1 Source Language

⟨*statement*⟩ ::= ⟨*compound-statement*⟩
| `if (` ⟨*expression*⟩ `)` ⟨*compound-statement*⟩ `else` ⟨*compound-statement*⟩
| `while (` ⟨*expression*⟩ `)` ⟨*compound-statement*⟩
| `for (` ⟨*statement*⟩ `;` ⟨*statement*⟩ `;` ⟨*statement*⟩ `)` ⟨*compound-statement*⟩
| ⟨*qualifier*⟩ ⟨*type*⟩ ⟨*name*⟩ `;`
| ⟨*qualifier*⟩ ⟨*type*⟩ ⟨*name*⟩ `=` ⟨*expression*⟩ `;`
| `return` ⟨*expression*⟩ `;`
| ⟨*expression*⟩ `;`

⟨*compound-statement*⟩ ::= `{` ⟨*statement*⟩* `}`

⟨*primary-expression*⟩ ::= ⟨*identifier*⟩
| ⟨*constant*⟩
| ⟨*string-literal*⟩
| `(` ⟨*expression*⟩ `)`

⟨*postfix-expression*⟩ ::= ⟨*primary-expression*⟩
| ⟨*postfix-expression*⟩ `[` ⟨*expression*⟩ `]`
| ⟨*name*⟩ `(` ⟨*argument-list*⟩ `)`
| ⟨*name*⟩ `++`
| ⟨*name*⟩ `--`

⟨*unary-expression*⟩ ::= ⟨*postfix-expression*⟩
| `++` ⟨*name*⟩
| `--` ⟨*name*⟩
| `&` ⟨*name*⟩
| ⟨*unary-operator*⟩ ⟨*cast-expression*⟩

⟨*multiplicative-expression*⟩ ::= ⟨*unary-expression*⟩
| ⟨*multiplicative-expression*⟩ `*` ⟨*unary-expression*⟩
| ⟨*multiplicative-expression*⟩ `/` ⟨*unary-expression*⟩
| ⟨*multiplicative-expression*⟩ `%` ⟨*unary-expression*⟩

⟨*additive-expression*⟩ ::= ⟨*multiplicative-expression*⟩
| ⟨*additive-expression*⟩ `+` ⟨*multiplicative-expression*⟩
| ⟨*additive-expression*⟩ `-` ⟨*multiplicative-expression*⟩

⟨*shift-expression*⟩ ::= ⟨*additive-expression*⟩
| ⟨*shift-expression*⟩ `<<` ⟨*additive-expression*⟩
| ⟨*shift-expression*⟩ `>>` ⟨*additive-expression*⟩

⟨*relational-expression*⟩ ::= ⟨*shift-expression*⟩
| ⟨*relational-expression*⟩ `<` ⟨*shift-expression*⟩
| ⟨*relational-expression*⟩ `>` ⟨*shift-expression*⟩
| ⟨*relational-expression*⟩ `<=` ⟨*shift-expression*⟩
| ⟨*relational-expression*⟩ `>=` ⟨*shift-expression*⟩

⟨*equality-expression*⟩ ::= ⟨*relational-expression*⟩
  | ⟨*equality-expression*⟩ `==` ⟨*relational-expression*⟩
  | ⟨*equality-expression*⟩ `!=` ⟨*relational-expression*⟩

⟨*bitwise-and-expression*⟩ ::= ⟨*equality-expression*⟩
  | ⟨*and-expression*⟩ `&` ⟨*equality-expression*⟩

⟨*exclusive-or-expression*⟩ ::= ⟨*and-expression*⟩
  | ⟨*exclusive-or-expression*⟩ `|` ⟨*exclusive-or-expression*⟩

⟨*bitwise-or-expression*⟩ ::= ⟨*exclusive-or-expression*⟩
  | ⟨*inclusive-or-expression*⟩ `|` ⟨*exclusive-or-expression*⟩

⟨*logical-and-expression*⟩ ::= ⟨*bitwise-or-expression*⟩
  | ⟨*logical-and-expression*⟩ `&&` ⟨*bitwise-or-expression*⟩

⟨*logical-or-expression*⟩ ::= ⟨*logical-and-expression*⟩
  | ⟨*logical-or-expression*⟩ `||` ⟨*logical-and-expression*⟩

⟨*conditional-expression*⟩ ::= ⟨*logical-or-expression*⟩
  | ⟨*conditional-expression*⟩ `?` ⟨*expression*⟩ `:` ⟨*conditional-expression*⟩

⟨*assignment-expression*⟩ ::= ⟨*conditional-expression*⟩
  | ⟨*unary-expression*⟩ ⟨*assignment-operator*⟩ ⟨*assignment-expression*⟩

⟨*expression*⟩ ::= ⟨*assignment-expression*⟩

⟨*parameter-list*⟩ ::= ⟨*qualifier*⟩ ⟨*type*⟩ ⟨*name*⟩ `,` ⟨*argument-list*⟩
  | ⟨*type-qualifier*⟩ ⟨*type*⟩ ⟨*name*⟩

⟨*argument-list*⟩ ::= ⟨*expression*⟩ `,` ⟨*argument-list*⟩
  | ⟨*expression*⟩

⟨*assignment-operator*⟩ ::= `=` | `*=` | `/=` | `%=` | `+=` | `-=`

⟨*unary-operator*⟩ ::= `*` | `-` | `!` | `~`

⟨*type-qualifier*⟩ ::= `const` | `volatile`

⟨*type-specifier*⟩ ::= `int` | `char` | `float`

⟨*type*⟩ ::= ⟨*type-qualifier*⟩ ⟨*type-specifier*⟩

⟨*function-definition*⟩ ::= ⟨*type*⟩ ⟨*name*⟩ `(` ⟨*parameter-list*⟩ `)` ⟨*compound-expression*⟩

2 Symbolic Execution

$S = \langle g; \rho; \mu \rangle$

2.1 Expressions

$$\frac{}{\langle S; v\rangle \Downarrow \langle S; v\rangle}\ \textsc{Literal}$$

$$\frac{\langle S; e\rangle \Downarrow S'; s\rangle}{\langle S; \texttt{-}e\rangle \Downarrow \langle S'; -s\rangle}\ \textsc{Negate}$$

$$\frac{\langle S; e_1\rangle \Downarrow S_1; s_1\rangle \qquad \langle S_1; e_2\rangle \Downarrow S_2; s_2\rangle}{\langle S; e_1\ \texttt{+}\ e_2\rangle \Downarrow \langle S_2; e_1 + e_2\rangle}\ \textsc{Add}$$

$$\frac{\langle S; e_1\ \texttt{=}\ e_1\ \texttt{+}\ e_2\rangle \Downarrow S'; s\rangle}{\langle S; e_1\ \texttt{+=}\ e_2\rangle \Downarrow \langle S'; s\rangle}\ \textsc{AssignAdd}$$

$$\frac{\forall i \in 1..n, \langle S_i; e_i\rangle \Downarrow \langle S_{i+1}; s_i\rangle}{\langle S_1; \texttt{x(}e_1\texttt{,..,}e_n\texttt{)}\rangle \Downarrow \langle S_{n+1}; x(s_1, .., s_n)\rangle}\ \textsc{FunCall}$$

2.2 Statements

$$\frac{\langle S; e\rangle \Downarrow \langle S'; s\rangle}{\langle S; e\texttt{;}\rangle \Downarrow \langle S'; \emptyset\rangle}\ \textsc{Expression}$$

$$\frac{\forall i \in 1..n, \langle S_i; c_i\rangle \Downarrow \langle S_{i+1}; s_{i+1}\rangle}{\langle S_1; \{c_1..c_n\}\rangle \Downarrow \langle S_{n+1}; s_{n+1}\rangle}\ \textsc{CompoundStatement}$$

$$\frac{\begin{array}{c}\langle S; e\rangle \Downarrow \langle S_1; g_1\rangle \qquad g(S) \not\Longrightarrow g_1 \qquad g(S) \not\Longrightarrow \neg g_1 \\ \langle S_1[g \mapsto g(S_1) \wedge g_1]; c_1\rangle \Downarrow \langle S_2; s_2\rangle \\ \langle S_1[g \mapsto g(S_1) \wedge \neg g_1]; c_1\rangle \Downarrow \langle S_3; s_3\rangle \\ S' = \langle (g_1 ? g(S_2) {:} g(S_3)); (g_1 ? \rho(S_2) {:} \rho(S_3)); (g_1 ? \mu(S_2) {:} \mu(S_3))\end{array}}{\langle S; \texttt{if}\ e\ c_1\ \texttt{else}\ c_2\rangle \Downarrow \langle S'; \emptyset\rangle}\ \textsc{IfElse}$$

$$\frac{\langle S; e\rangle \Downarrow \langle S_1; g_1\rangle \qquad g(S) \Longrightarrow g_1 \qquad \langle S_1; c_1\rangle \Downarrow \langle S_2; s\rangle}{\langle S; \texttt{if}\ e\ c_1\ \texttt{else}\ c_2\ \rangle \Downarrow \langle S_2; \emptyset\rangle}\ \textsc{IfTrue}$$

$$\frac{\langle S; e\rangle \Downarrow \langle S_1; g_1\rangle \qquad g(S) \Longrightarrow \neg g_1 \qquad \langle S_1; c_2\rangle \Downarrow \langle S_2; s\rangle}{\langle S; \texttt{if}\ e\ c_1\ \texttt{else}\ c_2\rangle \Downarrow \langle S_2; \emptyset\rangle}\ \textsc{IfFalse}$$

2.3 Memory

$$\frac{\rho(S)[x] = s}{\langle S; x\rangle \Downarrow \langle S; s\rangle}\ \textsc{Var}$$

$$\frac{x \notin \text{dom } \rho(S)}{\langle S; \tau\ \ x\texttt{;}\rangle \Downarrow \langle S[\rho \mapsto (\rho(S), (x \to \emptyset)]; s\rangle}\ \textsc{DeclareLocal}$$

$$\frac{x \notin \text{dom } \rho(S) \qquad \langle S; e\rangle \Downarrow \langle S'; s\rangle}{\langle S; \tau\ \ x\ \texttt{=}\ e\texttt{;}\rangle \Downarrow \langle S'[\rho \mapsto (\rho(S'), (x \to s)]; s\rangle}\ \textsc{DeclareAssignLocal}$$

$$\frac{\langle S; e_1\rangle \Downarrow \langle S_1; \text{ptr } x\rangle \qquad x \in \text{dom } \rho(S_1) \qquad \langle S_1; e_2\rangle \Downarrow \langle S_2; s\rangle}{\langle S; \texttt{*}e_1\ \texttt{=}\ e_2\rangle \Downarrow \langle S_2[\rho \mapsto (\rho(S_2), (x \to s)]; s\rangle}\ \textsc{UpdLocal}$$

$$\frac{\langle S; e_1\rangle \Downarrow \langle S_1; s_1\rangle \qquad s_1 \neq \text{ptr } x \qquad \langle S_1; e_2\rangle \Downarrow \langle S_2; s_2\rangle}{\langle S; \texttt{*}e_1\ \texttt{=}\ e_2\rangle \Downarrow \langle S_2[\mu \mapsto (\mu(S_2), (s_1 \to s_2)]; s_2\rangle}\ \textsc{UpdGlobal}$$

$$\frac{\langle S; e\rangle \Downarrow \langle S'; \text{ptr } x\rangle \qquad \rho(S')[x] = s}{\langle S; \texttt{*}e\rangle \Downarrow \langle S'; s\rangle}\ \textsc{SelLocal}$$

$$\frac{\langle S; e\rangle \Downarrow \langle S'; \text{ptr } x\rangle \qquad \rho(S')[x] = s}{\langle S; \texttt{*}e\rangle \Downarrow \langle S'; s\rangle}\ \textsc{SelGlobal}$$

$$\frac{\rho(S)[x] = s}{\langle S; \texttt{++}x\rangle \Downarrow \langle S[\rho \mapsto (\rho(S), x \to s+1)]; s+1\rangle}\ \textsc{IncPre}$$

$$\frac{\rho(S)[x] = s}{\langle S; x\texttt{++}\rangle \Downarrow \langle S[\rho \mapsto (\rho(S), x \to s+1)]; s\rangle}\ \textsc{IncPost}$$