Statement Semantics

Statement = Skip | Block | Assignment | Conditional | Loop

\[ M : Statement \times State \rightarrow State \]

\[ M(\text{Statement } s, \text{State } state) = \begin{cases} 
M((\text{Skip})s, state), & \text{if } s \text{ is a Skip} \\
M((\text{Assignment})s, state), & \text{if } s \text{ is an Assignment} \\
M((\text{Conditional})s, state), & \text{if } s \text{ is a Conditional} \\
M((\text{Loop})s, state), & \text{if } s \text{ is a Loop} \\
M((\text{Block})s, state), & \text{if } s \text{ is a Block} 
\end{cases} \]
Skip Semantics

\[ \text{Skip} = \]

- The meaning of a Skip statement is an identity function on the state; that is, the state is unchanged.

\[ M : \text{Statement} \times \text{State} \rightarrow \text{State} \]

\[ M(\text{Skip } s, \text{State } state) = state \]
Overriding Union

The **overriding union** of \( X \) and \( Y \), written \( X \cup Y \), is the result of replacing in \( X \) all pairs \( \langle x, v \rangle \) whose first member matches a pair \( \langle x, w \rangle \) from \( Y \) by \( \langle x, w \rangle \) and then adding to \( X \) any remaining pairs in \( Y \).

**Example:**

\[
X = \{ \langle a, 1 \rangle, \langle b, 5 \rangle, \langle c, -1 \rangle \}
\]

\[
Y = \{ \langle b, 6 \rangle, \langle d, 0 \rangle \}
\]

\[
X \cup Y = \{ \langle a, 1 \rangle, \langle b, 6 \rangle, \langle c, -1 \rangle, \langle d, 0 \rangle \}
\]
Assignment Semantics

\[ Assignment = Variable \ target; Expression \ source \]

- The \textit{meaning of an Assignment statement} is the result of \textbf{replacing} the value of the \textit{target} Variable by the value of the \textit{source} Expression in the current state.

\[
M : Statement \times State \rightarrow State
\]

\[
M(Assignment \ a, State \ state) = state \uplus \{(a.target, M(a.source, state))\}
\]
Example: \( a = b + 3 \times c; \)

\[
state = \{ \langle a, 5 \rangle, \langle b, 1 \rangle, \langle c, 3 \rangle \}
\]

\[
M(Assignment \ a, State \ state) = state \cup \{ \langle a.\text{target}, M(a.\text{source}, state) \rangle \}
\]
$M(b + 3 \cdot c, \{(b, 1), (c, 3)\}) = \text{ApplyBinary}(+, A, B)$

$= \text{ApplyBinary}(+, 1, 9)$

$= 10$

$A = M(b, \{(b, 1), (c, 3)\})$

$= 1$

$B = M(3 \cdot c, \{(b, 1), (c, 3)\})$

$= \text{ApplyBinary}(\cdot, C, D)$

$= \text{ApplyBinary}(\cdot, 3, 3)$

$= 9$

$C = M(3, \{(b, 1), (c, 3)\})$

$= 3$

$D = M(c, \{(b, 1), (c, 3)\})$

$= 3$
Example: \( a = b + 3 \times c; \)

\[
\text{state} = \{ \langle a, 5 \rangle, \langle b, 1 \rangle, \langle c, 3 \rangle \} 
\]

\[
M(\text{Assignment } a, \text{State state}) = \text{state} \cup \{ \langle a.\text{target}, M(a.\text{source}, \text{state}) \rangle \}
\]

\[
M(a = b + 3 \times c, \{ \langle a, 5 \rangle, \langle b, 1 \rangle, \langle c, 3 \rangle \}) = \{ \langle a, 5 \rangle, \langle b, 1 \rangle, \langle c, 3 \rangle \} \cup \{ \langle a, 10 \rangle \} = \{ \langle a, 10 \rangle, \langle b, 1 \rangle, \langle c, 3 \rangle \}
\]
Example: \( z = 3 \times x + y/2 \)

\[
\text{state} = \{\langle x, 2 \rangle, \langle y, -3 \rangle, \langle z, 1 \rangle\}
\]

\[
M(\text{Assignment } a, \text{State } state) = \text{state} \cup \{\langle a.\text{target}, M(a.\text{source}, state)\rangle\}
\]

\[
M(\text{Expression } e, \text{State } state) = \begin{cases} 
 e, & \text{if } e \text{ is a Value} \\
 \text{state}(e), & \text{if } e \text{ is a Variable} \\
 \text{ApplyBinary}(e.\text{op}, M(e.\text{term1}, state), M(e.\text{term2}, state)), & \text{if } e \text{ is a Binary} \\
 \text{ApplyUnary}(e.\text{op}, M(e.\text{term}, state)), & \text{if } e \text{ is a Unary}
\end{cases}
\]

\[
\text{ApplyBinary}(\text{Operator } op, \text{Value } v1, \text{Value } v2) = \begin{cases} 
 v1 + v2, & \text{if } op = \text{int+} \\
 v1 - v2, & \text{if } op = \text{int-} \\
 \left\lfloor \frac{v1}{v2} \right\rfloor \times \text{sign}(v1 \times v2), & \text{if } op = \text{int/}
\end{cases}
\]
Assignment Semantics of C and Java

✧ Assignments vs. Expressions

- C treats assignments as expression, but Java does not

✧ Copy Semantics vs. Reference Semantics

- **Copy semantics**: a copy of the value of the source expression is assigned to the target variable

  \[ a = b \]

- **Reference semantics**: a reference to the value of the source expression is assigned to the target variable

  \[ a = b \]

- C uses copy semantics. Java uses copy semantics for primitive types, and reference semantics for type `Object`