1 Representations

1.1 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 `)`
 | postfix-expression `++`
 | postfix-expression `--`

unary-expression ::= postfix-expression
 | `++` unary-expression
 | `--` unary-expression
 | unary-operator postfix-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 ::= bitwise-and-expression
| exclusive-or-expression `^` bitwise-and-expression

bitwise-or-expression ::= exclusive-or-expression
| bitwise-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` | `unsigned` | `char`

type ::= type-qualifier type-specifier

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

2 Symbolic Execution

2.1 Metavariables

e - source language expressions
c - source language statements
s - symbolic expressions
x, α - names
v - symbolic values

2.2 Symbolic Expressions

$$
\begin{aligned}
s ::=\ & v \mid x \\
\mid\ & s \text{ binop } s \\
\mid\ & \text{unop } s \\
\mid\ & \text{sel}(s, (s, .., s)) \\
\mid\ & \text{upd}(s, (s, .., s), s) \\
\mid\ & x(s, .., s) \\
\mid\ & s ? : s : s \\
\mid\ & \text{NewArr } \tau
\end{aligned}
$$

$$
\begin{aligned}
\tau ::=\ & \text{Arr } \tau \\
\mid\ & \text{int8} \\
\mid\ & \text{int32} \\
\mid\ & \text{word32}
\end{aligned}
$$

2.3 Symbolic State

$S = \langle g; \rho; \mu \rangle$
g - path condition
ρ - mapping of names to stack memory
μ - mapping of names to heap memory

2.4 Expressions

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

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

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

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

2.5 Statements

$$\frac{\langle S; e\rangle \Downarrow \langle S'; s\rangle}{\langle S; e\texttt{;}\rangle \Downarrow \langle S'; 0\rangle}\ \textsc{Expression}$$

$$\frac{\forall i \in \{1..n\}.\langle S_k; c_i\rangle \Downarrow \langle S_{i+1}; s_{i+1}\rangle}{\langle S_1; \{c_1..c_n\}\rangle \Downarrow \langle S_{n+1}[\rho \mapsto \rho(S_{n+1})|_{\text{dom }\rho(S_1)}]; s_{n+1}\rangle}\ \textsc{CompoundStatement}$$

$$\frac{\langle S; e\rangle \Downarrow \langle S_1; g_1\rangle \qquad \langle S_1[g \mapsto g(S_1) \wedge g_1]; c_1\rangle \Downarrow \langle S_2; s_2\rangle}{\langle S; \texttt{if}\ e\ c_1\ \texttt{else}\ c_2\rangle \Downarrow \langle S_2; 0\rangle}\ \textsc{IfTrue}$$

$$\frac{\langle S; e\rangle \Downarrow \langle S_1; g_1\rangle \qquad \langle S_1[g \mapsto g(S_1) \wedge \neg g_1]; c_2\rangle \Downarrow \langle S_2; s_2\rangle}{\langle S; \texttt{if}\ e\ c_1\ \texttt{else}\ c_2\rangle \Downarrow \langle S_2; 0\rangle}\ \textsc{IfFalse}$$

2.6 Memory

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

$$\frac{\langle S; e\rangle \Downarrow \langle S'; s\rangle \qquad \text{toOffset}(s) = (\text{ptr }\alpha + s_2) \qquad \rho(S')[\alpha] = s_1}{\langle S; \texttt{*}e\rangle \Downarrow \langle S'; \text{sel}(s_1, s_2)\rangle}\ \textsc{DerefStack}$$

$$\frac{\langle S; e\rangle \Downarrow \langle S'; s\rangle \qquad \text{toOffset}(s) = (\text{ptr }\alpha + s_2) \qquad \mu(S')[\alpha] = s_1}{\langle S; \texttt{*}e\rangle \Downarrow \langle S'; \text{sel}(s_1, s_2)\rangle}\ \textsc{DerefHeap}$$

$$\frac{\rho(S_1)[x] = \text{ptr }\alpha \qquad \rho(S_1)[\alpha] = s \qquad \forall i \in \{1..n\}.\langle S_i; e_i\rangle \Downarrow \langle S_{i+1}; s_i\rangle}{\langle S_1; x\texttt{[}e_1\texttt{]}\texttt{..}\texttt{[}e_n\texttt{]}\rangle \Downarrow \langle S_{i+1}; \text{sel}(s, (s_1, .., s_n))\rangle}\ \textsc{SelLocal}$$

$$\frac{\rho(S_1)[x] = \text{ptr }\alpha \qquad \mu(S_1)[\alpha] = s \qquad \forall i \in \{1..n\}.\langle S_i; e_i\rangle \Downarrow \langle S_{i+1}; s_i\rangle}{\langle S_1; x\texttt{[}e_1\texttt{]}\texttt{..}\texttt{[}e_n\texttt{]}\rangle \Downarrow \langle S_{i+1}; \text{sel}(s, (s_1, .., s_n))\rangle}\ \textsc{SelGlobal}$$

$$\frac{x \notin \text{dom } (\rho(S)) \qquad \dim(\tau) = 0}{\langle S; \tau \;\; x\texttt{;}\rangle \Downarrow \langle S[\rho \mapsto \rho[x \mapsto 0]]; 0\rangle} \textsc{DeclareLocal1}$$

$$\frac{x \notin \text{dom } (\rho(S)) \qquad \dim(\tau) > 0 \qquad \alpha \text{ is fresh}}{\langle S; \tau \;\; x\texttt{;}\rangle \Downarrow \langle S[\rho \mapsto \rho[x \mapsto \text{ptr } \alpha, \alpha \mapsto \text{NewArr } \tau]]; \text{ptr } \alpha\rangle} \textsc{DeclareLocal2}$$

$$\frac{x \in \text{dom } (\rho(S)) \qquad \langle S; e\rangle \Downarrow \langle S'; s\rangle}{\langle S; x \;\texttt{=}\; e\texttt{;}\rangle \Downarrow \langle S'[\rho \mapsto \rho[x \mapsto s]]; s\rangle} \textsc{AssignLocal}$$

$$\frac{\rho(S_1)[x] = \text{ptr } \alpha \qquad \rho(S_1)[\alpha] = s \qquad \forall i \in \{1..n\}.\langle S_i; e_i\rangle \Downarrow \langle S_{i+1}; s_i\rangle \qquad \langle S_{n+1}; e_v\rangle \Downarrow \langle S'; s'\rangle}{\langle S_1; x\texttt{[}e_1\texttt{]}\texttt{..}\texttt{[}e_n\texttt{]} \;\texttt{=}\; e_v\rangle \Downarrow \langle S'[\rho \mapsto \rho(S')[\alpha \mapsto \text{upd}(s, (s_1, .., s_n), s')]]; s'\rangle} \textsc{UpdLocal}$$

$$\frac{\rho(S_1)[x] = \text{ptr } \alpha \qquad \mu(S_1)[\alpha] = s \qquad \forall i \in \{1..n\}.\langle S_i; e_i\rangle \Downarrow \langle S_{i+1}; s_i\rangle \qquad \langle S_{n+1}; e_v\rangle \Downarrow \langle S'; s'\rangle}{\langle S_1; x\texttt{[}e_1\texttt{]}\texttt{..}\texttt{[}e_n\texttt{]} \;\texttt{=}\; e_v\rangle \Downarrow \langle S'[\mu \mapsto \mu(S')[\alpha \mapsto \text{upd}(s, (s_1, .., s_n), s')]]; s'\rangle} \textsc{UpdGlobal}$$

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

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