Primitive types and literals of Java$_{\mathcal{I}}$

Type	Size	Default	Literals
`boolean`		`false`	`true`, `false`
`byte`	8 bit	`(byte)0`	
`short`	16 bit	`(short)0`	
`int`	32 bit	`0`	`11`, `1969`, `0xff00`, `017`
`long`	64 bit	`0L`	`11L`, `0x1000L`, `0777L`
`float`	32 bit	`0.0f`	`3.141f`, `1.2e+23f`
`double`	64 bit	`0.0d`	`3.141`, `1e-9`, `0.1e10`
`char`	16 bit	`'\u0000'`	`'a'`, `'?'`,`'\n'`, `'\uFFFF'`

Subtype relation of Java$_{\mathcal{I}}$

$\texttt{byte} \preceq \texttt{short} \preceq \texttt{int} \preceq \texttt{long} \preceq \texttt{float} \preceq \texttt{double}$

$\texttt{char} \preceq \texttt{int}$

$A \preceq A$

$A \preceq B$ and $B \preceq C \implies A \preceq C$

Syntactic categories of Java$_{\mathcal{I}}$

Exp . . . . . . expressions,
Asgn . . . . . assignments,
Stm . . . . . statements,
Block . . . . blocks,
Bstm . . . . block statements,
Lit . . . . literals,
Loc . . . . . . local variables,
Uop . . . . . unary operators,
Bop . . . . . binary operators,
Lab . . . . . . labels.

Syntax of $\text{Java}_{\mathcal{I}}$

$$
\begin{aligned}
\mathit{Exp} &:= \mathit{Lit} \mid \mathit{Loc} \mid \mathit{Uop}\,\mathit{Exp} \mid \mathit{Exp}\,\mathit{Bop}\,\mathit{Exp} \\
&\mid \mathit{Exp}\,?\,\mathit{Exp} : \mathit{Exp} \mid \mathit{Asgn} \\
\mathit{Asgn} &:= \mathit{Loc} = \mathit{Exp} \\
\mathit{Stm} &:= ; \mid \mathit{Asgn}; \mid \mathit{Lab} : \mathit{Stm} \mid \texttt{break}\ \mathit{Lab}; \mid \texttt{continue}\ \mathit{Lab}; \\
&\mid \texttt{if}\ (\mathit{Exp})\ \mathit{Stm}\ \texttt{else}\ \mathit{Stm} \mid \texttt{while}\ (\mathit{Exp})\ \mathit{Stm} \mid \mathit{Block} \\
\mathit{Block} &:= \{\mathit{Bstm}_1 \ldots \mathit{Bstm}_n\} \\
\mathit{Bstm} &:= \mathit{Type}\,\mathit{Loc}; \mid \mathit{Stm} \\
\mathit{Phrase} &:= \mathit{Exp} \mid \mathit{Bstm} \mid \mathit{Val} \mid \mathit{Abr} \mid \mathit{Norm}
\end{aligned}
$$

Unary Operators

Prec.	*Uop*	Operand type	Result type	Operation
1	`+`	A numeric	$\max(A, \texttt{int})$	unary plus
1	`-`	A numeric	$\max(A, \texttt{int})$	unary minus
1	`~`	A integral	$\max(A, \texttt{int})$	bitwise NOT
1	`!`	`boolean`	`boolean`	logical complement
1	(B)	$A \neq \texttt{boolean}$	$B \neq \texttt{boolean}$	type cast

Binary Operators

Prec.	*Bop*	Operand types	Result type	Operation
2	`*`	A and B numeric	$\max(A, B, \texttt{int})$	multiplication
2	`/`	A and B numeric	$\max(A, B, \texttt{int})$	division
2	`%`	A and B numeric	$\max(A, B, \texttt{int})$	remainder
3	`+`	A and B numeric	$\max(A, B, \texttt{int})$	addition
3	`-`	A and B numeric	$\max(A, B, \texttt{int})$	subtraction
4	`<<`	A and B integral	$\max(A, \texttt{int})$	left shift
4	`>>`	A and B integral	$\max(A, \texttt{int})$	signed right shift
4	`>>>`	A and B integral	$\max(A, \texttt{int})$	unsigned right shift
5	`<`	A and B numeric	`boolean`	less than
5	`<=`	A and B numeric	`boolean`	less than or equal

Binary Operators (continued)

Prec.	*Bop*	Operand types	Result type	Operation
5	`>`	A and B numeric	`boolean`	greater than
5	`>=`	A and B numeric	`boolean`	greater than or equal
6	`==`	$A \preceq B$ or $B \preceq A$	`boolean`	equal
6	`!=`	$A \preceq B$ or $B \preceq A$	`boolean`	not equal
7	`&`	A and B integral	$\max(A, B, \texttt{int})$	bitwise AND
7	`&`	$A = B = \texttt{boolean}$	`boolean`	boolean AND
8	`^`	A and B integral	$\max(A, B, \texttt{int})$	bitwise XOR
8	`^`	$A = B = \texttt{boolean}$	`boolean`	boolean XOR
9	`|`	A and B integral	$\max(A, B, \texttt{int})$	bitwise OR
9	`|`	$A = B = \texttt{boolean}$	`boolean`	boolean OR

Typing conditions for expressions of Java$_{\mathcal{I}}$

$^{\alpha}lit$	$\mathcal{T}(\alpha)$ is the type of lit according to the JLS.
$^{\alpha}loc$	$\mathcal{T}(\alpha)$ is the declared type of loc.
$^{\alpha}(uop\,^{\beta}e)$	The result of applying uop to an operand of type $\mathcal{T}(\beta)$ is of type $\mathcal{T}(\alpha)$.
$^{\alpha}(^{\beta}e_1\ bop\ ^{\gamma}e_2)$	The result of applying bop to operands of type $\mathcal{T}(\beta)$ and $\mathcal{T}(\gamma)$ is of type $\mathcal{T}(\alpha)$.
$^{\alpha}(loc = {}^{\beta}e)$	$\mathcal{T}(\alpha)$ is the declared type of loc and $\mathcal{T}(\beta) \preceq \mathcal{T}(\alpha)$.
$^{\alpha}(^{\beta}e_0\,?\,^{\gamma}e_1 : {}^{\delta}e_2)$	Let $A = \mathcal{T}(\gamma)$ and $B = \mathcal{T}(\delta)$. Then $\mathcal{T}(\beta)$ is `boolean` and one of the following conditions is true: ■ A, B are numeric and $\mathcal{T}(\alpha) = \max(A, B, \texttt{int})$ ■ $A \preceq B$ and $\mathcal{T}(\alpha) = B$ ■ $B \preceq A$ and $\mathcal{T}(\alpha) = A$

Type constraints after introduction of primitive type casts

$^{\alpha}(loc = {}^{\beta}e)$	Let A be the declared type of loc. If A is a primitive type, then $\mathcal{T}(\beta) = A = \mathcal{T}(\alpha)$.
$^{\alpha}({}^{\beta}e_0\, ?\, {}^{\gamma}e_1 : {}^{\delta}e_2)$	If $\mathcal{T}(\gamma)$ and $\mathcal{T}(\delta)$ are primitive types, then $\mathcal{T}(\gamma) = \mathcal{T}(\delta) = \mathcal{T}(\alpha)$.

Vocabulary of the ASM for Java$_{\mathcal{I}}$

Universes:

Pos . . . . positions

Val . . . . values (*Type*, *BitString*)

Phrase . . . . semi-evaluated syntax trees

Abr . . . . reasons for abruptions: $Break($*Lab*$) \mid Continue($*Lab*$)$

Special constants:

$True$, $False$, $Norm$, $firstPos$

Static functions and constants:

up: *Pos* $\longrightarrow$ *Pos*

$body$: *Block*

Dynamic functions and constants:

pos: *Pos*

$restbody$: *Phrase*

$locals$: *Loc* $\longrightarrow$ *Val*

Transition rules for Java$_\mathcal{I}$

Initial state of Java$_\mathcal{I}$:

$$\begin{aligned} pos &= firstPos \\ restbody &= body \\ locals &= \emptyset \end{aligned}$$

Main transition rule for Java$_\mathcal{I}$:

$$execJava_I = \\ \quad execJavaExp_I \\ \quad execJavaStm_I$$

Rule macros:

$$yield(result) = \\ \quad restbody := restbody[result/pos]$$

$$yieldUp(result) = \\ \quad \begin{aligned} restbody &:= restbody[result/up(pos)] \\ pos &:= up(pos) \end{aligned}$$

Transition rules for Java$_{\mathcal{I}}$ (continued)

Derived function:

$$
\begin{aligned}
context(pos) = \;& \textbf{if } pos = \mathit{firstPos} \vee restbody/pos \in \mathsf{Bstm} \cup \mathsf{Exp} \\
& \textbf{then} \\
& \quad restbody/pos \\
& \textbf{else} \\
& \quad restbody/up(pos)
\end{aligned}
$$

Derived predicate:

$$
\begin{aligned}
& propagatesAbr(phrase) = \\
& \quad phrase \neq lab : s
\end{aligned}
$$

Transition rules for Java$_{\mathcal{I}}$ (Expressions)

$$
\begin{array}{l}
\mathit{execJavaExp}_I = \textbf{case } \mathit{context}(\mathit{pos}) \textbf{ of} \\
\quad \mathit{lit} \to \mathit{yield}(\mathit{JLS}(\mathit{lit})) \\
\\
\quad \mathit{loc} \to \mathit{yield}(\mathit{locals}(\mathit{loc})) \\
\\
\quad \mathit{uop}\, {}^{\alpha}\mathit{exp} \to \mathit{pos} := \alpha \\
\quad \mathit{uop}\, {}^{\blacktriangleright}\mathit{val} \to \mathit{yieldUp}(\mathit{JLS}(\mathit{uop}, \mathit{val})) \\
\\
\quad {}^{\alpha}\mathit{exp}_1\ \mathit{bop}\ {}^{\beta}\mathit{exp}_2 \to \mathit{pos} := \alpha \\
\quad {}^{\blacktriangleright}\mathit{val}\ \mathit{bop}\ {}^{\beta}\mathit{exp} \quad \to \mathit{pos} := \beta \\
\quad {}^{\alpha}\mathit{val}_1\ \mathit{bop}\ {}^{\blacktriangleright}\mathit{val}_2 \ \to \textbf{if } \neg(\mathit{bop} \in \mathit{divMod} \wedge \mathit{isZero}(\mathit{val}_2)) \textbf{ then} \\
\qquad\qquad\qquad\qquad \mathit{yieldUp}(\mathit{JLS}(\mathit{bop}, \mathit{val}_1, \mathit{val}_2)) \\
\\
\quad \mathit{loc} = {}^{\alpha}\mathit{exp} \to \mathit{pos} := \alpha \\
\quad \mathit{loc} = {}^{\blacktriangleright}\mathit{val} \to \mathit{locals} := \mathit{locals} \oplus \{(\mathit{loc}, \mathit{val})\} \\
\qquad\qquad\qquad\quad \mathit{yieldUp}(\mathit{val}) \\
\\
\quad {}^{\alpha}\mathit{exp}_0\, ?\, {}^{\beta}\mathit{exp}_1 \ :\ {}^{\gamma}\mathit{exp}_2 \to \mathit{pos} := \alpha \\
\quad {}^{\blacktriangleright}\mathit{val}\, ?\, {}^{\beta}\mathit{exp}_1 \ :\ {}^{\gamma}\mathit{exp}_2 \ \to \textbf{if } \mathit{val} \textbf{ then } \mathit{pos} := \beta \textbf{ else } \mathit{pos} := \gamma \\
\quad {}^{\alpha}\mathit{True}\, ?\, {}^{\blacktriangleright}\mathit{val} \ :\ {}^{\gamma}\mathit{exp} \ \to \mathit{yieldUp}(\mathit{val}) \\
\quad {}^{\alpha}\mathit{False}\, ?\, {}^{\beta}\mathit{exp} \ :\ {}^{\blacktriangleright}\mathit{val} \to \mathit{yieldUp}(\mathit{val})
\end{array}
$$

Transition rules for Java$_\mathcal{I}$ (Statements)

$$
\begin{aligned}
&execJavaStm_I = \textbf{case } context(pos) \textbf{ of}\\
&\quad ; \quad\quad \to yield(Norm)\\
&\quad {}^{\alpha}exp; \to pos := \alpha\\
&\quad {}^{\blacktriangleright}val; \to yieldUp(Norm)
\end{aligned}
$$

$$
\begin{aligned}
&\quad \texttt{break } lab; && \to yield(Break(lab))\\
&\quad \texttt{continue } lab; && \to yield(Continue(lab))\\
&\quad lab : {}^{\alpha}stm && \to pos := \alpha\\
&\quad lab : {}^{\blacktriangleright}Norm && \to yieldUp(Norm)\\
&\quad lab : {}^{\blacktriangleright}Break(lab_b) && \to \textbf{if } lab = lab_b \textbf{ then } yieldUp(Norm)\\
& && \quad\ \textbf{else } yieldUp(Break(lab_b))\\
&\quad lab : {}^{\blacktriangleright}Continue(lab_c) && \to \textbf{if } lab = lab_c \textbf{ then } yield(body/pos)\\
& && \quad\ \textbf{else } yieldUp(Continue(lab_c))
\end{aligned}
$$

$$
\begin{aligned}
&\quad phrase({}^{\blacktriangleright}abr) \to \textbf{if } pos \neq firstPos \wedge propagatesAbr(restbody/up(pos)) \textbf{ then}\\
&\quad\quad\quad\quad\quad\quad\quad\quad\quad yieldUp(abr)
\end{aligned}
$$

$$
\begin{aligned}
&\quad \{\,\} && \to yield(Norm)\\
&\quad \{{}^{\alpha_1}stm_1 \ldots {}^{\alpha_n}stm_n\} && \to pos := \alpha_1\\
&\quad \{{}^{\alpha_1}Norm \ldots {}^{\blacktriangleright}Norm\} && \to yieldUp(Norm)\\
&\quad \{{}^{\alpha_1}Norm \ldots {}^{\blacktriangleright}Norm^{\alpha_{i+1}}stm_{i+1} \ldots {}^{\alpha_n}stm_n\} && \to pos := \alpha_{i+1}
\end{aligned}
$$

Transition rules for Java$_\mathcal{I}$ (Statements continued)

$$\begin{array}{ll}
\texttt{if}\ ({}^\alpha exp)\ {}^\beta stm_1\ \texttt{else}\ {}^\gamma stm_2 & \to pos := \alpha \\
\texttt{if}\ ({}^\blacktriangleright val)\ {}^\beta stm_1\ \texttt{else}\ {}^\gamma stm_2 & \to \textbf{if}\ val\ \textbf{then}\ pos := \beta\ \textbf{else}\ pos := \gamma \\
\texttt{if}\ ({}^\alpha True)\ {}^\blacktriangleright Norm\ \texttt{else}\ {}^\gamma stm & \to yieldUp(Norm) \\
\texttt{if}\ ({}^\alpha False)\ {}^\beta stm\ \texttt{else}\ {}^\blacktriangleright Norm & \to yieldUp(Norm)
\end{array}$$

$$\begin{array}{ll}
\texttt{while}\ ({}^\alpha exp)\ {}^\beta stm & \to pos := \alpha \\
\texttt{while}\ ({}^\blacktriangleright val)\ {}^\beta stm & \to \textbf{if}\ val\ \textbf{then}\ pos := \beta\ \textbf{else}\ yieldUp(Norm) \\
\texttt{while}\ ({}^\alpha True)\ {}^\blacktriangleright Norm & \to yieldUp(body/up(pos))
\end{array}$$

$$Type\ x; \to yield(Norm)$$

Derived language constructs in Java$_{\mathcal{I}}$

Derived	Java$_{\mathcal{I}}$
exp_1 `&&` exp_2	exp_1 ? exp_2 : `false`
exp_1 `||` exp_2	exp_1 ? `true` : exp_2
`++`loc	$loc = (A)(loc + 1)$, where loc has type A
`--`loc	$loc = (A)(loc - 1)$, where loc has type A
`if` (exp) stm	`if` (exp) stm `else`;

The dangling 'else' problem:

`if` (exp_1) `if` (exp_2) stm_1 `else` stm_2

`if` (exp_1) $\{$`if` (exp_2) stm_1 `else` $stm_2\}$

`if` (exp_1) $\{$`if` (exp_2) $stm_1\}$ `else` stm_2