\( (v \leftarrow P; Q) = \text{new start } (\text{new done } (P \mid \text{done}(w), \text{start}(w)) \mid \text{start}(v), Q) \)

\( \text{NotNull}(P) = \text{case } r \text{ of } \{ \text{Null } \Rightarrow \text{ThrowNPE} \quad \text{Object } o \Rightarrow P \} \)

with

\( \text{ThrowNPE} = (r \leftarrow [\text{new NullPointerException()}; \overline{\text{throw}}(r)]) \)

3.2 Literal Expressions

Let \( i \) be a literal of type \( \text{int} \).

\[ [i] = \overline{\text{done}}(i) \]

3.3 Operators on Primitive Values

Let \( E_1 \) and \( E_2 \) be expressions of type \( \text{int} \).

\[ [E_1 + E_2] = (i_1 \leftarrow [E_1]; i_2 \leftarrow [E_2]; \overline{\text{done}}(i_1 + i_2)) \]

3.4 Reference Comparison

\[ [E_1 == E_2] = (r_1 \leftarrow [E_1]; r_2 \leftarrow [E_2]; \text{case } r_1 \text{ of } \{ \text{Null } \Rightarrow P_1 \quad \text{Object } o_1 \Rightarrow P_2 \} \)

with

\[ P_1 = \text{case } r_2 \text{ of } \{ \text{Null } \Rightarrow \overline{\text{done}}(\overline{\text{true}}) \quad \text{Object } o_2 \Rightarrow \overline{\text{done}}(\overline{\text{false}}) \} \]

\[ P_2 = \text{case } r_2 \text{ of } \{ \text{Null } \Rightarrow \overline{\text{done}}(\overline{\text{false}}) \quad \text{Object } o_2 \Rightarrow \overline{\text{done}}(o_1.\text{identity} = o_2.\text{identity}) \} \]
3.5 The Conditional Expression

Let $E_C$ be an expression of type boolean and let $E_T$ and $E_F$ be expressions of some type $T$.

$$[E_C \ ? \ E_T : E_F] = (b \leftarrow [E_C]; (b \ ? \ [E_T] : [E_F]))$$

3.6 This

$$[\text{this}] = \overline{\text{done}}(\text{Object this})$$

3.7 Local Variables

$$[n] = n(v).(\overline{\pi}(v) \ | \ \overline{\text{done}}(v))$$

$$[n = E] = (v \leftarrow [E]; n(w).(\overline{\pi}(w) \ | \ \overline{\text{done}}(w)))$$

3.8 Fields

Let the static type of expression $E$ be $C$, where $C$ is a class declared in program $P$.

Let $a$ be private, internal, or public, according to the accessibility of field $f$ in class $C$.

$$\text{GetField}(p, C, f, P) = \text{NotNull}(l \leftarrow \text{context.get-local}(o.a-fields.f); P)$$

$$[E.f]_p = (r \leftarrow [E]; \text{GetField}(p, C, f, (l(v)\.\overline{\text{done}}(v))))$$

Suppose the following expression appears in a class $D$ that extends a class $C$:

$$[\text{super}.f] = (\langle(C) \ this\rangle.f)$$

$$[E_1.f = E_2] = (r \leftarrow [E_1]; v \leftarrow [E_2]; \text{GetField}(p, C, f, (l(v)\.\overline{\text{done}}(v))))$$

3.9 Static Method Invocation

Suppose in the following expression overload resolution selects the method with signature $s$.

$$[C.m(E_1, \ldots , E_n)] = (v_1 \leftarrow [E_1]; \cdots; v_n \leftarrow [E_n]; C::s(context \ done \ throw \ v_1 \ \cdots \ v_n))$$
3.10 Instance Method Invocation

Suppose in the following expression the type of \( E \) is \( T \) and overload resolution selects the method with signature \( s \).

If the method is private:

\[
[E.m(E_1, \ldots, E_n)] = \langle \begin{array}{ll}
    r & \leftarrow [E]; \ v_1 & \leftarrow [E_1]; \ \cdots ; \ v_n & \leftarrow [E_n]; \ NotNull(T::s(\text{context done throw } o \ v_1 \ \cdots \ v_n))
\end{array} \rangle
\]

Otherwise, let \( a \) be internal or public, as appropriate.

\[
[E.m(E_1, \ldots, E_n)] = \langle \begin{array}{ll}
    r & \leftarrow [E]; \ v_1 & \leftarrow [E_1]; \ \cdots ; \ v_n & \leftarrow [E_n]; \ NotNull(o.a-methods.s(\text{context done throw } v_1 \ \cdots \ v_n))
\end{array} \rangle
\]

3.11 Super Invocation

Let \( S \) be the immediate superclass of the class in which the expression appears.

Suppose in the following expression overload resolution selects the method with signature \( s \).

\[
[\text{super}.m(E_1, \ldots, E_n)] = \langle \begin{array}{ll}
    v_1 & \leftarrow [E_1]; \ \cdots ; \ v_n & \leftarrow [E_n]; \ S::s(\text{context done throw (this.super) } v_1 \ \cdots \ v_n)
\end{array} \rangle
\]

3.12 Typecast Expressions

Let expression \( E \) be of type \( T_1 \).

If \( T_2 \) is a superclass or a superinterface of \( T_1 \) (that is, if the typecast is a widening conversion, i.e. an upcast):

\[
[(T_2) \ E] = \langle r & \leftarrow [E]; \ \text{case } r \ of \ \begin{cases}
    \text{Null} & \Rightarrow \ \
    \text{done}(\text{Null}) \\
    \text{Object } o_1 & \Rightarrow \ (o_2 & \leftarrow T_1::\uparrow T_2(o_1); \ \text{done}(\text{Object } o_2))
\end{cases} \rangle
\]

Otherwise:

\[
[E \ \text{as } T_2] = \langle r & \leftarrow [E]; \ \text{case } r \ of \ \begin{cases}
    \text{Null} & \Rightarrow \ \
    \text{done}(\text{Null}) \\
    \text{Object } o & \Rightarrow \ o\text{-dynamic-cast}([T_2] \ \text{done})
\end{cases} \rangle
\]

Here, \([T]\) is some encoding of the type name \( T \) in the datatype nat. For example, one could take the UTF-8 encoding of the type name and interpret it as a natural number.

Downcast and instanceof expressions can be translated using as expressions.
3.13 Class Instance Creation

\[ \text{new } C \] = \langle o \leftarrow C::\text{new}(); \text{done}(\text{Object } o) \rangle

4 Statements

data completion = Done \mid \text{Return} \mid \text{Throw} \mid \text{Break}

\text{TryFinally}(B, G) = \text{new start} \langle \text{new done return throw break } P_1 | P_2 \rangle

with

\[ P_1 = B | \text{done.start}(\text{Done}) | \text{return}(v).\text{start}(\text{Return } v) | \text{throw}(v).\text{start}(\text{Throw } v) | \text{break}(v).\text{start}(\text{Break } v) \]

\[ P_2 = \text{start}(c).(G; \text{case } c \text{ of} \begin{cases} \text{Done } & \Rightarrow \text{done} \\ \text{Return } v & \Rightarrow \text{return}(v) \\ \text{Throw } v & \Rightarrow \text{throw}(v) \\ \text{Break } v & \Rightarrow \text{break}(v) \end{cases} \]

4.1 Expression Statement

\[ [E;] = \langle v \leftarrow [E]; \text{done} \rangle \]

4.2 Return Statement

\[ [\text{return } E;] = \langle v \leftarrow [E]; \text{return}(v) \rangle \]

4.3 Sequential Composition Statement

\[ [S_1; S_2;] = ([S_1]; [S_2]) \]

4.4 Local Variable Declaration

\[ [[T \ n = E; S]] = \langle v \leftarrow [E]; \text{new } n(\pi(v) | [S]) \rangle \]

4.5 Empty Statement

\[ [:] = \text{done} \]
4.6 Labelled Statement

\[ \llbracket l : S \rrbracket = \text{new start (} \text{new break (} \llbracket S \rrbracket | \text{break(v).start(v)} \text{) | P}) \]

with

\[ P = \text{start(v).case v of } \begin{cases} \text{Zero} & \Rightarrow \text{done} \\ \text{Succ w} & \Rightarrow \text{break(w)} \end{cases} \]

4.7 If Statement

\[ \llbracket \text{if (} E \text{) } S_1 \text{ else } S_2 \rrbracket = (b \leftarrow [E]; (b?[S_1] : [S_2])) \]

4.8 While Statement

\[ \llbracket \text{while (} E \text{) } S \rrbracket = \text{new l (} \llbracket l \rrbracket (b \leftarrow [E]; (b?[S]; \text{done}) | \text{done}) | \text{done})) \]

4.9 Break Statement

\[ \llbracket \text{break } l \rrbracket = \text{break([}l]) \]

where \([l]\) is a natural number, encoded in datatype \text{nat}, equal to the number of enclosing labelled statements to be skipped. That is, the labelled statement with label \(l\) is the \(n + 1\)-th most enclosing labelled statement of this break statement.

A Java program can be rewritten, preserving semantics, into one without continue statements or unlabelled break statements.

4.10 Throw Statement

\[ \llbracket \text{throw } E \rrbracket = (r \leftarrow [E]; \text{NotNull(throw(o))}) \]

4.11 Try-Catch Statement

\[ \llbracket \text{try } S_1 \text{ catch (} T \text{ n) } S_2 \rrbracket = \text{new start (} \text{new throw (} \llbracket S_1 \rrbracket | \text{throw(o).start(o)} \text{) | P}) \]

where

\[ P = \text{start(o).} (r \leftarrow o.\text{dynamic-cast([}T]); \text{case } r \text{ of } \begin{cases} \text{Null} & \Rightarrow \text{throw(o)} \\ \text{Object o} & \Rightarrow \text{new n(} \pi(r) | [S_2]) \end{cases}) \]

4.12 Try-Finally Statement

\[ \llbracket \text{try } S_1 \text{ finally } S_2 \rrbracket = \text{TryFinally([}S_1],[S_2])} \]
4.13 Synchronized Statement

\[
\text{[synchronized } (E) \ S] = (r \leftarrow [E]; \text{NotNull}(P_1))
\]

where

\[
P_1 = (x \leftarrow [o.\text{lock-owner}]; \text{case } x \text{ of } \{
\begin{align*}
\text{Nothing} & \Rightarrow P_2 \\
\text{Just } n & \Rightarrow (n = \text{context} . \text{context-identity} ? [S] : P_2)
\end{align*}
\})
\]

\[
P_2 = o . \text{lock} . ([o.\text{lock-owner}] \leftarrow \text{Just context} . \text{context-identity}; \leftarrow \text{context} . \text{lock-locals}(); \text{TryFinally}([S], G))
\]

\[
G = (\leftarrow \text{context} . \text{unlock-locals}(); [o.\text{lock-owner}] \leftarrow \text{Nothing}; o.\text{lock} . \text{done})
\]

4.14 Asynchronous Statement

This statement, which does not exist in Java, can be used to implement `Thread.start()`. The statement starts a new thread to execute its body and completes normally without waiting for the new thread to finish. No jump labels are in scope in \( S \). Local variables are in scope only if they are final. `this` is in scope. It is a compile-time error for a return statement to appear in \( S \).

\[
\text{[async } S] = \text{NewContext}(\text{new done throw } [S]) | \text{ done}
\]

5 Programs

A Java program is a finite set of class and interface declarations:

\[
p = \{C_1, \ldots, C_n, I_1, \ldots, I_m\}
\]

\[
[p] = I | [C_1] | \cdots | [C_n] | [I_1] | \cdots | [I_m]
\]

\( I \) is the identity server.

\[
I = \text{new } !(!n) . \text{idty-server} (n) . \tilde{T} (\text{Succ } n) | \tilde{T} (\text{Zero})
\]

5.1 Interface Declarations

Let \( T \) be the name of interface \( I \); let \( I_1, \ldots, I_k \) be the superinterfaces of \( I \), including \( I \) itself. Let \( T_i \) be the name of interface \( I_i \).

\[
[I] = U_1 | \cdots | U_k
\]

Let \( s_1, \ldots, s_q \) be the signatures of the methods declared or inherited by \( I_i \).

\[
U_i =!T:: \uparrow T_i (o r) . \tau (\{a . \text{public-methods} \leftarrow \{s_1 \leftarrow a . \text{public-methods} . s_1, \ldots, s_q \leftarrow a . \text{public-methods} . s_q\}\})
\]
5.2 Class Declarations

Let $s_1, \ldots, s_q$ be the signatures of the methods declared or inherited by class $C$. Let $T$ be the name of $C$. Let $T_1, \ldots, T_k$ be the names of the superclasses and superinterfaces of $C$, including $C$ itself.

\[
[C] = N | N_\text{SP} | N_{\text{SI}} | M_1 | \cdots | M_q | U_1 | \cdots | U_k
\]

5.2.1 New Internal Super Server

**Root Class** If $C$ is the root class of the program (i.e. the one that does not extend a superclass):

Let $\ldots, f^*_i, \ldots$ be the names of the private instance fields declared in $C$. Let $\ldots, f^i_1, \ldots$ be the names of the internal instance fields declared in $C$. Let $\ldots, f^p_1, \ldots$ be the names of the public instance fields declared in $C$.

These names are used as labels and as $\pi$-calculus names, depending on the context.

Let $\vec{f} = f_1, \ldots, f_r$ be the names of all fields declared in $C$.

\[
N_{\text{SI}} = !T::\text{new-internal-super(p i r)}.\text{new o l v s d} \vec{f} \text{ identity-server}(n). (O \mid L \mid F)
\]

\[
O = \tau(\{\text{identity} \leftarrow n, \text{lock} \leftarrow l, \text{lock-owner} \leftarrow v, \text{private-fields} \leftarrow \{ \ldots, f^*_i \leftarrow f^*_i, \ldots \}, \text{internal-fields} \leftarrow \{ \ldots, f^i_1 \leftarrow f^i_1, \ldots \}, \text{public-fields} \leftarrow \{ \ldots, f^p_1 \leftarrow f^p_1, \ldots \}, \text{internal-methods} \leftarrow i, \text{public-methods} \leftarrow p, \text{super} \leftarrow s, \text{dynamic-cast} \leftarrow d\})
\]

\[
L = ([v] \leftarrow \text{Nothing}; \text{new w} (!w.l.l.w | w))
\]

\[
F = F_1 | \cdots | F_r
\]

Let $t_i$ be the type of field $f_i$. Let $(t_i)_0$ be the default value of type $t_i$.

\[
F_i = \text{identity-server}(n).\text{new m} (\text{m}(t_i)_0 | !\text{f}(n\text{m}))
\]

**Non-Root Class, Superclass in Same Package** Let $S$ be the name of the immediate superclass.

Let $\ldots, s^p_{i,1}, \ldots$ be the signatures of the public instance methods declared or inherited by $S$. Let $\ldots, s^{i,i}_{i,1}, \ldots$ be the signatures of the internal instance methods declared or inherited by $S$ which are also signatures of internal instance methods declared or inherited by $C$. Let $\ldots, s^{i,i+p}_{i,1}, \ldots$ be the signatures of the internal instance methods declared or inherited by $S$ which are overridden by public methods declared in $C$.
\[ N_{SI} = !C::\texttt{new-internal-super}(p i r). \]

\[ (s \leftarrow S::\texttt{new-internal-super}\{\ldots, s_{i}^{s,p} \leftarrow p.s_{i}^{s,p}, \ldots\}; \{\ldots, s_{i}^{s,i,i} \leftarrow i.s_{i}^{s,i,i}, \ldots, s_{i}^{s,i,p} \leftarrow p.s_{i}^{s,i,p}, \ldots\}; P) \]

Let \( \vec{f} = f_1, \ldots, f_r \) be the instance fields declared by \( C \).

\[ P = \texttt{new} \vec{f}(O \mid F) \]

Let \( \ldots, f_{i}^{s}, \ldots \) be the private fields declared by \( C \). Let \( \ldots, f_{i}^{i,d}, \ldots \) be the internal fields declared by \( C \). Let \( \ldots, f_{i}^{i,i}, \ldots \) be the internal fields inherited by \( C \). Let \( \ldots, f_{i}^{p,i}, \ldots \) be the public fields declared by \( C \). Let \( \ldots, f_{i}^{p,d}, \ldots \) be the public fields inherited by \( C \).

\[ O = \pi\{\text{identity} \leftarrow s.\text{identity}, \]
\[ \text{lock} \leftarrow s.\text{lock}, \]
\[ \text{lock-owner} \leftarrow s.\text{lock-owner}, \]
\[ \text{private-fields} \leftarrow \{\ldots, f_{i}^{s} \leftarrow f_{i}^{s}, \ldots\}, \]
\[ \text{internal-fields} \leftarrow \{\ldots, f_{i}^{i,d} \leftarrow f_{i}^{i,d}, \ldots, f_{i}^{i,i} \leftarrow s.\text{internal-fields}.f_{i}^{i,i}, \ldots\}, \]
\[ \text{public-fields} \leftarrow \{\ldots, f_{i}^{p,d} \leftarrow f_{i}^{p,d}, \ldots, f_{i}^{p,i} \leftarrow s.\text{public-fields}.f_{i}^{p,i}, \ldots\}, \]
\[ \text{internal-methods} \leftarrow i, \]
\[ \text{public-methods} \leftarrow p, \]
\[ \text{super} \leftarrow s, \]
\[ \text{dynamic-cast} \leftarrow s.\text{dynamic-cast}\} \]

\[ F = F_1 \mid \cdots \mid F_r \]

Let \( t_i \) be the type of field \( f_i \). Let \((t_i)_0\) be the default value of type \( t_i \).

\[ F_i = \texttt{identity-server}(n).\texttt{new m}(\pi(t_i)_0 \mid \mathcal{F}_i(n.m)) \]

**Non-Root Class, Superclass in Different Package** Let \( S \) be the name of the immediate superclass.

Let \( \ldots, s_{i}^{s,p}, \ldots \) be the signatures of the public instance methods declared or inherited by \( S \).

\[ N_{SI} = !C::\texttt{new-internal-super}(p i r). \]

\[ (s \leftarrow S::\texttt{new-public-super}\{\ldots, s_{i}^{s,p} \leftarrow p.s_{i}^{s,p}, \ldots\}; P) \]

Let \( \vec{f} = f_1, \ldots, f_r \) be the fields declared by \( C \).
\[ P = \text{new } \vec{f}(O \mid F) \]

Let \(\ldots, f^*_1, \ldots\) be the private fields declared by \(C\). Let \(\ldots, f^i_1, \ldots\) be the internal fields declared by \(C\). Let \(\ldots, f^{p,d}_1, \ldots\) be the public fields declared by \(C\). Let \(\ldots, f^{p,i}_1, \ldots\) be the public fields inherited by \(C\).

\[
O = \tau(\{ \text{identity} \leftarrow s.\text{identity}, \\
\text{lock} \leftarrow s.\text{lock}, \\
\text{lock-owner} \leftarrow s.\text{lock-owner}, \\
\text{private-fields} \leftarrow \{ \ldots, f^*_1 \leftarrow f^*_1, \ldots \}, \\
\text{internal-fields} \leftarrow \{ \ldots, f^i_1 \leftarrow f^i_1, \ldots \}, \\
\text{public-fields} \leftarrow \{ \ldots, f^{p,d}_1 \leftarrow f^{p,d}_1, \ldots, f^{p,i}_1 \leftarrow s.\text{public-fields}.f^{p,i}_1, \ldots \}, \\
\text{internal-methods} \leftarrow i, \\
\text{public-methods} \leftarrow p, \\
\text{super} \leftarrow s, \\
\text{dynamic-cast} \leftarrow s.\text{dynamic-cast} \})
\]

\[ F = F_1 \mid \cdots \mid F_r \]

Let \(t_i\) be the type of field \(f_i\). Let \((t_i)_0\) be the default value of type \(t_i\).

\[ F_i = \text{identity-server}(n).\text{new } m(\overline{t}_i)_0 \mid !F_i(n,m) \]

### 5.2.2 New Public Super Server

Let \(\ldots, s^i_1, \ldots\) be the signatures of the internal instance methods declared or inherited by \(C\).

\[
N_{SP} = !T::\text{new-public-super}(pr).
\]

\[
\begin{array}{l}
\text{new } \ldots \; s^i_1 \; \ldots \; (o \leftarrow C::\text{new-internal-super}(p \{ \ldots, s^i_1 \leftarrow s^i_1, \ldots \}); P)
\end{array}
\]

\[ P = \tau(o).(P_1 \mid \cdots \mid P_r) \]

Let \(s^i_j = m(t_1, \ldots, t_n)\).

\[ P_i = !s^i_j(c r t a_1 \cdots a_n).C::\overline{s}^i_j(c r t o a_1 \cdots a_n) \]

13
5.2.3 New Object Server

Let \( \ldots, s_i^p, \ldots = s_1^p, \ldots, s_r^p \) be the signatures of the public instance methods declared or inherited by \( C \).

\[
N = !T::\text{new}(r).\text{new} \cdots s_i^p \cdots (o \leftarrow C::\text{new-public-super}(\{\ldots, s_i^p \leftarrow s_i^p, \ldots\})); P
\]

\[P = \tau(o).(D | P_1 | \cdots | P_r)\]

\[D = !o.\text{dynamic-cast}(n \, r).D_0\]

Let \( m \) be a natural number that is greater than all encodings of names of superclasses or superinterfaces of \( C \).

\[D_m = \tau(\text{Null})\]

For each \( i < m \):

\[D_i = \text{case } n \text{ of } \{ \text{Zero } \Rightarrow D'_i \, \text{Succ } n \Rightarrow D_{i+1} \}\]

If \( i \) is the encoding of a superclass or superinterface type name \( T' \):

\[D'_i = (o \leftarrow T'; T'(o); \tau(\text{Object } o))\]

Otherwise:

\[D'_i = \tau(\text{Null})\]

Let \( s_i^p = m(t_1, \ldots, t_n) \).

\[P_1 = !s_i^p(c \, r \, t \, a_1 \cdots a_n).T::s_i^p(c \, r \, t \, o \, a_1 \cdots a_n)\]

5.2.4 Method Servers

If the method declared or inherited by \( C \) with signature \( s_i \) is static, then let

\[\vec{a} = \text{context, return, throw, } a_1, \ldots, a_n\]

Otherwise:

\[\vec{a} = \text{context, return, throw, this, } a_1, \ldots, a_n\]
Declared Methods Suppose $C$ declares the method.

If this method is declared abstract or native, then $M_i = 0$. Otherwise, let $p_1, \ldots, p_n$ be the parameter names and let $S$ be the statement that constitutes the method body.

$$M_i = !T::s_i(\vec{a}).new p_1 \cdots p_n (\vec{a_1}) | \cdots | \vec{a_n} | new break ([S]; return)$$

Inherited Methods Suppose $C$ inherits the method from its immediate superclass $S$.

Let $\vec{b}$ be the result of substitution of $this.super$ for $this$ in $\vec{a}$.

$$M_i = !T::s_i(\vec{a}).S::s_i(\vec{b})$$

5.2.5 Upcast Servers

If $T_i$ is $C$:

$$U_i = !T:: \uparrow T_i(o r) \tau(\vec{a})$$

If $T_i$ is a superinterface of $C$:

Let $s'_1, \ldots, s'_q$ be the signatures of the methods declared or inherited by $T_i$.

$$U_i = !T:: \uparrow T_i(o r) \tau\{o, public-methods \leftarrow \{s'_1 \leftarrow o.public-methods, s'_1, \ldots, s'_q \leftarrow o.public-methods.s'_q}\}\}$$

If $T_i$ is a superclass of $C$:

Let $S$ be the immediate superclass of $C$.

$$U_i = !T:: \uparrow T_i(o r) S:: \uparrow T_i(o . super)$$

6 Differences with the JLS

6.1 Unsupported Features

6.2 Changed Features

- Memory model

- Suppose an internal method $m_D$ in class $D$ overrides a method $m_C$ in class $C$, according to the JLS. Suppose also that some class $S$ that is a superclass of $D$ and a subclass of $C$ is in a different package. Then $m_D$ effectively does not override $m_C$ in our semantics. That is, an invocation of $m_C$ on a target of static type $C$ is never bound to $m_D$. 

6.3 Notes on the JLS

Suppose classes $C$ and $D$ are in the same package, and $D$ is a subclass of $C$. Suppose also that there is a class $S$ in a different package that is a subclass of $C$ and a superclass of $D$. Then $D$ never inherits the internal fields of $C$. However, it is easy to access these fields using an explicit upcast.