Featherweight VeriFast Formal Definition, Soundness Proof, Mechanisation

Frédéric Vogels, Bart Jacobs, and Frank Piessens

October 14, 2015

Contents

Contents

The Programming Language

The Programming Language

$z \in \mathbb{Z}, n \in \mathbb{N}$

The Programming Language

$z \in \mathbb{Z}, n \in \mathbb{N}$

$x \in \mathit{Vars}$

The Programming Language

$$
\begin{array}{rl}
 & z \in \mathbb{Z}, n \in \mathbb{N} \\
 & x \in \mathit{Vars} \\
e ::= & z \mid x \mid e + e \mid e - e
\end{array}
$$

The Programming Language

$$
\begin{array}{rl}
 & z \in \mathbb{Z}, n \in \mathbb{N} \\
 & x \in \mathit{Vars} \\
e ::= & z \mid x \mid e + e \mid e - e \\
b ::= & e = e \mid e < e \mid \neg b
\end{array}
$$

The Programming Language

$$
\begin{array}{rl}
 & z \in \mathbb{Z}, n \in \mathbb{N} \\
 & x \in \mathit{Vars} \\
e ::= & z \mid x \mid e + e \mid e - e \\
b ::= & e = e \mid e < e \mid \neg b \\
c ::= & x := e \mid (c; c) \mid \textbf{if } b \textbf{ then } c \textbf{ else } c \mid \textbf{while } b \textbf{ do } c \\
 & \mid r(\overline{e}) \mid x := \textbf{malloc}(n) \mid x := [e] \mid [e] := e \mid \textbf{free}(e)
\end{array}
$$

The Programming Language

$$
\begin{array}{rl}
 & z \in \mathbb{Z}, n \in \mathbb{N} \\
 & x \in \mathit{Vars} \\
e ::= & z \mid x \mid e + e \mid e - e \\
b ::= & e = e \mid e < e \mid \neg b \\
c ::= & x := e \mid (c; c) \mid \textbf{if } b \textbf{ then } c \textbf{ else } c \mid \textbf{while } b \textbf{ do } c \\
 & \mid r(\overline{e}) \mid x := \textbf{malloc}(n) \mid x := [e] \mid [e] := e \mid \textbf{free}(e) \\
\mathit{rdef} ::= & \textbf{routine } r(\overline{x}) = c
\end{array}
$$

Example Program

```
routine range(i, n, result) =
  if i = n then
    [result] := 0
  else (
    head := malloc(2);
    [result] := head;
    [head] := i;
    range(i + 1, n, head + 1)
  )
```

```
routine dispose(list) =
  if list = 0 then
    dummy := dummy
  else (
    tail := [list + 1];
    free(list);
    dispose(tail)
  )
```

```
cell := malloc(1); range(0, 100000000, cell);
list := [cell]; free(cell); dispose(list)
```

Contents

Example Concrete Execution

pair := **malloc**(2);

[pair] := 0;

free(pair)

Example Concrete Execution

// $s = \mathbf{0}, h = \mathbf{0}$
pair := **malloc**(2);

[pair] := 0;

free(pair)

where

$$\mathbf{0} = \lambda x.\ 0 = \text{empty store} = \text{empty heap} = \{\!\{\}\!\} = \text{empty multiset}$$

Example Concrete Execution

// $s = \mathbf{0}, h = \mathbf{0}$
pair := **malloc**(2);
// $s = \mathbf{0}[\text{pair} := 100], h = \{\!|\text{mb}(100, 2), 100 \mapsto 42, 101 \mapsto 24|\!\}$
[pair] := 0;

free(pair)

where

$$\mathbf{0} = \lambda x.\ 0 = \text{empty store} = \text{empty heap} = \{\!|\}\!| = \text{empty multiset}$$

Example Concrete Execution

// $s = \mathbf{0}, h = \mathbf{0}$
pair := **malloc**(2);
// $s = \mathbf{0}[\text{pair} := 100], h = \{\!|\text{mb}(100, 2), 100 \mapsto 42, 101 \mapsto 24|\!\}$
[pair] := 0;

free(pair)

where

$$f[x := y] = \text{function update} = \lambda z.\ \begin{cases} y & \text{if } z = x \\ f(z) & \text{otherwise} \end{cases}$$

$\mathbf{0} = \lambda x.\ 0 = \text{empty store} = \text{empty heap} = \{\!|\}\!| = \text{empty multiset}$

$\{\!|e_1, \ldots, e_n|\!\} = \mathbf{0} + \{\!|e_1|\!\} + \cdots + \{\!|e_n|\!\}$

$M + \{\!|e|\!\} = M[e := M(e) + 1]$

Example Concrete Execution

$// \ s = \mathbf{0}, h = \mathbf{0}$
pair := **malloc**(2);
$// \ s = \mathbf{0}[\text{pair} := 100], h = \{\!|\text{mb}(100, 2), 100 \mapsto 42, 101 \mapsto 24|\!\}$
[pair] := 0;
$// \ s = \mathbf{0}[\text{pair} := 100], h = \{\!|\text{mb}(100, 2), 100 \mapsto 0, 101 \mapsto 24|\!\}$
free(pair)

where

$$f[x := y] = \text{function update} = \lambda z. \begin{cases} y & \text{if } z = x \\ f(z) & \text{otherwise} \end{cases}$$

$\mathbf{0} = \lambda x.\ 0 = \text{empty store} = \text{empty heap} = \{\!|\}\!| = \text{empty multiset}$

$\{\!|e_1, \ldots, e_n|\!\} = \mathbf{0} + \{\!|e_1|\!\} + \cdots + \{\!|e_n|\!\}$

$M + \{\!|e|\!\} = M[e := M(e) + 1]$

Example Concrete Execution

// $s = \mathbf{0}, h = \mathbf{0}$
pair := **malloc**(2);
// $s = \mathbf{0}[\text{pair} := 100], h = \{\!\{\text{mb}(100, 2), 100 \mapsto 42, 101 \mapsto 24\}\!\}$
[pair] := 0;
// $s = \mathbf{0}[\text{pair} := 100], h = \{\!\{\text{mb}(100, 2), 100 \mapsto 0, 101 \mapsto 24\}\!\}$
free(pair)
// $s = \mathbf{0}[\text{pair} := 100], h = \mathbf{0}$

where

$$f[x := y] = \text{function update} = \lambda z. \begin{cases} y & \text{if } z = x \\ f(z) & \text{otherwise} \end{cases}$$

$\mathbf{0} = \lambda x.\, 0 = \text{empty store} = \text{empty heap} = \{\!\{\}\!\} = \text{empty multiset}$

$\{\!\{e_1, \ldots, e_n\}\!\} = \mathbf{0} + \{\!\{e_1\}\!\} + \cdots + \{\!\{e_n\}\!\}$

$M + \{\!\{e\}\!\} = M[e := M(e) + 1]$

Concrete Execution: Example Trace

routine range(i, n, r) =

if i = n **then** l := 0 **else** (

l := **malloc**(2);

[l] := i; range(i + 1, n, l + 1)

⋮ *(Execution of 3 nested* range *calls)*

);
[r] := l

Concrete Execution: Example Trace

routine range(i, n, r) =
s:**0**[i:5, n:8, r:41], h:$h_0 \uplus \{\!\!\{41 \mapsto 77\}\!\!\}$
if i = n **then** l := 0 **else** (

l := **malloc**(2);

[l] := i; range(i + 1, n, l + 1)

⋮ *(Execution of 3 nested* range *calls)*

);
[r] := l

where h_0 : $\{\!\!\{$mb(30),30↦3,31↦40,mb(40, 2),40↦4$\}\!\!\}$

Concrete Execution: Example Trace

routine range(i, n, r) =
s:**0**[i:5, n:8, r:41], h:$h_0 \uplus \{\!\{41 \mapsto 77\}\!\}$
if i = n **then** l := 0 **else** (
s:**0**[i:5, n:8, r:41], h:$h_0 \uplus \{\!\{41 \mapsto 77\}\!\}$
l := **malloc**(2);

[l] := i; range(i + 1, n, l + 1)

⋮ *(Execution of 3 nested* range *calls)*

);
[r] := l

where h_0 : $\{\!\{$mb(30),30↦3,31↦40,mb(40, 2),40↦4$\}\!\}$

Concrete Execution: Example Trace

routine range(i, n, r) =
s:**0**[i:5, n:8, r:41], h:$h_0 \uplus \{\!\{41 \mapsto 77\}\!\}$
if i = n **then** l := 0 **else** (
s:**0**[i:5, n:8, r:41], h:$h_0 \uplus \{\!\{41 \mapsto 77\}\!\}$
l := **malloc**(2);
s:**0**[i:5, n:8, r:41, l:50], h:$h_0 \uplus \{\!\{41 \mapsto 77, \mathrm{mb}(50, 2), 50 \mapsto 88, 51 \mapsto 99\}\!\}$
[l] := i; range(i + 1, n, l + 1)

⋮ *(Execution of 3 nested* range *calls)*

);
[r] := l

where h_0 : $\{\!\{\mathrm{mb}(30), 30 \mapsto 3, 31 \mapsto 40, \mathrm{mb}(40, 2), 40 \mapsto 4\}\!\}$

Concrete Execution: Example Trace

routine range(i, n, r) =
s:**0**[i:5, n:8, r:41], h:$h_0 \uplus \{\!\{41 \mapsto 77\}\!\}$
if i = n **then** l := 0 **else** (
s:**0**[i:5, n:8, r:41], h:$h_0 \uplus \{\!\{41 \mapsto 77\}\!\}$
l := **malloc**(2);
s:**0**[i:5, n:8, r:41, l:50], h:$h_0 \uplus \{\!\{41 \mapsto 77, \mathrm{mb}(50, 2), 50 \mapsto 88, 51 \mapsto 99\}\!\}$
[l] := i; range(i + 1, n, l + 1)

⋮ *(Execution of 3 nested* range *calls)*

s:**0**[i:5, n:8, r:41, l:50], h:$h_0 \uplus \{\!\{41 \mapsto 77, \mathrm{mb}(50, 2), 50 \mapsto 5, 51 \mapsto 60,$
$\mathrm{mb}(60, 2), 60 \mapsto 6, 61 \mapsto 70, \mathrm{mb}(70, 2), 70 \mapsto 7, 71 \mapsto 0\}\!\}$
);
[r] := l

where h_0 : $\{\!\{\mathrm{mb}(30), 30 \mapsto 3, 31 \mapsto 40, \mathrm{mb}(40, 2), 40 \mapsto 4\}\!\}$

Concrete Execution: Example Trace

routine range(i, n, r) =
s:**0**[i:5, n:8, r:41], h:$h_0 \uplus \{\!\{41 \mapsto 77\}\!\}$
if i = n **then** l := 0 **else** (
s:**0**[i:5, n:8, r:41], h:$h_0 \uplus \{\!\{41 \mapsto 77\}\!\}$
l := **malloc**(2);
s:**0**[i:5, n:8, r:41, l:50], h:$h_0 \uplus \{\!\{41 \mapsto 77, \mathrm{mb}(50, 2), 50 \mapsto 88, 51 \mapsto 99\}\!\}$
[l] := i; range(i + 1, n, l + 1)

⋮ *(Execution of 3 nested* range *calls)*

s:**0**[i:5, n:8, r:41, l:50], h:$h_0 \uplus \{\!\{41 \mapsto 77, \mathrm{mb}(50, 2), 50 \mapsto 5, 51 \mapsto 60,$
$\mathrm{mb}(60, 2), 60 \mapsto 6, 61 \mapsto 70, \mathrm{mb}(70, 2), 70 \mapsto 7, 71 \mapsto 0\}\!\}$
);
[r] := l
s:**0**[i:5, n:8, r:41, l:50], h:$h_0 \uplus \{\!\{41 \mapsto 50, \mathrm{mb}(50, 2), 50 \mapsto 5, 51 \mapsto 60,$
$\mathrm{mb}(60, 2), 60 \mapsto 6, 61 \mapsto 70, \mathrm{mb}(70, 2), 70 \mapsto 7, 71 \mapsto 0\}\!\}$
where h_0 : $\{\!\{\mathrm{mb}(30), 30 \mapsto 3, 31 \mapsto 40, \mathrm{mb}(40, 2), 40 \mapsto 4\}\!\}$

Concrete Execution States

Concrete Execution States

$$CStores = \quad Vars \to \mathbb{Z}$$

Concrete Execution States

$$
\begin{aligned}
\mathit{CStores} &= \mathit{Vars} \to \mathbb{Z} \\
\mathit{CPredicates} &= \{\mapsto, \mathsf{mb}\} \\
\mathit{CChunks} &= \{p(\ell, v) \mid p \in \mathit{CPredicates}, \ell, v \in \mathbb{Z}\}
\end{aligned}
$$

$\ell \mapsto v$ is alternative syntax for $\mapsto(\ell, v)$

Concrete Execution States

$$
\begin{aligned}
\mathit{CStores} &= \mathit{Vars} \to \mathbb{Z} \\
\mathit{CPredicates} &= \{\mapsto, \mathsf{mb}\} \\
\mathit{CChunks} &= \{p(\ell, v) \mid p \in \mathit{CPredicates}, \ell, v \in \mathbb{Z}\} \\
\mathit{CHeaps} &= \mathit{Bags}(\mathit{CChunks}) = \mathit{CChunks} \to \mathbb{N}
\end{aligned}
$$

$\ell \mapsto v$ is alternative syntax for $\mapsto(\ell, v)$

Concrete Execution States

$$
\begin{aligned}
\mathit{CStores} &= \mathit{Vars} \to \mathbb{Z} \\
\mathit{CPredicates} &= \{\mapsto, \mathsf{mb}\} \\
\mathit{CChunks} &= \{p(\ell, v) \mid p \in \mathit{CPredicates}, \ell, v \in \mathbb{Z}\} \\
\mathit{CHeaps} &= \mathit{Bags}(\mathit{CChunks}) = \mathit{CChunks} \to \mathbb{N} \\
\mathit{CStates} &= \mathit{CStores} \times \mathit{CHeaps}
\end{aligned}
$$

$\ell \mapsto v$ is alternative syntax for $\mapsto(\ell, v)$

Concrete Execution: Singleton Outcomes

$$\mathsf{exec}(\mathsf{p} := 42)((\mathbf{0}, \mathbf{0})) =$$

Concrete Execution: Singleton Outcomes

$$\mathsf{exec}(\mathsf{p} := 42)((\mathbf{0}, \mathbf{0})) = \langle(\mathbf{0}[\mathsf{p} := 42], \mathbf{0})\rangle$$

Concrete Execution: Demonic Choice

$$\mathsf{exec}(\mathsf{p} := \mathbf{malloc}(0))((\mathbf{0}, \mathbf{0})) \;=$$

Concrete Execution: Demonic Choice

$$\begin{aligned}\mathsf{exec}(\mathsf{p} := \mathbf{malloc}(0))((\mathbf{0},\mathbf{0})) \;&= \\ &\langle(\mathbf{0}[\mathsf{p} := 1], \{\!|\mathsf{mb}(1,0)|\!\})\rangle\end{aligned}$$

Concrete Execution: Demonic Choice

$$\begin{aligned}\mathsf{exec}(\mathsf{p} := \mathbf{malloc}(0))((\mathbf{0},\mathbf{0})) \;=\; & \\ & \langle(\mathbf{0}[\mathsf{p} := 1], \{\!|\mathsf{mb}(1,0)|\!\})\rangle \\ \otimes \quad & \langle(\mathbf{0}[\mathsf{p} := 2], \{\!|\mathsf{mb}(2,0)|\!\})\rangle\end{aligned}$$

Concrete Execution: Demonic Choice

$$
\begin{aligned}
\mathsf{exec}(\mathsf{p} := \mathbf{malloc}(0))((\mathbf{0}, \mathbf{0})) \;=\; & \\
& \langle (\mathbf{0}[\mathsf{p} := 1], \{\!|\mathsf{mb}(1,0)|\!\}) \rangle \\
\otimes \quad & \langle (\mathbf{0}[\mathsf{p} := 2], \{\!|\mathsf{mb}(2,0)|\!\}) \rangle \\
\otimes \quad & \langle (\mathbf{0}[\mathsf{p} := 3], \{\!|\mathsf{mb}(3,0)|\!\}) \rangle
\end{aligned}
$$

Concrete Execution: Demonic Choice

$$
\begin{aligned}
\mathsf{exec}(\mathsf{p} := \mathbf{malloc}(0))((\mathbf{0}, \mathbf{0})) \;=\; & \\
& \langle (\mathbf{0}[\mathsf{p} := 1], \{\!|\mathsf{mb}(1,0)|\!\}) \rangle \\
\otimes \quad & \langle (\mathbf{0}[\mathsf{p} := 2], \{\!|\mathsf{mb}(2,0)|\!\}) \rangle \\
\otimes \quad & \langle (\mathbf{0}[\mathsf{p} := 3], \{\!|\mathsf{mb}(3,0)|\!\}) \rangle \\
\otimes \quad & \cdots
\end{aligned}
$$

Concrete Execution: Demonic Choice

$$\begin{aligned}\mathsf{exec}(\mathsf{p} := \mathbf{malloc}(0))((\mathbf{0},\mathbf{0})) \;=\; & \\ & \langle(\mathbf{0}[\mathsf{p} := 1], \{\!|\mathsf{mb}(1,0)|\!\})\rangle \\ \otimes \quad & \langle(\mathbf{0}[\mathsf{p} := 2], \{\!|\mathsf{mb}(2,0)|\!\})\rangle \\ \otimes \quad & \langle(\mathbf{0}[\mathsf{p} := 3], \{\!|\mathsf{mb}(3,0)|\!\})\rangle \\ \otimes \quad & \ldots\end{aligned}$$

$$\mathsf{exec}(\mathsf{p} := \mathbf{malloc}(1))((\mathbf{0},\mathbf{0})) \;=$$

Concrete Execution: Demonic Choice

$$\begin{aligned}\mathsf{exec}(\mathsf{p} := \mathbf{malloc}(0))((\mathbf{0},\mathbf{0})) \;=\; & \\ & \langle(\mathbf{0}[\mathsf{p} := 1], \{\!|\mathsf{mb}(1,0)|\!\})\rangle \\ \otimes \quad & \langle(\mathbf{0}[\mathsf{p} := 2], \{\!|\mathsf{mb}(2,0)|\!\})\rangle \\ \otimes \quad & \langle(\mathbf{0}[\mathsf{p} := 3], \{\!|\mathsf{mb}(3,0)|\!\})\rangle \\ \otimes \quad & \ldots\end{aligned}$$

$$\begin{aligned}\mathsf{exec}(\mathsf{p} := \mathbf{malloc}(1))((\mathbf{0},\mathbf{0})) \;=\; & \\ & \langle(\mathbf{0}[\mathsf{p} := 1], \{\!|\mathsf{mb}(1,1), 1 \mapsto 0|\!\})\rangle\end{aligned}$$

Concrete Execution: Demonic Choice

$$\begin{aligned}\mathsf{exec}(\mathsf{p} := \mathbf{malloc}(0))((\mathbf{0},\mathbf{0})) \;=\; & \\ & \langle(\mathbf{0}[\mathsf{p} := 1], \{\!|\mathsf{mb}(1,0)|\!\})\rangle \\ \otimes\; & \langle(\mathbf{0}[\mathsf{p} := 2], \{\!|\mathsf{mb}(2,0)|\!\})\rangle \\ \otimes\; & \langle(\mathbf{0}[\mathsf{p} := 3], \{\!|\mathsf{mb}(3,0)|\!\})\rangle \\ \otimes\; & \cdots\end{aligned}$$

$$\begin{aligned}\mathsf{exec}(\mathsf{p} := \mathbf{malloc}(1))((\mathbf{0},\mathbf{0})) \;=\; & \\ & \langle(\mathbf{0}[\mathsf{p} := 1], \{\!|\mathsf{mb}(1,1), 1 \mapsto 0|\!\})\rangle \\ \otimes\; & \langle(\mathbf{0}[\mathsf{p} := 1], \{\!|\mathsf{mb}(1,1), 1 \mapsto 1|\!\})\rangle \otimes \cdots\end{aligned}$$

Concrete Execution: Demonic Choice

$$\begin{aligned}
&\mathsf{exec}(\mathsf{p} := \mathbf{malloc}(0))((\mathbf{0},\mathbf{0})) \;=\\
&\qquad \langle(\mathbf{0}[\mathsf{p} := 1], \{\!|\mathsf{mb}(1,0)|\!\})\rangle\\
&\quad \otimes \;\; \langle(\mathbf{0}[\mathsf{p} := 2], \{\!|\mathsf{mb}(2,0)|\!\})\rangle\\
&\quad \otimes \;\; \langle(\mathbf{0}[\mathsf{p} := 3], \{\!|\mathsf{mb}(3,0)|\!\})\rangle\\
&\quad \otimes \;\; \cdots
\end{aligned}$$

$$\begin{aligned}
&\mathsf{exec}(\mathsf{p} := \mathbf{malloc}(1))((\mathbf{0},\mathbf{0})) \;=\\
&\quad \langle(\mathbf{0}[\mathsf{p} := 1], \{\!|\mathsf{mb}(1,1), 1 \mapsto 0|\!\})\rangle\\
&\qquad \otimes \; \langle(\mathbf{0}[\mathsf{p} := 1], \{\!|\mathsf{mb}(1,1), 1 \mapsto 1|\!\})\rangle \otimes \cdots\\
&\quad \otimes \; \langle(\mathbf{0}[\mathsf{p} := 2], \{\!|\mathsf{mb}(2,1), 2 \mapsto 0|\!\})\rangle\\
&\qquad \otimes \; \langle(\mathbf{0}[\mathsf{p} := 2], \{\!|\mathsf{mb}(2,1), 2 \mapsto 1|\!\})\rangle \otimes \cdots
\end{aligned}$$

Concrete Execution: Demonic Choice

$$\begin{aligned}
&\mathsf{exec}(\mathsf{p} := \mathbf{malloc}(0))((\mathbf{0},\mathbf{0})) \;=\\
&\qquad \langle(\mathbf{0}[\mathsf{p} := 1], \{\!|\mathsf{mb}(1,0)|\!\})\rangle\\
&\quad \otimes \;\; \langle(\mathbf{0}[\mathsf{p} := 2], \{\!|\mathsf{mb}(2,0)|\!\})\rangle\\
&\quad \otimes \;\; \langle(\mathbf{0}[\mathsf{p} := 3], \{\!|\mathsf{mb}(3,0)|\!\})\rangle\\
&\quad \otimes \;\; \cdots
\end{aligned}$$

$$\begin{aligned}
&\mathsf{exec}(\mathsf{p} := \mathbf{malloc}(1))((\mathbf{0},\mathbf{0})) \;=\\
&\quad \langle(\mathbf{0}[\mathsf{p} := 1], \{\!|\mathsf{mb}(1,1), 1 \mapsto 0|\!\})\rangle\\
&\qquad \otimes \; \langle(\mathbf{0}[\mathsf{p} := 1], \{\!|\mathsf{mb}(1,1), 1 \mapsto 1|\!\})\rangle \otimes \cdots\\
&\quad \otimes \; \langle(\mathbf{0}[\mathsf{p} := 2], \{\!|\mathsf{mb}(2,1), 2 \mapsto 0|\!\})\rangle\\
&\qquad \otimes \; \langle(\mathbf{0}[\mathsf{p} := 2], \{\!|\mathsf{mb}(2,1), 2 \mapsto 1|\!\})\rangle \otimes \cdots\\
&\quad \otimes \; \langle(\mathbf{0}[\mathsf{p} := 3], \{\!|\mathsf{mb}(3,1), 3 \mapsto 0|\!\})\rangle\\
&\qquad \otimes \; \langle(\mathbf{0}[\mathsf{p} := 3], \{\!|\mathsf{mb}(3,1), 3 \mapsto 1|\!\})\rangle \otimes \cdots\\
&\quad \otimes \cdots
\end{aligned}$$

Concrete Execution: Failure, Nontermination, Angelic Choice

$$\mathsf{exec}([0] := 33)((\mathbf{0}, \mathbf{0})) =$$

Concrete Execution: Failure, Nontermination, Angelic Choice

$$\mathsf{exec}([0] := 33)((\mathbf{0}, \mathbf{0})) = \bot$$

Concrete Execution: Failure, Nontermination, Angelic Choice

$$\mathsf{exec}([0] := 33)((\mathbf{0}, \mathbf{0})) = \bot$$

$$\mathsf{exec}(\mathsf{recurse}())((\mathbf{0}, \mathbf{0})) =$$
$$\quad \text{where } \mathbf{routine}\ \mathsf{recurse}() = \mathsf{recurse}()$$

Concrete Execution: Failure, Nontermination, Angelic Choice

$$\mathsf{exec}([0] := 33)((\mathbf{0}, \mathbf{0})) = \bot$$

$$\mathsf{exec}(\mathsf{recurse}())((\mathbf{0}, \mathbf{0})) = \top$$
$$\text{where } \textbf{routine } \mathsf{recurse}() = \mathsf{recurse}()$$

Concrete Execution: Failure, Nontermination, Angelic Choice

$$\mathsf{exec}([0] := 33)((\mathbf{0}, \mathbf{0})) = \bot$$

$$\mathsf{exec}(\mathsf{recurse}())((\mathbf{0}, \mathbf{0})) = \top$$
$$\quad \text{where } \mathbf{routine}\ \mathsf{recurse}() = \mathsf{recurse}()$$

$$\mathsf{exec}(\mathbf{backtrack}(c_1, c_2))(\sigma) =$$

Concrete Execution: Failure, Nontermination, Angelic Choice

$$\mathsf{exec}([0] := 33)((\mathbf{0}, \mathbf{0})) = \bot$$

$$\mathsf{exec}(\mathsf{recurse}())((\mathbf{0}, \mathbf{0})) = \top$$
$$\quad \text{where } \mathbf{routine}\ \mathsf{recurse}() = \mathsf{recurse}()$$

$$\mathsf{exec}(\mathbf{backtrack}(c_1, c_2))(\sigma) = \mathsf{exec}(c_1)(\sigma) \oplus \mathsf{exec}(c_2)(\sigma)$$

Concrete Execution

$$CMutators = CStates \rightarrow Outcomes(CStates)$$

Concrete Execution

$$CMutators = CStates \rightarrow Outcomes(CStates)$$

$$\text{exec} \in Commands \rightarrow CMutators$$

Outcomes

$$\phi \quad ::=$$

Outcomes

$$\phi \quad ::= \quad \langle \sigma \rangle \quad \text{singleton outcome}$$

Outcomes

$$
\begin{array}{lcll}
\phi & ::= & \langle\sigma\rangle & \text{singleton outcome} \\
 & | & \bigotimes \Phi & \text{demonic choice}
\end{array}
$$

Outcomes

$$
\begin{array}{lcll}
\phi & ::= & \langle \sigma \rangle & \text{singleton outcome} \\
 & | & \bigotimes \Phi & \text{demonic choice} \\
 & | & \bigoplus \Phi & \text{angelic choice}
\end{array}
$$

Outcomes

$$
\begin{array}{lcll}
\phi & ::= & \langle\sigma\rangle & \text{singleton outcome} \\
 & | & \bigotimes \Phi & \text{demonic choice} \\
 & | & \bigoplus \Phi & \text{angelic choice}
\end{array}
$$

$$\sigma \in \mathcal{S} \quad \Rightarrow \quad \langle\sigma\rangle \in \mathit{Outcomes}(\mathcal{S})$$

Outcomes

$$
\begin{array}{rcll}
\phi & ::= & \langle\sigma\rangle & \text{singleton outcome} \\
 & | & \bigotimes \Phi & \text{demonic choice} \\
 & | & \bigoplus \Phi & \text{angelic choice}
\end{array}
$$

$$
\begin{array}{rcl}
\sigma \in \mathcal{S} & \Rightarrow & \langle\sigma\rangle \in \mathit{Outcomes}(\mathcal{S}) \\
\Phi \subseteq \mathit{Outcomes}(\mathcal{S}) & \Rightarrow & \bigotimes \Phi \in \mathit{Outcomes}(\mathcal{S})
\end{array}
$$

Outcomes

$$
\begin{array}{rcll}
\phi & ::= & \langle\sigma\rangle & \text{singleton outcome} \\
 & | & \bigotimes \Phi & \text{demonic choice} \\
 & | & \bigoplus \Phi & \text{angelic choice}
\end{array}
$$

$$
\begin{array}{rcl}
\sigma \in \mathcal{S} & \Rightarrow & \langle\sigma\rangle \in \mathit{Outcomes}(\mathcal{S}) \\
\Phi \subseteq \mathit{Outcomes}(\mathcal{S}) & \Rightarrow & \bigotimes \Phi \in \mathit{Outcomes}(\mathcal{S}) \\
\Phi \subseteq \mathit{Outcomes}(\mathcal{S}) & \Rightarrow & \bigoplus \Phi \in \mathit{Outcomes}(\mathcal{S})
\end{array}
$$

Outcomes

$$
\begin{array}{rcll}
\phi & ::= & \langle\sigma\rangle & \text{singleton outcome} \\
 & | & \bigotimes \Phi & \text{demonic choice} \\
 & | & \bigoplus \Phi & \text{angelic choice}
\end{array}
$$

$$
\begin{array}{rcl}
\sigma \in \mathcal{S} & \Rightarrow & \langle\sigma\rangle \in \mathit{Outcomes}(\mathcal{S}) \\
\Phi \subseteq \mathit{Outcomes}(\mathcal{S}) & \Rightarrow & \bigotimes \Phi \in \mathit{Outcomes}(\mathcal{S}) \\
\Phi \subseteq \mathit{Outcomes}(\mathcal{S}) & \Rightarrow & \bigoplus \Phi \in \mathit{Outcomes}(\mathcal{S})
\end{array}
$$

$$
\phi_1 \otimes \phi_2 \quad = \quad \text{binary demonic choice}
$$

Outcomes

$$
\begin{array}{rcll}
\phi & ::= & \langle\sigma\rangle & \text{singleton outcome} \\
 & | & \bigotimes \Phi & \text{demonic choice} \\
 & | & \bigoplus \Phi & \text{angelic choice}
\end{array}
$$

$$
\begin{array}{rcl}
\sigma \in \mathcal{S} & \Rightarrow & \langle\sigma\rangle \in \mathit{Outcomes}(\mathcal{S}) \\
\Phi \subseteq \mathit{Outcomes}(\mathcal{S}) & \Rightarrow & \bigotimes \Phi \in \mathit{Outcomes}(\mathcal{S}) \\
\Phi \subseteq \mathit{Outcomes}(\mathcal{S}) & \Rightarrow & \bigoplus \Phi \in \mathit{Outcomes}(\mathcal{S})
\end{array}
$$

$$
\phi_1 \otimes \phi_2 \quad = \quad \bigotimes\{\phi_1, \phi_2\} \quad \text{binary demonic choice}
$$

Outcomes

$$
\begin{array}{rcll}
\phi & ::= & \langle\sigma\rangle & \text{singleton outcome} \\
 & | & \bigotimes \Phi & \text{demonic choice} \\
 & | & \bigoplus \Phi & \text{angelic choice}
\end{array}
$$

$$
\begin{array}{rcl}
\sigma \in \mathcal{S} & \Rightarrow & \langle\sigma\rangle \in \mathit{Outcomes}(\mathcal{S}) \\
\Phi \subseteq \mathit{Outcomes}(\mathcal{S}) & \Rightarrow & \bigotimes \Phi \in \mathit{Outcomes}(\mathcal{S}) \\
\Phi \subseteq \mathit{Outcomes}(\mathcal{S}) & \Rightarrow & \bigoplus \Phi \in \mathit{Outcomes}(\mathcal{S})
\end{array}
$$

$$
\begin{array}{rcll}
\phi_1 \otimes \phi_2 & = & \bigotimes\{\phi_1, \phi_2\} & \text{binary demonic choice} \\
\phi_1 \oplus \phi_2 & = & & \text{binary angelic choice}
\end{array}
$$

Outcomes

$$
\begin{array}{rcll}
\phi & ::= & \langle\sigma\rangle & \text{singleton outcome} \\
 & | & \bigotimes \Phi & \text{demonic choice} \\
 & | & \bigoplus \Phi & \text{angelic choice}
\end{array}
$$

$$
\begin{array}{rcl}
\sigma \in \mathcal{S} & \Rightarrow & \langle\sigma\rangle \in \mathit{Outcomes}(\mathcal{S}) \\
\Phi \subseteq \mathit{Outcomes}(\mathcal{S}) & \Rightarrow & \bigotimes \Phi \in \mathit{Outcomes}(\mathcal{S}) \\
\Phi \subseteq \mathit{Outcomes}(\mathcal{S}) & \Rightarrow & \bigoplus \Phi \in \mathit{Outcomes}(\mathcal{S})
\end{array}
$$

$$
\begin{array}{rcll}
\phi_1 \otimes \phi_2 & = & \bigotimes\{\phi_1, \phi_2\} & \text{binary demonic choice} \\
\phi_1 \oplus \phi_2 & = & \bigoplus\{\phi_1, \phi_2\} & \text{binary angelic choice}
\end{array}
$$

Outcomes

$$
\begin{array}{rcll}
\phi & ::= & \langle\sigma\rangle & \text{singleton outcome} \\
 & | & \bigotimes \Phi & \text{demonic choice} \\
 & | & \bigoplus \Phi & \text{angelic choice}
\end{array}
$$

$$
\begin{array}{rcl}
\sigma \in \mathcal{S} & \Rightarrow & \langle\sigma\rangle \in \mathit{Outcomes}(\mathcal{S}) \\
\Phi \subseteq \mathit{Outcomes}(\mathcal{S}) & \Rightarrow & \bigotimes \Phi \in \mathit{Outcomes}(\mathcal{S}) \\
\Phi \subseteq \mathit{Outcomes}(\mathcal{S}) & \Rightarrow & \bigoplus \Phi \in \mathit{Outcomes}(\mathcal{S})
\end{array}
$$

$$
\begin{array}{rcll}
\phi_1 \otimes \phi_2 & = & \bigotimes\{\phi_1, \phi_2\} & \text{binary demonic choice} \\
\phi_1 \oplus \phi_2 & = & \bigoplus\{\phi_1, \phi_2\} & \text{binary angelic choice} \\
\top & = & & \text{nontermination}
\end{array}
$$

Outcomes

$$
\begin{array}{rcll}
\phi & ::= & \langle\sigma\rangle & \text{singleton outcome} \\
 & | & \bigotimes \Phi & \text{demonic choice} \\
 & | & \bigoplus \Phi & \text{angelic choice}
\end{array}
$$

$$
\begin{array}{rcl}
\sigma \in \mathcal{S} & \Rightarrow & \langle\sigma\rangle \in \mathit{Outcomes}(\mathcal{S}) \\
\Phi \subseteq \mathit{Outcomes}(\mathcal{S}) & \Rightarrow & \bigotimes \Phi \in \mathit{Outcomes}(\mathcal{S}) \\
\Phi \subseteq \mathit{Outcomes}(\mathcal{S}) & \Rightarrow & \bigoplus \Phi \in \mathit{Outcomes}(\mathcal{S})
\end{array}
$$

$$
\begin{array}{rcll}
\phi_1 \otimes \phi_2 & = & \bigotimes\{\phi_1, \phi_2\} & \text{binary demonic choice} \\
\phi_1 \oplus \phi_2 & = & \bigoplus\{\phi_1, \phi_2\} & \text{binary angelic choice} \\
\top & = & \bigotimes \emptyset & \text{nontermination}
\end{array}
$$

Outcomes

$$
\begin{array}{rcll}
\phi & ::= & \langle\sigma\rangle & \text{singleton outcome} \\
 & | & \bigotimes \Phi & \text{demonic choice} \\
 & | & \bigoplus \Phi & \text{angelic choice}
\end{array}
$$

$$
\begin{array}{rcl}
\sigma \in \mathcal{S} & \Rightarrow & \langle\sigma\rangle \in \mathit{Outcomes}(\mathcal{S}) \\
\Phi \subseteq \mathit{Outcomes}(\mathcal{S}) & \Rightarrow & \bigotimes \Phi \in \mathit{Outcomes}(\mathcal{S}) \\
\Phi \subseteq \mathit{Outcomes}(\mathcal{S}) & \Rightarrow & \bigoplus \Phi \in \mathit{Outcomes}(\mathcal{S})
\end{array}
$$

$$
\begin{array}{rcll}
\phi_1 \otimes \phi_2 & = & \bigotimes\{\phi_1, \phi_2\} & \text{binary demonic choice} \\
\phi_1 \oplus \phi_2 & = & \bigoplus\{\phi_1, \phi_2\} & \text{binary angelic choice} \\
\top & = & \bigotimes \emptyset & \text{nontermination} \\
\bot & = & & \text{failure}
\end{array}
$$

Outcomes

$$
\begin{array}{rcll}
\phi & ::= & \langle\sigma\rangle & \text{singleton outcome} \\
 & | & \bigotimes \Phi & \text{demonic choice} \\
 & | & \bigoplus \Phi & \text{angelic choice}
\end{array}
$$

$$
\begin{array}{rcl}
\sigma \in \mathcal{S} & \Rightarrow & \langle\sigma\rangle \in \mathit{Outcomes}(\mathcal{S}) \\
\Phi \subseteq \mathit{Outcomes}(\mathcal{S}) & \Rightarrow & \bigotimes \Phi \in \mathit{Outcomes}(\mathcal{S}) \\
\Phi \subseteq \mathit{Outcomes}(\mathcal{S}) & \Rightarrow & \bigoplus \Phi \in \mathit{Outcomes}(\mathcal{S})
\end{array}
$$

$$
\begin{array}{rcll}
\phi_1 \otimes \phi_2 & = & \bigotimes\{\phi_1, \phi_2\} & \text{binary demonic choice} \\
\phi_1 \oplus \phi_2 & = & \bigoplus\{\phi_1, \phi_2\} & \text{binary angelic choice} \\
\top & = & \bigotimes \emptyset & \text{nontermination} \\
\bot & = & \bigoplus \emptyset & \text{failure}
\end{array}
$$

Outcomes with Answers

$$\phi \quad ::=$$

Outcomes with Answers

$$\phi \quad ::= \quad \langle \sigma, a \rangle \quad \text{singleton outcome}$$

Outcomes with Answers

$$
\begin{array}{lcll}
\phi & ::= & \langle \sigma, a \rangle & \text{singleton outcome} \\
 & | & \bigotimes \Phi & \text{indexed demonic choice}
\end{array}
$$

Outcomes with Answers

$$
\begin{array}{lcll}
\phi & ::= & \langle \sigma, a \rangle & \text{singleton outcome} \\
 & | & \bigotimes \Phi & \text{indexed demonic choice} \\
 & | & \bigoplus \Phi & \text{indexed angelic choice}
\end{array}
$$

Outcomes with Answers

$$
\begin{array}{lcll}
\phi & ::= & \langle \sigma, a \rangle & \text{singleton outcome} \\
 & | & \bigotimes \Phi & \text{indexed demonic choice} \\
 & | & \bigoplus \Phi & \text{indexed angelic choice}
\end{array}
$$

$$\sigma \in \mathcal{S} \wedge a \in \mathcal{A} \quad \Rightarrow \quad \langle \sigma, a \rangle \in \mathit{Outcomes}(\mathcal{S}, \mathcal{A})$$

Outcomes with Answers

$$
\begin{array}{rcll}
\phi & ::= & \langle \sigma, a \rangle & \text{singleton outcome} \\
 & | & \bigotimes \Phi & \text{indexed demonic choice} \\
 & | & \bigoplus \Phi & \text{indexed angelic choice}
\end{array}
$$

$$
\begin{array}{rcl}
\sigma \in \mathcal{S} \wedge a \in \mathcal{A} & \Rightarrow & \langle \sigma, a \rangle \in \mathit{Outcomes}(\mathcal{S}, \mathcal{A}) \\
\Phi \subseteq \mathit{Outcomes}(\mathcal{S}, \mathcal{A}) & \Rightarrow & \bigotimes \Phi \in \mathit{Outcomes}(\mathcal{S}, \mathcal{A})
\end{array}
$$

Outcomes with Answers

$$
\begin{array}{rcll}
\phi & ::= & \langle \sigma, a \rangle & \text{singleton outcome} \\
 & | & \bigotimes \Phi & \text{indexed demonic choice} \\
 & | & \bigoplus \Phi & \text{indexed angelic choice}
\end{array}
$$

$$
\begin{array}{rcl}
\sigma \in \mathcal{S} \wedge a \in \mathcal{A} & \Rightarrow & \langle \sigma, a \rangle \in \mathit{Outcomes}(\mathcal{S}, \mathcal{A}) \\
\Phi \subseteq \mathit{Outcomes}(\mathcal{S}, \mathcal{A}) & \Rightarrow & \bigotimes \Phi \in \mathit{Outcomes}(\mathcal{S}, \mathcal{A}) \\
\Phi \subseteq \mathit{Outcomes}(\mathcal{S}, \mathcal{A}) & \Rightarrow & \bigoplus \Phi \in \mathit{Outcomes}(\mathcal{S}, \mathcal{A})
\end{array}
$$

Outcomes with Answers

$$
\begin{array}{rcll}
\phi & ::= & \langle \sigma, a \rangle & \text{singleton outcome} \\
 & | & \bigotimes \Phi & \text{indexed demonic choice} \\
 & | & \bigoplus \Phi & \text{indexed angelic choice}
\end{array}
$$

$$
\begin{array}{rcl}
\sigma \in \mathcal{S} \wedge a \in \mathcal{A} & \Rightarrow & \langle \sigma, a \rangle \in \mathit{Outcomes}(\mathcal{S}, \mathcal{A}) \\
\Phi \subseteq \mathit{Outcomes}(\mathcal{S}, \mathcal{A}) & \Rightarrow & \bigotimes \Phi \in \mathit{Outcomes}(\mathcal{S}, \mathcal{A}) \\
\Phi \subseteq \mathit{Outcomes}(\mathcal{S}, \mathcal{A}) & \Rightarrow & \bigoplus \Phi \in \mathit{Outcomes}(\mathcal{S}, \mathcal{A})
\end{array}
$$

$$
\langle \sigma \rangle = \langle \sigma, \mathsf{tt} \rangle \in \mathit{Outcomes}(\mathcal{S}) = \mathit{Outcomes}(\mathcal{S}, \mathsf{unit})
$$

Outcomes: Satisfaction, Coverage

$$\phi \in \mathit{Outcomes}(\mathcal{S}, \mathcal{A})$$

Outcomes: Satisfaction, Coverage

$$\phi \in \mathit{Outcomes}(\mathcal{S}, \mathcal{A}) \quad Q \subseteq \mathcal{S} \times \mathcal{A}$$

Outcomes: Satisfaction, Coverage

$$\phi \in \mathit{Outcomes}(\mathcal{S}, \mathcal{A}) \quad Q \subseteq \mathcal{S} \times \mathcal{A}$$

$$\phi \, \{Q\} \quad (\text{“outcome } \phi \text{ satisfies postcondition } Q\text{”})$$

Outcomes: Satisfaction, Coverage

$$\phi \in \mathit{Outcomes}(\mathcal{S}, \mathcal{A}) \quad Q \subseteq \mathcal{S} \times \mathcal{A}$$

$$\phi \ \{Q\} \quad (\text{“outcome } \phi \text{ satisfies postcondition } Q\text{”})$$

$$\langle \sigma, a \rangle \ \{Q\} \quad \Leftrightarrow$$

Outcomes: Satisfaction, Coverage

$$\phi \in \mathit{Outcomes}(\mathcal{S}, \mathcal{A}) \quad Q \subseteq \mathcal{S} \times \mathcal{A}$$

$$\phi \, \{Q\} \quad (\text{“outcome } \phi \text{ satisfies postcondition } Q\text{”})$$

$$\langle \sigma, a \rangle \, \{Q\} \quad \Leftrightarrow \quad (\sigma, a) \in Q$$

Outcomes: Satisfaction, Coverage

$$\phi \in \mathit{Outcomes}(\mathcal{S}, \mathcal{A}) \quad Q \subseteq \mathcal{S} \times \mathcal{A}$$

$$\phi \, \{Q\} \quad (\text{“outcome } \phi \text{ satisfies postcondition } Q\text{”})$$

$$\begin{array}{lll} \langle \sigma, a \rangle \, \{Q\} & \Leftrightarrow & (\sigma, a) \in Q \\ \bigotimes \Phi \, \{Q\} & \Leftrightarrow & \end{array}$$

Outcomes: Satisfaction, Coverage

$$\phi \in \mathit{Outcomes}(\mathcal{S}, \mathcal{A}) \quad Q \subseteq \mathcal{S} \times \mathcal{A}$$

$$\phi \{Q\} \quad (\text{“outcome } \phi \text{ satisfies postcondition } Q\text{”})$$

$$\begin{array}{rcl} \langle \sigma, a \rangle \{Q\} & \Leftrightarrow & (\sigma, a) \in Q \\ \bigotimes \Phi \{Q\} & \Leftrightarrow & \forall \phi \in \Phi.\ \phi \{Q\} \end{array}$$

Outcomes: Satisfaction, Coverage

$$\phi \in \mathit{Outcomes}(\mathcal{S}, \mathcal{A}) \quad Q \subseteq \mathcal{S} \times \mathcal{A}$$

$$\phi \, \{Q\} \quad (\text{“outcome } \phi \text{ satisfies postcondition } Q\text{”})$$

$$\begin{array}{lcl} \langle \sigma, a \rangle \, \{Q\} & \Leftrightarrow & (\sigma, a) \in Q \\ \bigotimes \Phi \, \{Q\} & \Leftrightarrow & \forall \phi \in \Phi.\ \phi \, \{Q\} \\ \bigoplus \Phi \, \{Q\} & \Leftrightarrow & \end{array}$$

Outcomes: Satisfaction, Coverage

$$\phi \in \mathit{Outcomes}(\mathcal{S}, \mathcal{A}) \quad Q \subseteq \mathcal{S} \times \mathcal{A}$$

$$\phi \{Q\} \quad (\text{“outcome } \phi \text{ satisfies postcondition } Q\text{”})$$

$$\begin{array}{lcl} \langle \sigma, a \rangle \{Q\} & \Leftrightarrow & (\sigma, a) \in Q \\ \bigotimes \Phi \{Q\} & \Leftrightarrow & \forall \phi \in \Phi.\ \phi \{Q\} \\ \bigoplus \Phi \{Q\} & \Leftrightarrow & \exists \phi \in \Phi.\ \phi \{Q\} \end{array}$$

Outcomes: Satisfaction, Coverage

$$\phi \in \mathit{Outcomes}(\mathcal{S}, \mathcal{A}) \quad Q \subseteq \mathcal{S} \times \mathcal{A}$$

$$\phi \{Q\} \quad (\text{“outcome } \phi \text{ satisfies postcondition } Q\text{”})$$

$$\begin{array}{lcl} \langle \sigma, a \rangle \{Q\} & \Leftrightarrow & (\sigma, a) \in Q \\ \bigotimes \Phi \{Q\} & \Leftrightarrow & \forall \phi \in \Phi.\ \phi \{Q\} \\ \bigoplus \Phi \{Q\} & \Leftrightarrow & \exists \phi \in \Phi.\ \phi \{Q\} \end{array}$$

$$\phi \Rrightarrow \phi'$$

Outcomes: Satisfaction, Coverage

$$\phi \in \mathit{Outcomes}(\mathcal{S}, \mathcal{A}) \quad Q \subseteq \mathcal{S} \times \mathcal{A}$$

$$\phi\ \{Q\} \quad (\text{“outcome } \phi \text{ satisfies postcondition } Q\text{”})$$

$$\begin{array}{rcl} \langle \sigma, a \rangle\ \{Q\} & \Leftrightarrow & (\sigma, a) \in Q \\ \bigotimes \Phi\ \{Q\} & \Leftrightarrow & \forall \phi \in \Phi.\ \phi\ \{Q\} \\ \bigoplus \Phi\ \{Q\} & \Leftrightarrow & \exists \phi \in \Phi.\ \phi\ \{Q\} \end{array}$$

$$\phi \Rrightarrow \phi' \quad \Leftrightarrow \quad \forall Q.\ \phi\ \{Q\} \Rightarrow \phi'\ \{Q\}$$

Outcomes: Satisfaction, Coverage

$$\phi \in \mathit{Outcomes}(\mathcal{S}, \mathcal{A}) \quad Q \subseteq \mathcal{S} \times \mathcal{A}$$

$$\phi\ \{Q\} \quad (\text{“outcome } \phi \text{ satisfies postcondition } Q\text{”})$$

$$\begin{array}{rcl} \langle \sigma, a \rangle\ \{Q\} & \Leftrightarrow & (\sigma, a) \in Q \\ \bigotimes \Phi\ \{Q\} & \Leftrightarrow & \forall \phi \in \Phi.\ \phi\ \{Q\} \\ \bigoplus \Phi\ \{Q\} & \Leftrightarrow & \exists \phi \in \Phi.\ \phi\ \{Q\} \end{array}$$

$$\begin{array}{rcl} \phi \Rrightarrow \phi' & \Leftrightarrow & \forall Q.\ \phi\ \{Q\} \Rightarrow \phi'\ \{Q\} \\ C \Rrightarrow C' & & \end{array}$$

Outcomes: Satisfaction, Coverage

$$\phi \in \mathit{Outcomes}(\mathcal{S}, \mathcal{A}) \quad Q \subseteq \mathcal{S} \times \mathcal{A}$$

$$\phi \{Q\} \quad (\text{“outcome } \phi \text{ satisfies postcondition } Q\text{”})$$

$$\begin{array}{rcl} \langle \sigma, a \rangle \{Q\} & \Leftrightarrow & (\sigma, a) \in Q \\ \bigotimes \Phi \{Q\} & \Leftrightarrow & \forall \phi \in \Phi.\ \phi \{Q\} \\ \bigoplus \Phi \{Q\} & \Leftrightarrow & \exists \phi \in \Phi.\ \phi \{Q\} \end{array}$$

$$\begin{array}{rcl} \phi \Rrightarrow \phi' & \Leftrightarrow & \forall Q.\ \phi \{Q\} \Rightarrow \phi' \{Q\} \\ C \Rrightarrow C' & \Leftrightarrow & \forall \sigma.\ C(\sigma) \Rrightarrow C'(\sigma) \end{array}$$

Outcomes: Sequential composition

$$-;- \;:\; \mathit{Outcomes}(\mathcal{S}) \to (\mathcal{S} \to \mathit{Outcomes}(\mathcal{S}')) \to \mathit{Outcomes}(\mathcal{S}')$$

Outcomes: Sequential composition

$$-;- \;:\; \mathit{Outcomes}(\mathcal{S}) \to (\mathcal{S} \to \mathit{Outcomes}(\mathcal{S}')) \to \mathit{Outcomes}(\mathcal{S}')$$

$$\langle \sigma \rangle; C \quad =$$

Outcomes: Sequential composition

$$-;- \;:\; \mathit{Outcomes}(\mathcal{S}) \to (\mathcal{S} \to \mathit{Outcomes}(\mathcal{S}')) \to \mathit{Outcomes}(\mathcal{S}')$$

$$\langle \sigma \rangle; C \quad = \quad C(\sigma)$$

Outcomes: Sequential composition

$$-;- \;:\; \mathit{Outcomes}(\mathcal{S}) \to (\mathcal{S} \to \mathit{Outcomes}(\mathcal{S}')) \to \mathit{Outcomes}(\mathcal{S}')$$

$$\begin{array}{rcl} \langle\sigma\rangle; C & = & C(\sigma) \\ (\bigotimes \Phi); C & = & \end{array}$$

Outcomes: Sequential composition

$$-;- \;:\; \mathit{Outcomes}(\mathcal{S}) \to (\mathcal{S} \to \mathit{Outcomes}(\mathcal{S}')) \to \mathit{Outcomes}(\mathcal{S}')$$

$$\begin{array}{rcl} \langle\sigma\rangle; C & = & C(\sigma) \\ (\bigotimes \Phi); C & = & \bigotimes\{\phi \in \Phi.\ (\phi; C)\} \end{array}$$

Outcomes: Sequential composition

$$-;- \;:\; \mathit{Outcomes}(\mathcal{S}) \to (\mathcal{S} \to \mathit{Outcomes}(\mathcal{S}')) \to \mathit{Outcomes}(\mathcal{S}')$$

$$\begin{array}{rcl} \langle\sigma\rangle; C & = & C(\sigma) \\ (\bigotimes \Phi); C & = & \bigotimes\{\phi \in \Phi.\ (\phi; C)\} \\ (\bigoplus \Phi); C & = & \end{array}$$

Outcomes: Sequential composition

$$-;- \;:\; \mathit{Outcomes}(\mathcal{S}) \to (\mathcal{S} \to \mathit{Outcomes}(\mathcal{S}')) \to \mathit{Outcomes}(\mathcal{S}')$$

$$\begin{array}{rcl} \langle\sigma\rangle; C & = & C(\sigma) \\ (\bigotimes \Phi); C & = & \bigotimes\{\phi \in \Phi.\ (\phi; C)\} \\ (\bigoplus \Phi); C & = & \bigoplus\{\phi \in \Phi.\ (\phi; C)\} \end{array}$$

Outcomes: Sequential composition

$$-;- \;:\; \mathit{Outcomes}(\mathcal{S}) \to (\mathcal{S} \to \mathit{Outcomes}(\mathcal{S}')) \to \mathit{Outcomes}(\mathcal{S}')$$

$$\begin{array}{rcl} \langle\sigma\rangle; C & = & C(\sigma) \\ (\bigotimes \Phi); C & = & \bigotimes\{\phi \in \Phi.\ (\phi; C)\} \\ (\bigoplus \Phi); C & = & \bigoplus\{\phi \in \Phi.\ (\phi; C)\} \end{array}$$

Lemma (Satisfaction)

Outcomes: Sequential composition

$$-;- \;:\; \mathit{Outcomes}(\mathcal{S}) \to (\mathcal{S} \to \mathit{Outcomes}(\mathcal{S}')) \to \mathit{Outcomes}(\mathcal{S}')$$

$$\begin{array}{rcl} \langle\sigma\rangle; C & = & C(\sigma) \\ (\bigotimes \Phi); C & = & \bigotimes\{\phi \in \Phi.\ (\phi; C)\} \\ (\bigoplus \Phi); C & = & \bigoplus\{\phi \in \Phi.\ (\phi; C)\} \end{array}$$

Lemma (Satisfaction)

We have $\phi; C\ \{Q\} \Leftrightarrow \phi\ \{\sigma \mid C(\sigma)\ \{Q\}\}$.

Mutators: Sequential Composition

$$C; C' =$$

Mutators: Sequential Composition

$$C; C' = \lambda\sigma.\ C(\sigma); C'$$

Mutators: Sequential Composition

$$C; C' = \lambda\sigma.\ C(\sigma); C'$$

Lemma (Associativity)

Lemma (Monotonicity)

Mutators: Sequential Composition

$$C; C' = \lambda\sigma.\ C(\sigma); C'$$

Lemma (Associativity)

We have $(C; C'); C'' = C; (C'; C'')$.

Lemma (Monotonicity)

Mutators: Sequential Composition

$$C; C' = \lambda\sigma.\ C(\sigma); C'$$

Lemma (Associativity)

We have $(C; C'); C'' = C; (C'; C'')$.

Lemma (Monotonicity)

If $C_1 \Rrightarrow C_1'$ *and* $C_2 \Rrightarrow C_2'$ *then* $C_1; C_2 \Rrightarrow C_1'; C_2'$.

Outcomes: Sequential composition (with Answers)

$$x \leftarrow \cdot; \cdot(x) \;\; : \;\; O(\mathcal{S}, \mathcal{A}) \rightarrow (\mathcal{A} \rightarrow \mathcal{S} \rightarrow O(\mathcal{S}', \mathcal{B})) \rightarrow O(\mathcal{S}', \mathcal{B})$$

Outcomes: Sequential composition (with Answers)

$$x \leftarrow \cdot; \cdot(x) \;:\; O(\mathcal{S}, \mathcal{A}) \to (\mathcal{A} \to \mathcal{S} \to O(\mathcal{S}', \mathcal{B})) \to O(\mathcal{S}', \mathcal{B})$$

$$x \leftarrow \langle \sigma, a \rangle; C(x) \quad =$$

Outcomes: Sequential composition (with Answers)

$$x \leftarrow \cdot; \cdot(x) \;:\; O(\mathcal{S}, \mathcal{A}) \to (\mathcal{A} \to \mathcal{S} \to O(\mathcal{S}', \mathcal{B})) \to O(\mathcal{S}', \mathcal{B})$$

$$x \leftarrow \langle \sigma, a \rangle; C(x) \quad = \quad C(a)(\sigma)$$

Outcomes: Sequential composition (with Answers)

$$x \leftarrow \cdot; \cdot(x) \;:\; O(\mathcal{S}, \mathcal{A}) \to (\mathcal{A} \to \mathcal{S} \to O(\mathcal{S}', \mathcal{B})) \to O(\mathcal{S}', \mathcal{B})$$

$$\begin{array}{rcl} x \leftarrow \langle \sigma, a \rangle; C(x) & = & C(a)(\sigma) \\ x \leftarrow (\bigotimes \Phi); C(x) & = & \end{array}$$

Outcomes: Sequential composition (with Answers)

$$x \leftarrow \cdot; \cdot(x) \;:\; O(\mathcal{S}, \mathcal{A}) \to (\mathcal{A} \to \mathcal{S} \to O(\mathcal{S}', \mathcal{B})) \to O(\mathcal{S}', \mathcal{B})$$

$$\begin{array}{rcl} x \leftarrow \langle \sigma, a \rangle; C(x) & = & C(a)(\sigma) \\ x \leftarrow (\bigotimes \Phi); C(x) & = & \bigotimes \{\phi \in \Phi.\ (x \leftarrow \phi; C(x))\} \end{array}$$

Outcomes: Sequential composition (with Answers)

$$x \leftarrow \cdot; \cdot(x) \;:\; O(\mathcal{S}, \mathcal{A}) \to (\mathcal{A} \to \mathcal{S} \to O(\mathcal{S}', \mathcal{B})) \to O(\mathcal{S}', \mathcal{B})$$

$$\begin{array}{rcl} x \leftarrow \langle \sigma, a \rangle; C(x) & = & C(a)(\sigma) \\ x \leftarrow (\bigotimes \Phi); C(x) & = & \bigotimes \{\phi \in \Phi.\ (x \leftarrow \phi; C(x))\} \\ x \leftarrow (\bigoplus \Phi); C(x) & = & \end{array}$$

Outcomes: Sequential composition (with Answers)

$$x \leftarrow \cdot; \cdot(x) \;:\; O(\mathcal{S}, \mathcal{A}) \to (\mathcal{A} \to \mathcal{S} \to O(\mathcal{S}', \mathcal{B})) \to O(\mathcal{S}', \mathcal{B})$$

$$\begin{array}{rcl} x \leftarrow \langle \sigma, a \rangle; C(x) & = & C(a)(\sigma) \\ x \leftarrow (\bigotimes \Phi); C(x) & = & \bigotimes \{\phi \in \Phi.\ (x \leftarrow \phi; C(x))\} \\ x \leftarrow (\bigoplus \Phi); C(x) & = & \bigoplus \{\phi \in \Phi.\ (x \leftarrow \phi; C(x))\} \end{array}$$

Outcomes: Sequential composition (with Answers)

$$x \leftarrow \cdot; \cdot(x) \;:\; O(\mathcal{S}, \mathcal{A}) \to (\mathcal{A} \to \mathcal{S} \to O(\mathcal{S}', \mathcal{B})) \to O(\mathcal{S}', \mathcal{B})$$

$$\begin{array}{rcl} x \leftarrow \langle \sigma, a \rangle; C(x) & = & C(a)(\sigma) \\ x \leftarrow (\bigotimes \Phi); C(x) & = & \bigotimes \{\phi \in \Phi.\ (x \leftarrow \phi; C(x))\} \\ x \leftarrow (\bigoplus \Phi); C(x) & = & \bigoplus \{\phi \in \Phi.\ (x \leftarrow \phi; C(x))\} \end{array}$$

Lemma (Satisfaction)

We have $x \leftarrow \phi; C(x)\ \{Q\} \Leftrightarrow$.

Outcomes: Sequential composition (with Answers)

$$x \leftarrow \cdot; \cdot(x) \;:\; O(\mathcal{S}, \mathcal{A}) \to (\mathcal{A} \to \mathcal{S} \to O(\mathcal{S}', \mathcal{B})) \to O(\mathcal{S}', \mathcal{B})$$

$$\begin{array}{rcl} x \leftarrow \langle \sigma, a \rangle; C(x) & = & C(a)(\sigma) \\ x \leftarrow (\bigotimes \Phi); C(x) & = & \bigotimes \{\phi \in \Phi.\ (x \leftarrow \phi; C(x))\} \\ x \leftarrow (\bigoplus \Phi); C(x) & = & \bigoplus \{\phi \in \Phi.\ (x \leftarrow \phi; C(x))\} \end{array}$$

Lemma (Satisfaction)

We have $x \leftarrow \phi; C(x)\ \{Q\} \Leftrightarrow \phi\ \{(\sigma, a) \mid C(a)(\sigma)\ \{Q\}\}$.

Mutators: Sequential composition (with Answers)

$$x \leftarrow C; C'(x) =$$

Mutators: Sequential composition (with Answers)

$$x \leftarrow C; C'(x) = \lambda\sigma.\ x \leftarrow C(\sigma); C'(x)$$

Mutators: Sequential composition (with Answers)

$$x \leftarrow C; C'(x) = \lambda\sigma.\ x \leftarrow C(\sigma); C'(x)$$

Lemma (Associativity)

$y \leftarrow (x \leftarrow C; C'(x)); C''(y) =$.

Mutators: Sequential composition (with Answers)

$$x \leftarrow C; C'(x) = \lambda\sigma.\ x \leftarrow C(\sigma); C'(x)$$

Lemma (Associativity)

$$y \leftarrow (x \leftarrow C; C'(x)); C''(y) = x \leftarrow C; (y \leftarrow C'(x); C''(y)).$$

Mutators: Sequential composition (with Answers)

$$x \leftarrow C; C'(x) = \lambda\sigma.\ x \leftarrow C(\sigma); C'(x)$$

Lemma (Associativity)

$y \leftarrow (x \leftarrow C; C'(x)); C''(y) = x \leftarrow C; (y \leftarrow C'(x); C''(y))$.

Lemma (Monotonicity)

If $C_1 \Rrightarrow C_1'$ *and* $\forall a.\ C_2(a) \Rrightarrow C_2'(a)$ *then*

Mutators: Sequential composition (with Answers)

$$x \leftarrow C; C'(x) = \lambda\sigma.\ x \leftarrow C(\sigma); C'(x)$$

Lemma (Associativity)

$y \leftarrow (x \leftarrow C; C'(x)); C''(y) = x \leftarrow C; (y \leftarrow C'(x); C''(y))$.

Lemma (Monotonicity)

If $C_1 \Rrightarrow C_1'$ *and* $\forall a.\ C_2(a) \Rrightarrow C_2'(a)$ *then*
$x \leftarrow C_1; C_2(x) \Rrightarrow x \leftarrow C_1'; C_2'(x)$

Outcomes: Notations

$\bigotimes \tilde{C} =$

$\tilde{C} \subseteq \mathit{Mutators}(\mathcal{S}, \mathcal{S}', \mathcal{A})$

Outcomes: Notations

$$\bigotimes \tilde{C} = \lambda\sigma.\ \bigotimes\{C \in \tilde{C}.\ C(\sigma)\} \qquad \tilde{C} \subseteq \mathit{Mutators}(\mathcal{S}, \mathcal{S}', \mathcal{A})$$

Outcomes: Notations

$$\bigotimes \tilde{C} = \lambda\sigma.\ \bigotimes\{C \in \tilde{C}.\ C(\sigma)\} \qquad \tilde{C} \subseteq \mathit{Mutators}(\mathcal{S}, \mathcal{S}', \mathcal{A})$$

$$\bigoplus \tilde{C} = \qquad \tilde{C} \subseteq \mathit{Mutators}(\mathcal{S}, \mathcal{S}', \mathcal{A})$$

Outcomes: Notations

$$\bigotimes \tilde{C} = \lambda\sigma.\ \bigotimes\{C \in \tilde{C}.\ C(\sigma)\} \qquad \tilde{C} \subseteq \mathit{Mutators}(\mathcal{S}, \mathcal{S}', \mathcal{A})$$
$$\bigoplus \tilde{C} = \lambda\sigma.\ \bigoplus\{C \in \tilde{C}.\ C(\sigma)\} \qquad \tilde{C} \subseteq \mathit{Mutators}(\mathcal{S}, \mathcal{S}', \mathcal{A})$$

Outcomes: Notations

$$\bigotimes \tilde{C} = \lambda\sigma.\ \bigotimes\{C \in \tilde{C}.\ C(\sigma)\} \qquad \tilde{C} \subseteq \mathit{Mutators}(\mathcal{S}, \mathcal{S}', \mathcal{A})$$
$$\bigoplus \tilde{C} = \lambda\sigma.\ \bigoplus\{C \in \tilde{C}.\ C(\sigma)\} \qquad \tilde{C} \subseteq \mathit{Mutators}(\mathcal{S}, \mathcal{S}', \mathcal{A})$$

$$\bigotimes i \in I.\ \phi_i =$$

Outcomes: Notations

$$\bigotimes \tilde{C} = \lambda\sigma.\ \bigotimes\{C \in \tilde{C}.\ C(\sigma)\} \qquad \tilde{C} \subseteq \mathit{Mutators}(\mathcal{S}, \mathcal{S}', \mathcal{A})$$
$$\bigoplus \tilde{C} = \lambda\sigma.\ \bigoplus\{C \in \tilde{C}.\ C(\sigma)\} \qquad \tilde{C} \subseteq \mathit{Mutators}(\mathcal{S}, \mathcal{S}', \mathcal{A})$$

$$\bigotimes i \in I.\ \phi_i = \bigotimes\{\phi_i \mid i \in I\}$$

Outcomes: Notations

$$\bigotimes \tilde{C} = \lambda\sigma.\ \bigotimes\{C \in \tilde{C}.\ C(\sigma)\} \qquad \tilde{C} \subseteq \mathit{Mutators}(\mathcal{S}, \mathcal{S}', \mathcal{A})$$
$$\bigoplus \tilde{C} = \lambda\sigma.\ \bigoplus\{C \in \tilde{C}.\ C(\sigma)\} \qquad \tilde{C} \subseteq \mathit{Mutators}(\mathcal{S}, \mathcal{S}', \mathcal{A})$$

$$\bigotimes i \in I.\ \phi_i = \ \bigotimes\{\phi_i \mid i \in I\}$$
$$\bigoplus i \in I.\ \phi_i =$$

Outcomes: Notations

$$\bigotimes \tilde{C} = \lambda\sigma.\ \bigotimes\{C \in \tilde{C}.\ C(\sigma)\} \qquad \tilde{C} \subseteq \mathit{Mutators}(\mathcal{S}, \mathcal{S}', \mathcal{A})$$
$$\bigoplus \tilde{C} = \lambda\sigma.\ \bigoplus\{C \in \tilde{C}.\ C(\sigma)\} \qquad \tilde{C} \subseteq \mathit{Mutators}(\mathcal{S}, \mathcal{S}', \mathcal{A})$$

$$\bigotimes i \in I.\ \phi_i = \bigotimes\{\phi_i \mid i \in I\}$$
$$\bigoplus i \in I.\ \phi_i = \bigoplus\{\phi_i \mid i \in I\}$$

Outcomes: Notations

$$\bigotimes \tilde{C} = \lambda\sigma.\ \bigotimes\{C \in \tilde{C}.\ C(\sigma)\} \qquad \tilde{C} \subseteq \mathit{Mutators}(\mathcal{S}, \mathcal{S}', \mathcal{A})$$
$$\bigoplus \tilde{C} = \lambda\sigma.\ \bigoplus\{C \in \tilde{C}.\ C(\sigma)\} \qquad \tilde{C} \subseteq \mathit{Mutators}(\mathcal{S}, \mathcal{S}', \mathcal{A})$$

$$\bigotimes i \in I.\ \phi_i = \bigotimes\{\phi_i \mid i \in I\}$$
$$\bigoplus i \in I.\ \phi_i = \bigoplus\{\phi_i \mid i \in I\}$$

$$\bigotimes \mathsf{true}.\ \phi =$$

Outcomes: Notations

$$\bigotimes \tilde{C} = \lambda\sigma.\ \bigotimes\{C \in \tilde{C}.\ C(\sigma)\} \qquad \tilde{C} \subseteq \mathit{Mutators}(\mathcal{S}, \mathcal{S}', \mathcal{A})$$
$$\bigoplus \tilde{C} = \lambda\sigma.\ \bigoplus\{C \in \tilde{C}.\ C(\sigma)\} \qquad \tilde{C} \subseteq \mathit{Mutators}(\mathcal{S}, \mathcal{S}', \mathcal{A})$$

$$\bigotimes i \in I.\ \phi_i = \bigotimes\{\phi_i \mid i \in I\}$$
$$\bigoplus i \in I.\ \phi_i = \bigoplus\{\phi_i \mid i \in I\}$$

$$\bigotimes \mathsf{true}.\ \phi = \phi$$

Outcomes: Notations

$$\bigotimes \tilde{C} = \lambda\sigma.\ \bigotimes\{C \in \tilde{C}.\ C(\sigma)\} \qquad \tilde{C} \subseteq \mathit{Mutators}(\mathcal{S}, \mathcal{S}', \mathcal{A})$$
$$\bigoplus \tilde{C} = \lambda\sigma.\ \bigoplus\{C \in \tilde{C}.\ C(\sigma)\} \qquad \tilde{C} \subseteq \mathit{Mutators}(\mathcal{S}, \mathcal{S}', \mathcal{A})$$

$$\bigotimes i \in I.\ \phi_i = \bigotimes\{\phi_i \mid i \in I\}$$
$$\bigoplus i \in I.\ \phi_i = \bigoplus\{\phi_i \mid i \in I\}$$

$$\bigotimes \mathsf{true}.\ \phi = \phi$$
$$\bigoplus \mathsf{true}.\ \phi =$$

Outcomes: Notations

$$\bigotimes \tilde{C} = \lambda\sigma.\ \bigotimes\{C \in \tilde{C}.\ C(\sigma)\} \qquad \tilde{C} \subseteq \mathit{Mutators}(\mathcal{S}, \mathcal{S}', \mathcal{A})$$
$$\bigoplus \tilde{C} = \lambda\sigma.\ \bigoplus\{C \in \tilde{C}.\ C(\sigma)\} \qquad \tilde{C} \subseteq \mathit{Mutators}(\mathcal{S}, \mathcal{S}', \mathcal{A})$$

$$\bigotimes i \in I.\ \phi_i = \bigotimes\{\phi_i \mid i \in I\}$$
$$\bigoplus i \in I.\ \phi_i = \bigoplus\{\phi_i \mid i \in I\}$$

$$\bigotimes \mathsf{true}.\ \phi = \phi$$
$$\bigoplus \mathsf{true}.\ \phi = \phi$$

Outcomes: Notations

$$\bigotimes \tilde{C} = \lambda\sigma.\ \bigotimes\{C \in \tilde{C}.\ C(\sigma)\} \qquad \tilde{C} \subseteq \mathit{Mutators}(\mathcal{S}, \mathcal{S}', \mathcal{A})$$
$$\bigoplus \tilde{C} = \lambda\sigma.\ \bigoplus\{C \in \tilde{C}.\ C(\sigma)\} \qquad \tilde{C} \subseteq \mathit{Mutators}(\mathcal{S}, \mathcal{S}', \mathcal{A})$$

$$\bigotimes i \in I.\ \phi_i = \bigotimes\{\phi_i \mid i \in I\}$$
$$\bigoplus i \in I.\ \phi_i = \bigoplus\{\phi_i \mid i \in I\}$$

$$\bigotimes \text{true}.\ \phi = \phi \qquad \bigotimes \text{false}.\ \phi =$$
$$\bigoplus \text{true}.\ \phi = \phi$$

Outcomes: Notations

$$\bigotimes \tilde{C} = \lambda\sigma.\ \bigotimes\{C \in \tilde{C}.\ C(\sigma)\} \qquad \tilde{C} \subseteq \mathit{Mutators}(\mathcal{S}, \mathcal{S}', \mathcal{A})$$
$$\bigoplus \tilde{C} = \lambda\sigma.\ \bigoplus\{C \in \tilde{C}.\ C(\sigma)\} \qquad \tilde{C} \subseteq \mathit{Mutators}(\mathcal{S}, \mathcal{S}', \mathcal{A})$$

$$\bigotimes i \in I.\ \phi_i = \bigotimes\{\phi_i \mid i \in I\}$$
$$\bigoplus i \in I.\ \phi_i = \bigoplus\{\phi_i \mid i \in I\}$$

$$\bigotimes \mathsf{true}.\ \phi = \phi \qquad \bigotimes \mathsf{false}.\ \phi = \top$$
$$\bigoplus \mathsf{true}.\ \phi = \phi$$

Outcomes: Notations

$$\bigotimes \tilde{C} = \lambda\sigma.\ \bigotimes\{C \in \tilde{C}.\ C(\sigma)\} \qquad \tilde{C} \subseteq \mathit{Mutators}(\mathcal{S}, \mathcal{S}', \mathcal{A})$$
$$\bigoplus \tilde{C} = \lambda\sigma.\ \bigoplus\{C \in \tilde{C}.\ C(\sigma)\} \qquad \tilde{C} \subseteq \mathit{Mutators}(\mathcal{S}, \mathcal{S}', \mathcal{A})$$

$$\bigotimes i \in I.\ \phi_i = \bigotimes\{\phi_i \mid i \in I\}$$
$$\bigoplus i \in I.\ \phi_i = \bigoplus\{\phi_i \mid i \in I\}$$

$$\bigotimes \text{true}.\ \phi = \phi \qquad \bigotimes \text{false}.\ \phi = \top$$
$$\bigoplus \text{true}.\ \phi = \phi \qquad \bigoplus \text{false}.\ \phi =$$

Outcomes: Notations

$$\bigotimes \tilde{C} = \lambda\sigma.\ \bigotimes\{C \in \tilde{C}.\ C(\sigma)\} \qquad \tilde{C} \subseteq \mathit{Mutators}(\mathcal{S}, \mathcal{S}', \mathcal{A})$$
$$\bigoplus \tilde{C} = \lambda\sigma.\ \bigoplus\{C \in \tilde{C}.\ C(\sigma)\} \qquad \tilde{C} \subseteq \mathit{Mutators}(\mathcal{S}, \mathcal{S}', \mathcal{A})$$

$$\bigotimes i \in I.\ \phi_i = \bigotimes\{\phi_i \mid i \in I\}$$
$$\bigoplus i \in I.\ \phi_i = \bigoplus\{\phi_i \mid i \in I\}$$

$$\bigotimes \text{true}.\ \phi = \phi \qquad \bigotimes \text{false}.\ \phi = \top$$
$$\bigoplus \text{true}.\ \phi = \phi \qquad \bigoplus \text{false}.\ \phi = \bot$$

Outcomes: Notations

$$\bigotimes \tilde{C} = \lambda\sigma.\ \bigotimes\{C \in \tilde{C}.\ C(\sigma)\} \qquad \tilde{C} \subseteq \mathit{Mutators}(\mathcal{S}, \mathcal{S}', \mathcal{A})$$
$$\bigoplus \tilde{C} = \lambda\sigma.\ \bigoplus\{C \in \tilde{C}.\ C(\sigma)\} \qquad \tilde{C} \subseteq \mathit{Mutators}(\mathcal{S}, \mathcal{S}', \mathcal{A})$$

$$\bigotimes i \in I.\ \phi_i = \bigotimes\{\phi_i \mid i \in I\}$$
$$\bigoplus i \in I.\ \phi_i = \bigoplus\{\phi_i \mid i \in I\}$$

$$\bigotimes \text{true}.\ \phi = \phi \qquad \bigotimes \text{false}.\ \phi = \top$$
$$\bigoplus \text{true}.\ \phi = \phi \qquad \bigoplus \text{false}.\ \phi = \bot$$

$$\text{yield}\ a =$$

Outcomes: Notations

$$\bigotimes \tilde{C} = \lambda\sigma.\ \bigotimes\{C \in \tilde{C}.\ C(\sigma)\} \qquad \tilde{C} \subseteq \mathit{Mutators}(\mathcal{S}, \mathcal{S}', \mathcal{A})$$
$$\bigoplus \tilde{C} = \lambda\sigma.\ \bigoplus\{C \in \tilde{C}.\ C(\sigma)\} \qquad \tilde{C} \subseteq \mathit{Mutators}(\mathcal{S}, \mathcal{S}', \mathcal{A})$$

$$\bigotimes i \in I.\ \phi_i = \ \bigotimes\{\phi_i \mid i \in I\}$$
$$\bigoplus i \in I.\ \phi_i = \ \bigoplus\{\phi_i \mid i \in I\}$$

$$\bigotimes \mathsf{true}.\ \phi = \ \phi \qquad \bigotimes \mathsf{false}.\ \phi = \ \top$$
$$\bigoplus \mathsf{true}.\ \phi = \ \phi \qquad \bigoplus \mathsf{false}.\ \phi = \ \bot$$

$$\mathsf{yield}\ a = \ \lambda\sigma.\ \langle\sigma, a\rangle$$

Outcomes: Notations

$$\bigotimes \tilde{C} = \lambda\sigma.\ \bigotimes\{C \in \tilde{C}.\ C(\sigma)\} \qquad \tilde{C} \subseteq \mathit{Mutators}(\mathcal{S}, \mathcal{S}', \mathcal{A})$$
$$\bigoplus \tilde{C} = \lambda\sigma.\ \bigoplus\{C \in \tilde{C}.\ C(\sigma)\} \qquad \tilde{C} \subseteq \mathit{Mutators}(\mathcal{S}, \mathcal{S}', \mathcal{A})$$

$$\bigotimes i \in I.\ \phi_i = \ \bigotimes\{\phi_i \mid i \in I\}$$
$$\bigoplus i \in I.\ \phi_i = \ \bigoplus\{\phi_i \mid i \in I\}$$

$$\bigotimes \text{true}.\ \phi = \ \phi \qquad \bigotimes \text{false}.\ \phi = \ \top$$
$$\bigoplus \text{true}.\ \phi = \ \phi \qquad \bigoplus \text{false}.\ \phi = \ \bot$$

$$\text{yield}\ a = \ \lambda\sigma.\ \langle\sigma, a\rangle$$
$$\text{noop} =$$

Outcomes: Notations

$$\bigotimes \tilde{C} = \lambda\sigma.\ \bigotimes\{C \in \tilde{C}.\ C(\sigma)\} \qquad \tilde{C} \subseteq \mathit{Mutators}(\mathcal{S}, \mathcal{S}', \mathcal{A})$$
$$\bigoplus \tilde{C} = \lambda\sigma.\ \bigoplus\{C \in \tilde{C}.\ C(\sigma)\} \qquad \tilde{C} \subseteq \mathit{Mutators}(\mathcal{S}, \mathcal{S}', \mathcal{A})$$

$$\bigotimes i \in I.\ \phi_i = \bigotimes\{\phi_i \mid i \in I\}$$
$$\bigoplus i \in I.\ \phi_i = \bigoplus\{\phi_i \mid i \in I\}$$

$$\bigotimes \text{true}.\ \phi = \phi \qquad \bigotimes \text{false}.\ \phi = \top$$
$$\bigoplus \text{true}.\ \phi = \phi \qquad \bigoplus \text{false}.\ \phi = \bot$$

$$\text{yield}\ a = \lambda\sigma.\ \langle\sigma, a\rangle$$
$$\text{noop} = \text{yield tt}$$

Outcomes: Notations

$$\bigotimes \tilde{C} = \lambda\sigma.\ \bigotimes\{C \in \tilde{C}.\ C(\sigma)\} \qquad \tilde{C} \subseteq \mathit{Mutators}(\mathcal{S}, \mathcal{S}', \mathcal{A})$$
$$\bigoplus \tilde{C} = \lambda\sigma.\ \bigoplus\{C \in \tilde{C}.\ C(\sigma)\} \qquad \tilde{C} \subseteq \mathit{Mutators}(\mathcal{S}, \mathcal{S}', \mathcal{A})$$

$$\bigotimes i \in I.\ \phi_i = \bigotimes\{\phi_i \mid i \in I\}$$
$$\bigoplus i \in I.\ \phi_i = \bigoplus\{\phi_i \mid i \in I\}$$

$$\bigotimes \text{true}.\ \phi = \phi \qquad \bigotimes \text{false}.\ \phi = \top$$
$$\bigoplus \text{true}.\ \phi = \phi \qquad \bigoplus \text{false}.\ \phi = \bot$$

$$\text{yield}\ a = \lambda\sigma.\ \langle\sigma, a\rangle$$
$$\text{noop} = \text{yield tt}$$

$$C;, C' =$$

Outcomes: Notations

$$\bigotimes \tilde{C} = \lambda\sigma.\ \bigotimes\{C \in \tilde{C}.\ C(\sigma)\} \qquad \tilde{C} \subseteq \mathit{Mutators}(\mathcal{S}, \mathcal{S}', \mathcal{A})$$
$$\bigoplus \tilde{C} = \lambda\sigma.\ \bigoplus\{C \in \tilde{C}.\ C(\sigma)\} \qquad \tilde{C} \subseteq \mathit{Mutators}(\mathcal{S}, \mathcal{S}', \mathcal{A})$$

$$\bigotimes i \in I.\ \phi_i = \bigotimes\{\phi_i \mid i \in I\}$$
$$\bigoplus i \in I.\ \phi_i = \bigoplus\{\phi_i \mid i \in I\}$$

$$\bigotimes \text{true}.\ \phi = \phi \qquad \bigotimes \text{false}.\ \phi = \top$$
$$\bigoplus \text{true}.\ \phi = \phi \qquad \bigoplus \text{false}.\ \phi = \bot$$

$$\text{yield}\ a = \lambda\sigma.\ \langle\sigma, a\rangle$$
$$\text{noop} = \text{yield tt}$$

$$C;, C' = x \leftarrow C;\ C';\ \text{yield}\ x$$

Some Auxiliary Definitions

$$\mathsf{dom}(h) =$$

Some Auxiliary Definitions

$$\mathsf{dom}(h) = \{p(\ell) \mid \exists v.\ p(\ell, v) \in h\}$$

Some Auxiliary Definitions

$$\mathsf{dom}(h) = \{p(\ell) \mid \exists v.\ p(\ell, v) \in h\}$$
$$\mathsf{assume}(b) =$$

Some Auxiliary Definitions

$$\mathsf{dom}(h) = \{p(\ell) \mid \exists v.\ p(\ell, v) \in h\}$$
$$\mathsf{assume}(b) = \lambda(s, h).\ \bigotimes [\![b]\!]_s = \mathsf{true}.\ \langle (s, h) \rangle$$

Some Auxiliary Definitions

$$\begin{aligned}\mathsf{dom}(h) &= \{p(\ell) \mid \exists v.\ p(\ell, v) \in h\} \\ \mathsf{assume}(b) &= \lambda(s, h).\ \bigotimes [\![b]\!]_s = \mathsf{true}.\ \langle (s, h) \rangle \\ \mathsf{store} &= \end{aligned}$$

Some Auxiliary Definitions

$$\begin{aligned}\mathsf{dom}(h) &= \{p(\ell) \mid \exists v.\ p(\ell, v) \in h\} \\ \mathsf{assume}(b) &= \lambda(s, h).\ \bigotimes [\![b]\!]_s = \mathsf{true}.\ \langle (s, h) \rangle \\ \mathsf{store} &= \lambda(s, h).\ \langle (s, h), s \rangle\end{aligned}$$

Some Auxiliary Definitions

$$\begin{aligned}
\mathsf{dom}(h) &= \{p(\ell) \mid \exists v.\ p(\ell, v) \in h\} \\
\mathsf{assume}(b) &= \lambda(s, h).\ \bigotimes [\![b]\!]_s = \mathsf{true}.\ \langle (s, h) \rangle \\
\mathsf{store} &= \lambda(s, h).\ \langle (s, h), s \rangle \\
\mathsf{store} := s' &=
\end{aligned}$$

Some Auxiliary Definitions

$$\begin{aligned}
\mathsf{dom}(h) &= \{p(\ell) \mid \exists v.\ p(\ell, v) \in h\} \\
\mathsf{assume}(b) &= \lambda(s, h).\ \bigotimes [\![b]\!]_s = \mathsf{true}.\ \langle (s, h) \rangle \\
\mathsf{store} &= \lambda(s, h).\ \langle (s, h), s \rangle \\
\mathsf{store} := s' &= \lambda(s, h).\ \langle (s', h) \rangle
\end{aligned}$$

Some Auxiliary Definitions

$$\begin{aligned}
\mathsf{dom}(h) &= \{p(\ell) \mid \exists v.\ p(\ell, v) \in h\} \\
\mathsf{assume}(b) &= \lambda(s, h).\ \bigotimes [\![b]\!]_s = \mathsf{true}.\ \langle (s, h) \rangle \\
\mathsf{store} &= \lambda(s, h).\ \langle (s, h), s \rangle \\
\mathsf{store} := s' &= \lambda(s, h).\ \langle (s', h) \rangle \\
\mathsf{with}(s', C) &=
\end{aligned}$$

Some Auxiliary Definitions

$$\begin{aligned}
\mathsf{dom}(h) &= \{p(\ell) \mid \exists v.\ p(\ell, v) \in h\} \\
\mathsf{assume}(b) &= \lambda(s, h).\ \bigotimes [\![b]\!]_s = \mathsf{true}.\ \langle (s, h) \rangle \\
\mathsf{store} &= \lambda(s, h).\ \langle (s, h), s \rangle \\
\mathsf{store} := s' &= \lambda(s, h).\ \langle (s', h) \rangle \\
\mathsf{with}(s', C) &= s \leftarrow \mathsf{store}; \mathsf{store} := s'; \mathsf{C}; , \mathsf{store} := s
\end{aligned}$$

Some Auxiliary Definitions

$$
\begin{aligned}
\mathsf{dom}(h) &= \{p(\ell) \mid \exists v.\ p(\ell, v) \in h\} \\
\mathsf{assume}(b) &= \lambda(s, h).\ \bigotimes [\![b]\!]_s = \mathsf{true}.\ \langle (s, h) \rangle \\
\mathsf{store} &= \lambda(s, h).\ \langle (s, h), s \rangle \\
\mathsf{store} := s' &= \lambda(s, h).\ \langle (s', h) \rangle \\
\mathsf{with}(s', C) &= s \leftarrow \mathsf{store}; \mathsf{store} := s'; \mathsf{C}; , \mathsf{store} := s \\
\mathsf{eval}(e) &=
\end{aligned}
$$

Some Auxiliary Definitions

$$
\begin{aligned}
\mathsf{dom}(h) &= \{p(\ell) \mid \exists v.\ p(\ell, v) \in h\} \\
\mathsf{assume}(b) &= \lambda(s, h).\ \bigotimes [\![b]\!]_s = \mathsf{true}.\ \langle (s, h) \rangle \\
\mathsf{store} &= \lambda(s, h).\ \langle (s, h), s \rangle \\
\mathsf{store} := s' &= \lambda(s, h).\ \langle (s', h) \rangle \\
\mathsf{with}(s', C) &= s \leftarrow \mathsf{store}; \mathsf{store} := s'; \mathsf{C};, \mathsf{store} := s \\
\mathsf{eval}(e) &= \lambda(s, h).\ \langle (s, h), [\![e]\!]_s \rangle
\end{aligned}
$$

Some Auxiliary Definitions

$$
\begin{aligned}
\mathsf{dom}(h) &= \{p(\ell) \mid \exists v.\ p(\ell, v) \in h\} \\
\mathsf{assume}(b) &= \lambda(s, h).\ \bigotimes [\![b]\!]_s = \mathsf{true}.\ \langle (s, h) \rangle \\
\mathsf{store} &= \lambda(s, h).\ \langle (s, h), s \rangle \\
\mathsf{store} := s' &= \lambda(s, h).\ \langle (s', h) \rangle \\
\mathsf{with}(s', C) &= s \leftarrow \mathsf{store}; \mathsf{store} := s'; \mathsf{C}; , \mathsf{store} := s \\
\mathsf{eval}(e) &= \lambda(s, h).\ \langle (s, h), [\![e]\!]_s \rangle \\
x := v &=
\end{aligned}
$$

Some Auxiliary Definitions

$$
\begin{aligned}
\mathsf{dom}(h) &= \{p(\ell) \mid \exists v.\ p(\ell, v) \in h\} \\
\mathsf{assume}(b) &= \lambda(s, h).\ \bigotimes [\![b]\!]_s = \mathsf{true}.\ \langle (s, h) \rangle \\
\mathsf{store} &= \lambda(s, h).\ \langle (s, h), s \rangle \\
\mathsf{store} := s' &= \lambda(s, h).\ \langle (s', h) \rangle \\
\mathsf{with}(s', C) &= s \leftarrow \mathsf{store}; \mathsf{store} := s'; \mathsf{C}; , \mathsf{store} := s \\
\mathsf{eval}(e) &= \lambda(s, h).\ \langle (s, h), [\![e]\!]_s \rangle \\
x := v &= \lambda(s, h).\ \langle (s[x := v], h) \rangle
\end{aligned}
$$

Some Auxiliary Definitions

$$
\begin{aligned}
\mathsf{dom}(h) &= \{p(\ell) \mid \exists v.\ p(\ell, v) \in h\} \\
\mathsf{assume}(b) &= \lambda(s, h).\ \bigotimes [\![b]\!]_s = \mathsf{true}.\ \langle (s, h) \rangle \\
\mathsf{store} &= \lambda(s, h).\ \langle (s, h), s \rangle \\
\mathsf{store} := s' &= \lambda(s, h).\ \langle (s', h) \rangle \\
\mathsf{with}(s', C) &= s \leftarrow \mathsf{store}; \mathsf{store} := s'; \mathsf{C};, \mathsf{store} := s \\
\mathsf{eval}(e) &= \lambda(s, h).\ \langle (s, h), [\![e]\!]_s \rangle \\
x := v &= \lambda(s, h).\ \langle (s[x := v], h) \rangle \\
C^0 &=
\end{aligned}
$$

Some Auxiliary Definitions

$$
\begin{aligned}
\mathsf{dom}(h) &= \{p(\ell) \mid \exists v.\ p(\ell, v) \in h\} \\
\mathsf{assume}(b) &= \lambda(s, h).\ \bigotimes [\![b]\!]_s = \mathsf{true}.\ \langle (s, h) \rangle \\
\mathsf{store} &= \lambda(s, h).\ \langle (s, h), s \rangle \\
\mathsf{store} := s' &= \lambda(s, h).\ \langle (s', h) \rangle \\
\mathsf{with}(s', C) &= s \leftarrow \mathsf{store}; \mathsf{store} := s'; \mathsf{C};, \mathsf{store} := s \\
\mathsf{eval}(e) &= \lambda(s, h).\ \langle (s, h), [\![e]\!]_s \rangle \\
x := v &= \lambda(s, h).\ \langle (s[x := v], h) \rangle \\
C^0 &= \mathsf{noop}
\end{aligned}
$$

Some Auxiliary Definitions

$$
\begin{aligned}
\mathsf{dom}(h) &= \{p(\ell) \mid \exists v.\ p(\ell, v) \in h\} \\
\mathsf{assume}(b) &= \lambda(s, h).\ \bigotimes [\![b]\!]_s = \mathsf{true}.\ \langle (s, h) \rangle \\
\mathsf{store} &= \lambda(s, h).\ \langle (s, h), s \rangle \\
\mathsf{store} := s' &= \lambda(s, h).\ \langle (s', h) \rangle \\
\mathsf{with}(s', C) &= s \leftarrow \mathsf{store}; \mathsf{store} := s'; \mathsf{C};, \mathsf{store} := s \\
\mathsf{eval}(e) &= \lambda(s, h).\ \langle (s, h), [\![e]\!]_s \rangle \\
x := v &= \lambda(s, h).\ \langle (s[x := v], h) \rangle \\
C^0 &= \mathsf{noop} \\
C^{n+1} &=
\end{aligned}
$$

Some Auxiliary Definitions

$$
\begin{aligned}
\mathsf{dom}(h) &= \{p(\ell) \mid \exists v.\ p(\ell, v) \in h\} \\
\mathsf{assume}(b) &= \lambda(s, h).\ \bigotimes [\![b]\!]_s = \mathsf{true}.\ \langle (s, h) \rangle \\
\mathsf{store} &= \lambda(s, h).\ \langle (s, h), s \rangle \\
\mathsf{store} := s' &= \lambda(s, h).\ \langle (s', h) \rangle \\
\mathsf{with}(s', C) &= s \leftarrow \mathsf{store}; \mathsf{store} := s'; \mathsf{C};, \mathsf{store} := s \\
\mathsf{eval}(e) &= \lambda(s, h).\ \langle (s, h), [\![e]\!]_s \rangle \\
x := v &= \lambda(s, h).\ \langle (s[x := v], h) \rangle \\
C^0 &= \mathsf{noop} \\
C^{n+1} &= C; C^n
\end{aligned}
$$

Some Auxiliary Definitions

$$
\begin{aligned}
\mathsf{dom}(h) &= \{p(\ell) \mid \exists v.\ p(\ell, v) \in h\} \\
\mathsf{assume}(b) &= \lambda(s, h).\ \bigotimes [\![b]\!]_s = \mathsf{true}.\ \langle (s, h) \rangle \\
\mathsf{store} &= \lambda(s, h).\ \langle (s, h), s \rangle \\
\mathsf{store} := s' &= \lambda(s, h).\ \langle (s', h) \rangle \\
\mathsf{with}(s', C) &= s \leftarrow \mathsf{store}; \mathsf{store} := s'; \mathsf{C};, \mathsf{store} := s \\
\mathsf{eval}(e) &= \lambda(s, h).\ \langle (s, h), [\![e]\!]_s \rangle \\
x := v &= \lambda(s, h).\ \langle (s[x := v], h) \rangle \\
C^0 &= \mathsf{noop} \\
C^{n+1} &= C; C^n \\
C^* &=
\end{aligned}
$$

Some Auxiliary Definitions

$$
\begin{aligned}
\mathsf{dom}(h) &= \{p(\ell) \mid \exists v.\ p(\ell, v) \in h\} \\
\mathsf{assume}(b) &= \lambda(s, h).\ \bigotimes [\![b]\!]_s = \mathsf{true}.\ \langle (s, h) \rangle \\
\mathsf{store} &= \lambda(s, h).\ \langle (s, h), s \rangle \\
\mathsf{store} := s' &= \lambda(s, h).\ \langle (s', h) \rangle \\
\mathsf{with}(s', C) &= s \leftarrow \mathsf{store}; \mathsf{store} := s'; \mathsf{C};, \mathsf{store} := s \\
\mathsf{eval}(e) &= \lambda(s, h).\ \langle (s, h), [\![e]\!]_s \rangle \\
x := v &= \lambda(s, h).\ \langle (s[x := v], h) \rangle \\
C^0 &= \mathsf{noop} \\
C^{n+1} &= C; C^n \\
C^* &= \bigotimes n \in \mathbb{N}.\ C^n
\end{aligned}
$$

Some Auxiliary Definitions

$$
\begin{aligned}
\mathrm{dom}(h) &= \{p(\ell) \mid \exists v.\ p(\ell, v) \in h\} \\
\mathrm{assume}(b) &= \lambda(s, h).\ \bigotimes [\![b]\!]_s = \mathrm{true}.\ \langle (s, h) \rangle \\
\mathrm{store} &= \lambda(s, h).\ \langle (s, h), s \rangle \\
\mathrm{store} := s' &= \lambda(s, h).\ \langle (s', h) \rangle \\
\mathrm{with}(s', C) &= s \leftarrow \mathrm{store}; \mathrm{store} := s'; \mathrm{C};, \mathrm{store} := s \\
\mathrm{eval}(e) &= \lambda(s, h).\ \langle (s, h), [\![e]\!]_s \rangle \\
x := v &= \lambda(s, h).\ \langle (s[x := v], h) \rangle \\
C^0 &= \mathrm{noop} \\
C^{n+1} &= C; C^n \\
C^* &= \bigotimes n \in \mathbb{N}.\ C^n \\
\mathrm{cconsume}(h') &=
\end{aligned}
$$

Some Auxiliary Definitions

$$
\begin{aligned}
\mathrm{dom}(h) &= \{p(\ell) \mid \exists v.\ p(\ell, v) \in h\} \\
\mathrm{assume}(b) &= \lambda(s,h).\ \bigotimes [\![b]\!]_s = \mathrm{true}.\ \langle (s,h) \rangle \\
\mathrm{store} &= \lambda(s,h).\ \langle (s,h), s \rangle \\
\mathrm{store} := s' &= \lambda(s,h).\ \langle (s',h) \rangle \\
\mathrm{with}(s', C) &= s \leftarrow \mathrm{store}; \mathrm{store} := s'; \mathrm{C};, \mathrm{store} := s \\
\mathrm{eval}(e) &= \lambda(s,h).\ \langle (s,h), [\![e]\!]_s \rangle \\
x := v &= \lambda(s,h).\ \langle (s[x := v], h) \rangle \\
C^0 &= \mathrm{noop} \\
C^{n+1} &= C; C^n \\
C^* &= \bigotimes n \in \mathbb{N}.\ C^n \\
\mathrm{cconsume}(h') &= \lambda(s,h).\ \bigoplus h' \leq h.\ \langle (s, h - h') \rangle
\end{aligned}
$$

Some Auxiliary Definitions

$$
\begin{aligned}
\mathrm{dom}(h) &= \{p(\ell) \mid \exists v.\ p(\ell, v) \in h\} \\
\mathrm{assume}(b) &= \lambda(s,h).\ \bigotimes [\![b]\!]_s = \mathrm{true}.\ \langle (s,h) \rangle \\
\mathrm{store} &= \lambda(s,h).\ \langle (s,h), s \rangle \\
\mathrm{store} := s' &= \lambda(s,h).\ \langle (s',h) \rangle \\
\mathrm{with}(s', C) &= s \leftarrow \mathrm{store}; \mathrm{store} := s'; \mathrm{C};, \mathrm{store} := s \\
\mathrm{eval}(e) &= \lambda(s,h).\ \langle (s,h), [\![e]\!]_s \rangle \\
x := v &= \lambda(s,h).\ \langle (s[x := v], h) \rangle \\
C^0 &= \mathrm{noop} \\
C^{n+1} &= C; C^n \\
C^* &= \bigotimes n \in \mathbb{N}.\ C^n \\
\mathrm{cconsume}(h') &= \lambda(s,h).\ \bigoplus h' \leq h.\ \langle (s, h - h') \rangle \\
\mathrm{cproduce}(h') &=
\end{aligned}
$$

Some Auxiliary Definitions

$$
\begin{aligned}
\mathsf{dom}(h) &= \{p(\ell) \mid \exists v.\ p(\ell, v) \in h\} \\
\mathsf{assume}(b) &= \lambda(s, h).\ \bigotimes [\![b]\!]_s = \mathsf{true}.\ \langle (s, h) \rangle \\
\mathsf{store} &= \lambda(s, h).\ \langle (s, h), s \rangle \\
\mathsf{store} := s' &= \lambda(s, h).\ \langle (s', h) \rangle \\
\mathsf{with}(s', C) &= s \leftarrow \mathsf{store}; \mathsf{store} := s'; \mathsf{C}; , \mathsf{store} := s \\
\mathsf{eval}(e) &= \lambda(s, h).\ \langle (s, h), [\![e]\!]_s \rangle \\
x := v &= \lambda(s, h).\ \langle (s[x := v], h) \rangle \\
C^0 &= \mathsf{noop} \\
C^{n+1} &= C; C^n \\
C^* &= \bigotimes n \in \mathbb{N}.\ C^n \\
\mathsf{cconsume}(h') &= \lambda(s, h).\ \bigoplus h' \leq h.\ \langle (s, h - h') \rangle \\
\mathsf{cproduce}(h') &= \lambda(s, h).\ \bigotimes \mathsf{dom}(h) \cap \mathsf{dom}(h') = \emptyset.\ \langle (s, h \uplus h') \rangle
\end{aligned}
$$

Concrete Execution of Commands

$\text{exec}_0(c) =$

Concrete Execution of Commands

$$\mathsf{exec}_0(c) = \top$$

Concrete Execution of Commands

$$\mathsf{exec}_0(c) = \top$$

$$\mathsf{exec}_{n+1}(x := e) =$$

Concrete Execution of Commands

$$\mathsf{exec}_0(c) = \top$$

$$\mathsf{exec}_{n+1}(x := e) = v \leftarrow \mathsf{eval}(e); x := v$$

Concrete Execution of Commands

$$\mathsf{exec}_0(c) = \top$$

$$\mathsf{exec}_{n+1}(x := e) = v \leftarrow \mathsf{eval}(e); x := v$$

$$\mathsf{exec}_{n+1}(c; c') =$$

Concrete Execution of Commands

$$\mathsf{exec}_0(c) = \top$$

$$\mathsf{exec}_{n+1}(x := e) = v \leftarrow \mathsf{eval}(e); x := v$$

$$\mathsf{exec}_{n+1}(c; c') = \mathsf{exec}_n(c); \mathsf{exec}_n(c')$$

Concrete Execution of Commands

$$\mathsf{exec}_0(c) = \top$$

$$\mathsf{exec}_{n+1}(x := e) = v \leftarrow \mathsf{eval}(e); x := v$$

$$\mathsf{exec}_{n+1}(c; c') = \mathsf{exec}_n(c); \mathsf{exec}_n(c')$$

$$\mathsf{exec}_{n+1}(\textbf{if } b \textbf{ then } c \textbf{ else } c') =$$

Concrete Execution of Commands

$$\mathsf{exec}_0(c) = \top$$

$$\mathsf{exec}_{n+1}(x := e) = v \leftarrow \mathsf{eval}(e); x := v$$

$$\mathsf{exec}_{n+1}(c; c') = \mathsf{exec}_n(c); \mathsf{exec}_n(c')$$

$$\mathsf{exec}_{n+1}(\textbf{if } b \textbf{ then } c \textbf{ else } c') = \\ \mathsf{assume}(b); \mathsf{exec}_n(c) \otimes \mathsf{assume}(\neg b); \mathsf{exec}_n(c')$$

Concrete Execution of Commands

$$\mathrm{exec}_0(c) = \top$$

$$\mathrm{exec}_{n+1}(x := e) = v \leftarrow \mathrm{eval}(e); x := v$$

$$\mathrm{exec}_{n+1}(c; c') = \mathrm{exec}_n(c); \mathrm{exec}_n(c')$$

$$\mathrm{exec}_{n+1}(\textbf{if } b \textbf{ then } c \textbf{ else } c') = \\ \quad \mathrm{assume}(b); \mathrm{exec}_n(c) \otimes \mathrm{assume}(\neg b); \mathrm{exec}_n(c')$$

$$\mathrm{exec}_{n+1}(\textbf{while } b \textbf{ do } c) =$$

Concrete Execution of Commands

$$\mathrm{exec}_0(c) = \top$$

$$\mathrm{exec}_{n+1}(x := e) = v \leftarrow \mathrm{eval}(e); x := v$$

$$\mathrm{exec}_{n+1}(c; c') = \mathrm{exec}_n(c); \mathrm{exec}_n(c')$$

$$\mathrm{exec}_{n+1}(\textbf{if } b \textbf{ then } c \textbf{ else } c') = \\ \quad \mathrm{assume}(b); \mathrm{exec}_n(c) \otimes \mathrm{assume}(\neg b); \mathrm{exec}_n(c')$$

$$\mathrm{exec}_{n+1}(\textbf{while } b \textbf{ do } c) = \\ \quad (\mathrm{assume}(b); \mathrm{exec}_n(c))^*; \mathrm{assume}(\neg b)$$

Concrete Execution of Commands

$$\mathrm{exec}_0(c) = \top$$

$$\mathrm{exec}_{n+1}(x := e) = v \leftarrow \mathrm{eval}(e); x := v$$

$$\mathrm{exec}_{n+1}(c; c') = \mathrm{exec}_n(c); \mathrm{exec}_n(c')$$

$$\mathrm{exec}_{n+1}(\textbf{if } b \textbf{ then } c \textbf{ else } c') = \\ \quad \mathrm{assume}(b); \mathrm{exec}_n(c) \otimes \mathrm{assume}(\neg b); \mathrm{exec}_n(c')$$

$$\mathrm{exec}_{n+1}(\textbf{while } b \textbf{ do } c) = \\ \quad (\mathrm{assume}(b); \mathrm{exec}_n(c))^*; \mathrm{assume}(\neg b)$$

$$\mathrm{exec}_{n+1}(r(\overline{e})) =$$

Concrete Execution of Commands

$$\mathrm{exec}_0(c) = \top$$

$$\mathrm{exec}_{n+1}(x := e) = v \leftarrow \mathrm{eval}(e); x := v$$

$$\mathrm{exec}_{n+1}(c; c') = \mathrm{exec}_n(c); \mathrm{exec}_n(c')$$

$$\mathrm{exec}_{n+1}(\textbf{if } b \textbf{ then } c \textbf{ else } c') = \\ \mathrm{assume}(b); \mathrm{exec}_n(c) \otimes \mathrm{assume}(\neg b); \mathrm{exec}_n(c')$$

$$\mathrm{exec}_{n+1}(\textbf{while } b \textbf{ do } c) = \\ (\mathrm{assume}(b); \mathrm{exec}_n(c))^*; \mathrm{assume}(\neg b)$$

$$\mathrm{exec}_{n+1}(r(\overline{e})) = \overline{v} \leftarrow \mathrm{eval}(\overline{e}); \mathrm{with}(\mathbf{0}[\overline{x} := \overline{v}], \mathrm{exec}_n(c)) \\ \text{where } \textbf{routine } r(\overline{x}) = c$$

Concrete Execution of Commands

$$\mathrm{exec}_{n+1}(x := \mathbf{malloc}(n)) =$$

Concrete Execution of Commands

$$\begin{aligned}&\mathsf{exec}_{n+1}(x := \mathbf{malloc}(n)) = \\ &\quad \bigotimes \ell, v_1, \ldots, v_n \in \mathbb{Z}. \\ &\qquad \mathsf{cproduce}(\mathsf{mb}(\ell, n), \ell \mapsto v_1, \ldots, \ell + n - 1 \mapsto v_n); x := \ell\end{aligned}$$

Concrete Execution of Commands

$$\mathsf{exec}_{n+1}(x := \mathbf{malloc}(n)) =$$
$$\bigotimes \ell, v_1, \ldots, v_n \in \mathbb{Z}.$$
$$\mathsf{cproduce}(\mathsf{mb}(\ell, n), \ell \mapsto v_1, \ldots, \ell + n - 1 \mapsto v_n); x := \ell$$

$$\mathsf{exec}_{n+1}(x := [e]) =$$

Concrete Execution of Commands

$$\mathsf{exec}_{n+1}(x := \mathbf{malloc}(n)) =$$
$$\bigotimes \ell, v_1, \ldots, v_n \in \mathbb{Z}.$$
$$\mathsf{cproduce}(\mathsf{mb}(\ell, n), \ell \mapsto v_1, \ldots, \ell + n - 1 \mapsto v_n); x := \ell$$

$$\mathsf{exec}_{n+1}(x := [e]) = \ell \leftarrow \mathsf{eval}(e);$$
$$\bigoplus v.\ \mathsf{cconsume}(\ell \mapsto v); \mathsf{cproduce}(\ell \mapsto v); x := v$$

Concrete Execution of Commands

$$\mathsf{exec}_{n+1}(x := \mathbf{malloc}(n)) =$$
$$\quad \bigotimes \ell, v_1, \ldots, v_n \in \mathbb{Z}.$$
$$\quad\quad \mathsf{cproduce}(\mathsf{mb}(\ell, n), \ell \mapsto v_1, \ldots, \ell + n - 1 \mapsto v_n); x := \ell$$

$$\mathsf{exec}_{n+1}(x := [e]) = \ell \leftarrow \mathsf{eval}(e);$$
$$\quad \bigoplus v.\ \mathsf{cconsume}(\ell \mapsto v); \mathsf{cproduce}(\ell \mapsto v); x := v$$

$$\mathsf{exec}_{n+1}([e] := e') =$$

Concrete Execution of Commands

$$\mathsf{exec}_{n+1}(x := \mathbf{malloc}(n)) =$$
$$\bigotimes \ell, v_1, \ldots, v_n \in \mathbb{Z}.$$
$$\mathsf{cproduce}(\mathsf{mb}(\ell, n), \ell \mapsto v_1, \ldots, \ell + n - 1 \mapsto v_n); x := \ell$$

$$\mathsf{exec}_{n+1}(x := [e]) = \ell \leftarrow \mathsf{eval}(e);$$
$$\bigoplus v.\ \mathsf{cconsume}(\ell \mapsto v); \mathsf{cproduce}(\ell \mapsto v); x := v$$

$$\mathsf{exec}_{n+1}([e] := e') = \ell \leftarrow \mathsf{eval}(e); v \leftarrow \mathsf{eval}(e');$$
$$\bigoplus v_0.\ \mathsf{cconsume}(\ell \mapsto v_0); \mathsf{cproduce}(\ell \mapsto v)$$

Concrete Execution of Commands

$$\mathsf{exec}_{n+1}(x := \mathbf{malloc}(n)) =$$
$$\quad \bigotimes \ell, v_1, \ldots, v_n \in \mathbb{Z}.$$
$$\quad\quad \mathsf{cproduce}(\mathsf{mb}(\ell, n), \ell \mapsto v_1, \ldots, \ell + n - 1 \mapsto v_n); x := \ell$$

$$\mathsf{exec}_{n+1}(x := [e]) = \ell \leftarrow \mathsf{eval}(e);$$
$$\quad \bigoplus v.\ \mathsf{cconsume}(\ell \mapsto v); \mathsf{cproduce}(\ell \mapsto v); x := v$$

$$\mathsf{exec}_{n+1}([e] := e') = \ell \leftarrow \mathsf{eval}(e); v \leftarrow \mathsf{eval}(e');$$
$$\quad \bigoplus v_0.\ \mathsf{cconsume}(\ell \mapsto v_0); \mathsf{cproduce}(\ell \mapsto v)$$

$$\mathsf{exec}_{n+1}(\mathbf{free}(e)) =$$

Concrete Execution of Commands

$$\mathsf{exec}_{n+1}(x := \mathbf{malloc}(n)) =$$
$$\quad \bigotimes \ell, v_1, \ldots, v_n \in \mathbb{Z}.$$
$$\quad\quad \mathsf{cproduce}(\mathsf{mb}(\ell, n), \ell \mapsto v_1, \ldots, \ell + n - 1 \mapsto v_n); x := \ell$$

$$\mathsf{exec}_{n+1}(x := [e]) = \ell \leftarrow \mathsf{eval}(e);$$
$$\quad \bigoplus v.\ \mathsf{cconsume}(\ell \mapsto v); \mathsf{cproduce}(\ell \mapsto v); x := v$$

$$\mathsf{exec}_{n+1}([e] := e') = \ell \leftarrow \mathsf{eval}(e); v \leftarrow \mathsf{eval}(e');$$
$$\quad \bigoplus v_0.\ \mathsf{cconsume}(\ell \mapsto v_0); \mathsf{cproduce}(\ell \mapsto v)$$

$$\mathsf{exec}_{n+1}(\mathbf{free}(e)) = \ell \leftarrow \mathsf{eval}(e);$$
$$\quad \bigoplus N \in \mathbb{N}, v_1, \ldots, v_N \in \mathbb{Z}.$$
$$\quad\quad \mathsf{cconsume}(\mathsf{mb}(\ell, N), \ell \mapsto v_1, \ldots, \ell + N - 1 \mapsto v_N)$$

Concrete Execution of Commands

$$\text{exec}(c) =$$

Concrete Execution of Commands

$$\mathrm{exec}(c) = \bigotimes n \in \mathbb{N}.\ \mathrm{exec}_n(c)$$

The Verification Problem

$$\sigma_0 \quad =$$

The Verification Problem

$$\sigma_0 \quad = \quad (\mathbf{0}, \mathbf{0})$$

The Verification Problem

$$\begin{aligned} \sigma_0 &= (\mathbf{0}, \mathbf{0}) \\ a \triangleright f &= \end{aligned}$$

The Verification Problem

$$\begin{aligned} \sigma_0 &= (\mathbf{0}, \mathbf{0}) \\ a \triangleright f &= f(a) \end{aligned}$$

The Verification Problem

$$\begin{aligned} \sigma_0 &= (\mathbf{0}, \mathbf{0}) \\ a \triangleright f &= f(a) \\ \text{safe\_program}(c) &= \end{aligned}$$

The Verification Problem

$$
\begin{aligned}
\sigma_0 &= (\mathbf{0}, \mathbf{0}) \\
a \triangleright f &= f(a) \\
\mathsf{safe\_program}(c) &= \sigma_0 \triangleright \mathsf{exec}(c)\ \{\mathrm{true}\}
\end{aligned}
$$

The Verification Problem

$$
\begin{aligned}
\sigma_0 &= (\mathbf{0}, \mathbf{0}) \\
a \triangleright f &= f(a) \\
\mathsf{safe\_program}(c) &= \sigma_0 \triangleright \mathsf{exec}(c)\ \{\mathrm{true}\}
\end{aligned}
$$

Definition (The Verification Problem)

The Verification Problem

$$\begin{aligned} \sigma_0 &= (\mathbf{0}, \mathbf{0}) \\ a \triangleright f &= f(a) \\ \text{safe\_program}(c) &= \sigma_0 \triangleright \text{exec}(c)\ \{\text{true}\} \end{aligned}$$

Definition (The Verification Problem)

safe_program(c)

Solving the Verification Problem

	exec		

Solving the Verification Problem

	exec		
Recursion			

Solving the Verification Problem

	exec		
Recursion	Yes		

Solving the Verification Problem

	exec		
Recursion Looping	Yes		

Solving the Verification Problem

	exec		
Recursion	Yes		
Looping	Yes		

Solving the Verification Problem

	exec		
Recursion Looping Branching	Yes Yes		

Solving the Verification Problem

	exec		
Recursion	Yes		
Looping	Yes		
Branching	Infinite		

Solving the Verification Problem

	exec		
Recursion Looping Branching	Yes Yes Infinite		
Is Algorithm			

Solving the Verification Problem

	exec		
Recursion	Yes		
Looping	Yes		
Branching	Infinite		
Is Algorithm	No		

Solving the Verification Problem

	exec		
Recursion Looping Branching	Yes Yes Infinite		
Is Algorithm	No		

exec $\longrightarrow$ scexec

Solving the Verification Problem

	exec		
Recursion Looping Branching	Yes Yes Infinite		
Is Algorithm	No		

Assertions

exec ⟶ scexec

Solving the Verification Problem

	exec		
Recursion Looping Branching	Yes Yes Infinite		
Is Algorithm	No		

Assertions
Predicates

exec ⟶ scexec

Solving the Verification Problem

	exec		
Recursion Looping Branching	Yes Yes Infinite		
Is Algorithm	No		

Assertions
Predicates
Routine contracts

exec $\longrightarrow$ scexec

Solving the Verification Problem

	exec		
Recursion Looping Branching	Yes Yes Infinite		
Is Algorithm	No		

exec $\xrightarrow{\text{Assertions, Predicates, Routine contracts, Loop invariants}}$ scexec

Solving the Verification Problem

	exec	scexec	
Recursion	Yes		
Looping	Yes		
Branching	Infinite		
Is Algorithm	No		

exec $\xrightarrow{\text{Assertions, Predicates, Routine contracts, Loop invariants}}$ scexec

Solving the Verification Problem

	exec	scexec	
Recursion	Yes	No	
Looping	Yes		
Branching	Infinite		
Is Algorithm	No		

exec $\xrightarrow{\text{Assertions, Predicates, Routine contracts, Loop invariants}}$ scexec

Solving the Verification Problem

	exec	scexec	
Recursion	Yes	No	
Looping	Yes	No	
Branching	Infinite		
Is Algorithm	No		

Assertions
Predicates
Routine contracts
Loop invariants

exec $\longrightarrow$ scexec

Solving the Verification Problem

	exec	scexec	
Recursion	Yes	No	
Looping	Yes	No	
Branching	Infinite	Infinite	
Is Algorithm	No		

exec $\xrightarrow{\text{Assertions, Predicates, Routine contracts, Loop invariants}}$ scexec

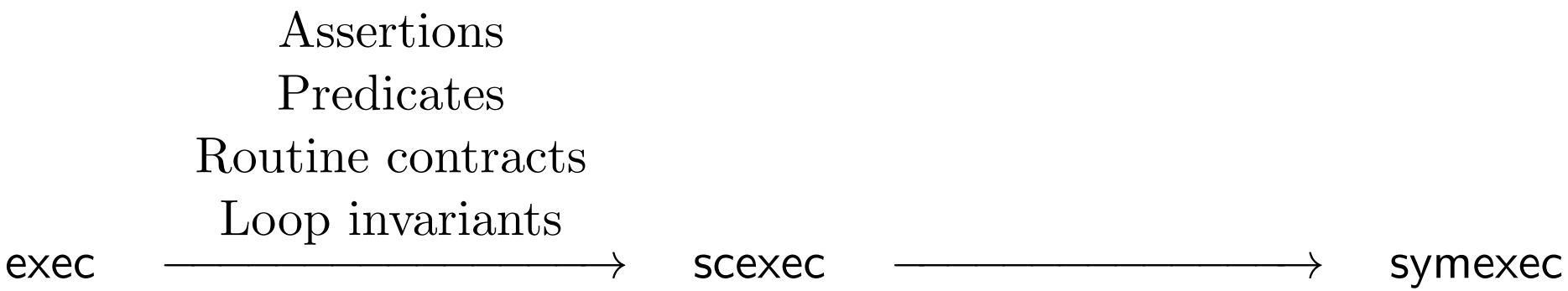
Solving the Verification Problem

	exec	scexec	
Recursion	Yes	No	
Looping	Yes	No	
Branching	Infinite	Infinite	
Is Algorithm	No	No	

exec $\xrightarrow{\text{Assertions, Predicates, Routine contracts, Loop invariants}}$ scexec

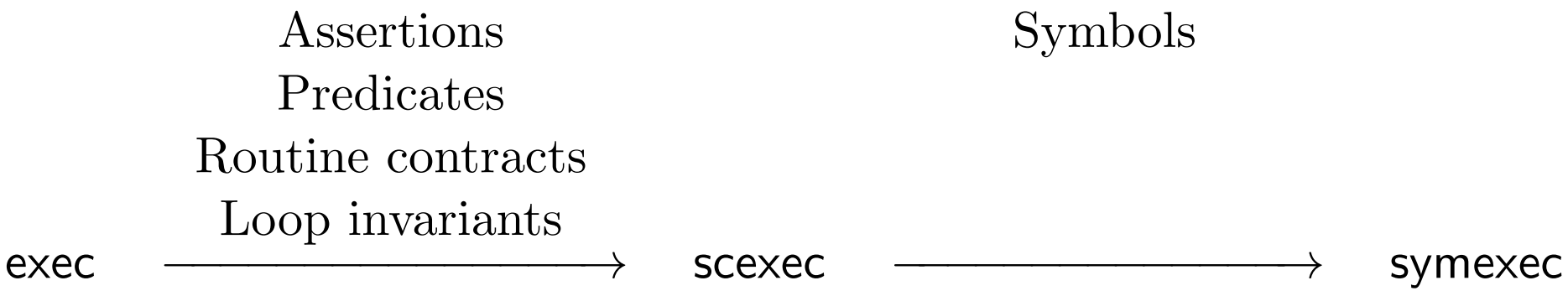
Solving the Verification Problem

	exec	scexec	
Recursion	Yes	No	
Looping	Yes	No	
Branching	Infinite	Infinite	
Is Algorithm	No	No	



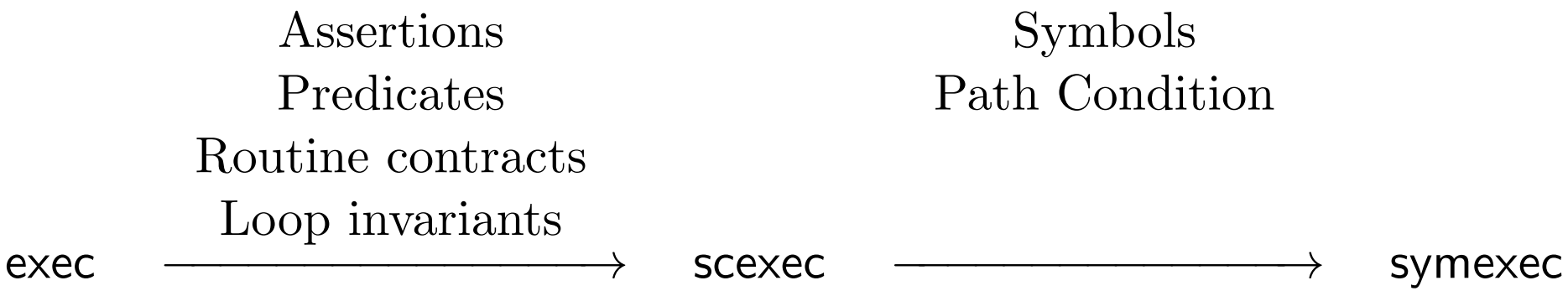
Solving the Verification Problem

	exec	scexec	
Recursion	Yes	No	
Looping	Yes	No	
Branching	Infinite	Infinite	
Is Algorithm	No	No	



Solving the Verification Problem

	exec	scexec	
Recursion	Yes	No	
Looping	Yes	No	
Branching	Infinite	Infinite	
Is Algorithm	No	No	



Solving the Verification Problem

	exec	scexec	
Recursion	Yes	No	
Looping	Yes	No	
Branching	Infinite	Infinite	
Is Algorithm	No	No	

exec $\xrightarrow{\text{Assertions, Predicates, Routine contracts, Loop invariants}}$ scexec $\xrightarrow{\text{Symbols, Path Condition, Fresh Symbols}}$ symexec

Solving the Verification Problem

	exec	scexec	
Recursion	Yes	No	
Looping	Yes	No	
Branching	Infinite	Infinite	
Is Algorithm	No	No	

exec $\xrightarrow{\text{Assertions, Predicates, Routine contracts, Loop invariants}}$ scexec $\xrightarrow{\text{Symbols, Path Condition, Fresh Symbols, Theorem Prover}}$ symexec

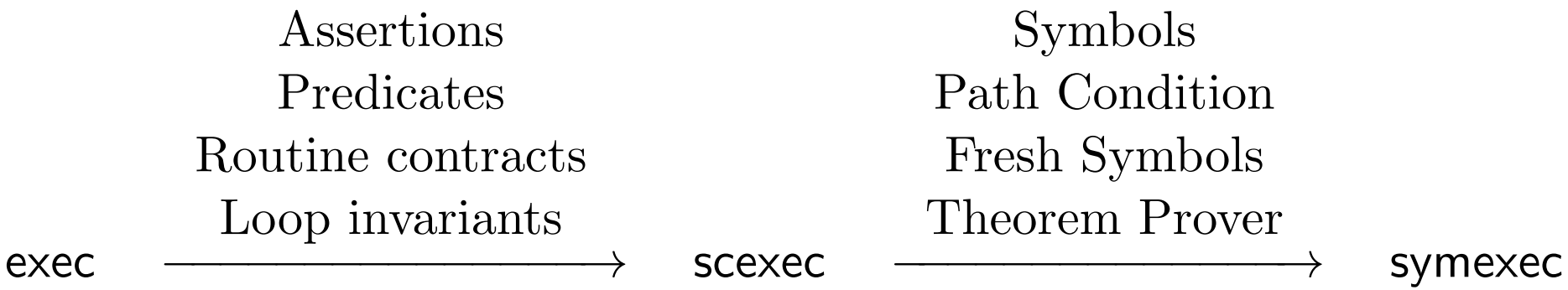
Solving the Verification Problem

	exec	scexec	symexec
Recursion	Yes	No	No
Looping	Yes	No	
Branching	Infinite	Infinite	
Is Algorithm	No	No	

exec $\xrightarrow{\text{Assertions, Predicates, Routine contracts, Loop invariants}}$ scexec $\xrightarrow{\text{Symbols, Path Condition, Fresh Symbols, Theorem Prover}}$ symexec

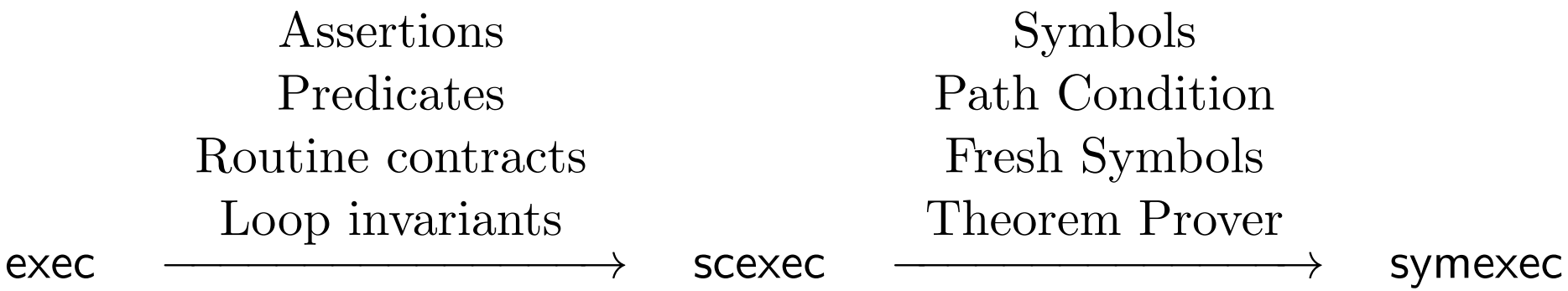
Solving the Verification Problem

	exec	scexec	symexec
Recursion	Yes	No	No
Looping	Yes	No	No
Branching	Infinite	Infinite	
Is Algorithm	No	No	



Solving the Verification Problem

	exec	scexec	symexec
Recursion	Yes	No	No
Looping	Yes	No	No
Branching	Infinite	Infinite	Finite
Is Algorithm	No	No	



Solving the Verification Problem

	exec	scexec	symexec
Recursion	Yes	No	No
Looping	Yes	No	No
Branching	Infinite	Infinite	Finite
Is Algorithm	No	No	Yes

exec $\xrightarrow{\text{Assertions, Predicates, Routine contracts, Loop invariants}}$ scexec $\xrightarrow{\text{Symbols, Path Condition, Fresh Symbols, Theorem Prover}}$ symexec

Solving the Verification Problem

	exec	scexec	symexec
Recursion	Yes	No	No
Looping	Yes	No	No
Branching	Infinite	Infinite	Finite
Is Algorithm	No	No	Yes

exec $\xrightarrow{\text{Assertions, Predicates, Routine contracts, Loop invariants}}$ scexec $\xrightarrow{\text{Symbols, Path Condition, Fresh Symbols, Theorem Prover}}$ symexec

Definition (Soundness)

Solving the Verification Problem

	exec	scexec	symexec
Recursion	Yes	No	No
Looping	Yes	No	No
Branching	Infinite	Infinite	Finite
Is Algorithm	No	No	Yes

exec $\xrightarrow{\text{Assertions, Predicates, Routine contracts, Loop invariants}}$ scexec $\xrightarrow{\text{Symbols, Path Condition, Fresh Symbols, Theorem Prover}}$ symexec

Definition (Soundness)

sym-safe_program(c)

Solving the Verification Problem

	exec	scexec	symexec
Recursion	Yes	No	No
Looping	Yes	No	No
Branching	Infinite	Infinite	Finite
Is Algorithm	No	No	Yes

exec $\xrightarrow{\text{Assertions, Predicates, Routine contracts, Loop invariants}}$ scexec $\xrightarrow{\text{Symbols, Path Condition, Fresh Symbols, Theorem Prover}}$ symexec

Definition (Soundness)

sym-safe_program(c) $\Rightarrow$ sc-safe_program(c)

Solving the Verification Problem

	exec	scexec	symexec
Recursion	Yes	No	No
Looping	Yes	No	No
Branching	Infinite	Infinite	Finite
Is Algorithm	No	No	Yes

exec $\xrightarrow{\text{Assertions, Predicates, Routine contracts, Loop invariants}}$ scexec $\xrightarrow{\text{Symbols, Path Condition, Fresh Symbols, Theorem Prover}}$ symexec

Definition (Soundness)

$$\text{sym-safe\_program}(c) \Rightarrow \text{sc-safe\_program}(c) \Rightarrow \text{safe\_program}(c)$$

Contents

Annotations: Simple Example

routine swap(cell1, cell2)
  req cell1 $\mapsto$?v1 $*$ cell2 $\mapsto$?v2
  ens cell1 $\mapsto$ v2 $*$ cell2 $\mapsto$ v1
  =
    value1 := [cell1];
    value2 := [cell2];
    [cell1] := value2;
    [cell2] := value1

Annotations: Predicates

```
predicate list(l) =
  if l = 0 then 0 = 0 else
    mb(l, 2) ∗ l ↦ ?v ∗ l + 1 ↦ ?n ∗ list(n)

routine range(i, n, result)
  req result ↦ ?dummy
  ens result ↦ ?list ∗ list(list)
  =
    if i = n then head := 0 else (
      head := malloc(2);
      [head] := i;
      range(i + 1, n, head + 1)
    );
    close list(head); [result] := head
```

Annotations

$$
\begin{array}{rl}
 & q \in \mathit{UserDefinedPredicates} \\
p ::= & \mapsto \mid \mathsf{mb} \mid q \\
a ::= & b \mid p(\overline{e}, \overline{?x}) \mid a * a \mid \textbf{if } b \textbf{ then } a \textbf{ else } a \\
\mathit{preddef} ::= & \textbf{predicate } q(\overline{x}) = a \\
c ::= & \cdots \mid \textbf{while } b \textbf{ inv } a \textbf{ do } c \mid \textbf{open } q(\overline{e}) \mid \textbf{close } q(\overline{e}) \\
\mathit{rspec} ::= & \textbf{routine } r(\overline{x}) \textbf{ req } a \textbf{ ens } a
\end{array}
$$

$e \mapsto ?x$ is alternative syntax for $\mapsto(e, ?x)$

Concrete Execution: Example Trace

routine range(i, n, r) =
s:**0**[i:5, n:8, r:41], h:$h_0 \uplus \{\!\{41 \mapsto 77\}\!\}$
if i = n **then** l := 0 **else** (
l := **malloc**(2);
s:**0**[i:5, n:8, r:41, l:50], h:$h_0 \uplus \{\!\{41 \mapsto 77, \mathrm{mb}(50, 2), 50 \mapsto 88, 51 \mapsto 99\}\!\}$
[l] := i; range(i + 1, n, l + 1)

⋮ *(Execution of 3 nested* range *calls)*

s:**0**[i:5, n:8, r:41, l:50], h:$h_0 \uplus \{\!\{41 \mapsto 77, \mathrm{mb}(50, 2), 50 \mapsto 5, 51 \mapsto 60,$
$\mathrm{mb}(60, 2), 60 \mapsto 6, 61 \mapsto 70, \mathrm{mb}(70, 2), 70 \mapsto 7, 71 \mapsto 0\}\!\}$
);
[r] := l
s:**0**[i:5, n:8, r:41, l:50], h:$h_0 \uplus \{\!\{41 \mapsto 50, \mathrm{mb}(50, 2), 50 \mapsto 5, 51 \mapsto 60,$
$\mathrm{mb}(60, 2), 60 \mapsto 6, 61 \mapsto 70, \mathrm{mb}(70, 2), 70 \mapsto 7, 71 \mapsto 0\}\!\}$
where h_0 : $\{\!\{\mathrm{mb}(30), 30 \mapsto 3, 31 \mapsto 40, \mathrm{mb}(40, 2), 40 \mapsto 4\}\!\}$

Semiconcrete Execution: Example Trace

routine range(i, n, r)

req $r \mapsto ?dummy$ **ens** $r \mapsto ?list * list(list)$

s:**0**[i:5, n:8, r:41], h:**0**

produce($r \mapsto ?dummy$)

s:**0**[i:5, n:8, r:41], h:$\{\!\{41 \mapsto 77\}\!\}$

if i = n **then** l := 0 **else** (

l := **malloc**(2);

s:**0**[i:5, n:8, r:41, l:50], h:$\{\!\{41 \mapsto 77, mb(50, 2), 50 \mapsto 88, 51 \mapsto 99\}\!\}$

[l] := i; range(i + 1, n, l + 1)

consume($l+1 \mapsto ?dummy$); *produce*($l+1 \mapsto ?list * list(list)$)

s:**0**[i:5, n:8, r:41, l:50], h:$\{\!\{41 \mapsto 77, mb(50, 2), 50 \mapsto 5, 51 \mapsto 60, list(60)\}\!\}$

); **close** list(l); [r] := l

s:**0**[i:5, n:8, r:41, l:50], h:$\{\!\{41 \mapsto 50, list(50)\}\!\}$

consume($r \mapsto ?list * list(list)$)

s:**0**[i:5, n:8, r:41, l:50], h:**0**

Semiconcrete execution

$$
\begin{aligned}
\mathit{SCStores} &= \mathit{Vars} \to \mathbb{Z} \\
\mathit{SCPredicates} &= \{\mapsto, \mathsf{mb}\} \cup \mathit{UserDefinedPredicates} \\
\mathit{SCChunks} &= \{p(\overline{v}) \mid p \in \mathit{SCPredicates}, \overline{v} \in \mathbb{Z}\} \\
\mathit{SCHeaps} &= \mathit{SCChunks} \to \mathbb{N} \\
\mathit{SCStates} &= \mathit{SCStores} \times \mathit{SCHeaps} \\
\mathit{SCMutators} &= \mathit{SCStates} \to \mathit{Outcomes}(\mathit{SCStates})
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{scexec} &\in \mathit{Commands} \to \mathit{SCMutators} \\
\mathsf{consume} &\in \mathit{Assertions} \to \mathit{SCMutators} \\
\mathsf{produce} &\in \mathit{Assertions} \to \mathit{SCMutators}
\end{aligned}
$$

Some Auxiliary Definitions

$$\begin{aligned}
\mathsf{consume}(h') &= \lambda(s,h).\ \bigoplus h' \leq h.\ \langle (s, h - h') \rangle \\
\mathsf{produce}(h') &= \lambda(s,h).\ \langle (s, h \uplus h') \rangle \\
\mathsf{assert}(b) &= \lambda(s,h).\ \bigoplus [\![b]\!]_s = \mathsf{true}.\ \langle (s,h) \rangle
\end{aligned}$$

Producing Assertions

$$\mathsf{produce}(b) = \mathsf{assume}(b)$$

$$\mathsf{produce}(p(\overline{e}, \overline{?x})) = \\ \quad \overline{v} \leftarrow \mathsf{eval}(\overline{e}); \bigotimes \overline{v}'.\ \mathsf{produce}(p(\overline{v}, \overline{v}')); \overline{x} := \overline{v}'$$

$$\mathsf{produce}(a * a') = \mathsf{produce}(a); \mathsf{produce}(a')$$

$$\mathsf{produce}(\textbf{if } b \textbf{ then } a \textbf{ else } a') = \\ \quad \mathsf{assume}(b); \mathsf{produce}(a) \otimes \mathsf{assume}(\neg b); \mathsf{produce}(a')$$

Consuming Assertions

$$\mathsf{consume}(b) = \mathsf{assert}(b)$$

$$\mathsf{consume}(p(\overline{e}, \overline{?x})) = \\ \quad \overline{v} \leftarrow \mathsf{eval}(\overline{e}); \bigoplus \overline{v}'.\ \mathsf{consume}(p(\overline{v}, \overline{v}')); \overline{x} := \overline{v}'$$

$$\mathsf{consume}(a * a') = \mathsf{consume}(a); \mathsf{consume}(a')$$

$$\mathsf{consume}(\textbf{if}\ b\ \textbf{then}\ a\ \textbf{else}\ a') = \\ \quad \mathsf{assume}(b); \mathsf{consume}(a) \otimes \mathsf{assume}(\neg b); \mathsf{consume}(a')$$

Semiconcrete Execution of Commands

$$\mathrm{scexec}(x := e) = v \leftarrow \mathrm{eval}(e); x := v$$

$$\mathrm{scexec}(c; c') = \mathrm{scexec}(c); \mathrm{scexec}(c')$$

$$\mathrm{scexec}(\textbf{if } b \textbf{ then } a \textbf{ else } a') = \mathrm{assume}(b); \mathrm{scexec}(c) \otimes \mathrm{assume}(\neg b); \mathrm{scexec}(c')$$

$$\mathrm{scexec}(r(\overline{e})) = \overline{v} \leftarrow \mathrm{eval}(\overline{e}); \mathrm{with}(\mathbf{0}[\overline{x} := \overline{v}], \mathrm{consume}(a); \mathrm{produce}(a'))$$
where **routine** $r(\overline{x})$ **req** a **ens** a'

Semiconcrete Execution of Commands

$$\mathsf{havoc}(\overline{x}) = \lambda(s, h).\ \bigotimes \overline{v} \in \mathbb{Z}.\ \langle (s[\overline{x} := \overline{v}], h) \rangle$$
$$\mathsf{leakcheck} = \lambda(s, h).\ \bigoplus h = \mathbf{0}.\ \top$$

$\mathsf{scexec}(\textbf{while}\ b\ \textbf{inv}\ a\ \textbf{do}\ c) =$
$\quad s \leftarrow \mathsf{store}; \mathsf{with}(s, \mathsf{consume}(a));$
$\quad \mathsf{havoc}(\mathsf{targets}(c));$
$\quad ($
$\quad\quad \mathsf{heap} := \mathbf{0};$
$\quad\quad s \leftarrow \mathsf{store}; \mathsf{with}(s, \mathsf{produce}(a));$
$\quad\quad \mathsf{assume}(b); \mathsf{scexec}(c);$
$\quad\quad s \leftarrow \mathsf{store}; \mathsf{with}(s, \mathsf{consume}(a));$
$\quad\quad \mathsf{leakcheck}$
$\quad \otimes$
$\quad\quad s \leftarrow \mathsf{store}; \mathsf{with}(s, \mathsf{produce}(a));$
$\quad\quad \mathsf{assume}(\neg b)$
$\quad)$

Semiconcrete Execution of Commands

$$\begin{aligned}
&\mathsf{scexec}(x := \mathbf{malloc}(n)) = \\
&\quad \bigotimes \ell, v_1, \ldots, v_n \in \mathbb{Z}. \\
&\qquad \mathsf{produce}(\mathsf{mb}(\ell, n), \ell \mapsto v_1, \ldots, \ell + n - 1 \mapsto v_n); x := \ell
\end{aligned}$$

$$\begin{aligned}
&\mathsf{scexec}(x := [e]) = \ell \leftarrow \mathsf{eval}(e); \\
&\quad \bigoplus v.\ \mathsf{consume}(\ell \mapsto v); \mathsf{produce}(\ell \mapsto v); x := v
\end{aligned}$$

$$\begin{aligned}
&\mathsf{scexec}([e] := e') = \ell \leftarrow \mathsf{eval}(e); v \leftarrow \mathsf{eval}(e'); \\
&\quad \bigoplus v_0.\ \mathsf{consume}(\ell \mapsto v_0); \mathsf{produce}(\ell \mapsto v)
\end{aligned}$$

$$\begin{aligned}
&\mathsf{scexec}(\mathbf{free}(e)) = \ell \leftarrow \mathsf{eval}(e); \\
&\quad \bigoplus N \in \mathbb{N}, v_1, \ldots, v_N \in \mathbb{Z}. \\
&\qquad \mathsf{consume}(\mathsf{mb}(\ell, N), \ell \mapsto v_1, \ldots, \ell + N - 1 \mapsto v_N)
\end{aligned}$$

Semiconcrete Execution of Commands

$$\begin{array}{l}\mathsf{scexec}(\textbf{open}\ p(\overline{e})) = \overline{v} \leftarrow \mathsf{eval}(\overline{e});\\ \quad \mathsf{consume}(p(\overline{v})); \mathsf{with}(\mathbf{0}[\overline{x} := \overline{v}], \mathsf{produce}(a))\\ \quad \text{where } \textbf{predicate}\ p(\overline{x}) = a\end{array}$$

$$\begin{array}{l}\mathsf{scexec}(\textbf{close}\ p(\overline{e})) = \overline{v} \leftarrow \mathsf{eval}(\overline{e});\\ \quad \mathsf{with}(\mathbf{0}[\overline{x} := \overline{v}], \mathsf{consume}(a)); \mathsf{produce}(p(\overline{v}))\\ \quad \text{where } \textbf{predicate}\ p(\overline{x}) = a\end{array}$$

Validity of Routines

$$
\begin{array}{l}
\mathsf{valid}(r) = \\
\quad (\mathbf{0}, \mathbf{0}) \triangleright \\
\quad \bigotimes \overline{v}. \\
\quad \mathsf{with}(\mathbf{0}[\overline{x} := \overline{v}], \\
\qquad s' \leftarrow \mathsf{with}(\mathbf{0}[\overline{x} := \overline{v}], \mathsf{produce}(a); \mathsf{store}); \\
\qquad \mathsf{scexec}(c); \\
\qquad \mathsf{with}(s', \mathsf{consume}(a')) \\
\quad); \\
\quad \mathsf{leakcheck} \\
\quad \{\mathrm{true}\} \\
\quad \text{where } \mathbf{routine}\ r(\overline{x})\ \mathbf{req}\ a\ \mathbf{ens}\ a' = c
\end{array}
$$

Semiconcrete Execution: Program Safety

$$\text{sc-safe\_program}(c) \quad = \quad (\forall r.\ \text{valid}(r)) \land \sigma_0 \triangleright \text{scexec}(c)\ \{\text{true}\}$$

Soundness: The Consumption Relation

$$\frac{[\![b]\!]_s = \text{true}}{(s,h) \xrightarrow{b}_{\mathsf{c}} (s,h)} \qquad \frac{h = \{\!\{p([\![\overline{e}]\!]_s, \overline{v})\}\!\} \uplus h'}{(s,h) \xrightarrow{p(\overline{e},?\overline{x})}_{\mathsf{c}} (s[\overline{x} := \overline{v}], h')}$$

$$\frac{(s,h) \xrightarrow{a}_{\mathsf{c}} (s',h') \qquad (s',h') \xrightarrow{a'}_{\mathsf{c}} (s'',h'')}{(s,h) \xrightarrow{a*a'}_{\mathsf{c}} (s'',h'')}$$

$$\frac{[\![b]\!]_s = \text{true} \qquad (s,h) \xrightarrow{a}_{\mathsf{c}} (s',h')}{(s,h) \xrightarrow{\textbf{if } b \textbf{ then } a \textbf{ else } a'}_{\mathsf{c}} (s',h')}$$

$$\frac{[\![b]\!]_s = \text{false} \qquad (s,h) \xrightarrow{a'}_{\mathsf{c}} (s',h')}{(s,h) \xrightarrow{\textbf{if } b \textbf{ then } a \textbf{ else } a'}_{\mathsf{c}} (s',h')}$$

Soundness: The Production Relation

$$\frac{[\![b]\!]_s = \text{true}}{(s,h) \xrightarrow{b}_{\text{p}} (s,h)} \qquad \frac{h' = \{\!|p([\![\overline{e}]\!]_s, \overline{v})|\!\} \uplus h}{(s,h) \xrightarrow{p(\overline{e},?\overline{x})}_{\text{p}} (s[\overline{x} := \overline{v}], h')}$$

$$\frac{(s,h) \xrightarrow{a}_{\text{p}} (s',h') \qquad (s',h') \xrightarrow{a'}_{\text{p}} (s'',h'')}{(s,h) \xrightarrow{a*a'}_{\text{p}} (s'',h'')}$$

$$\frac{[\![b]\!]_s = \text{true} \qquad (s,h) \xrightarrow{a}_{\text{p}} (s',h')}{(s,h) \xrightarrow{\textbf{if } b \textbf{ then } a \textbf{ else } a'}_{\text{p}} (s',h')}$$

$$\frac{[\![b]\!]_s = \text{false} \qquad (s,h) \xrightarrow{a'}_{\text{p}} (s',h')}{(s,h) \xrightarrow{\textbf{if } b \textbf{ then } a \textbf{ else } a'}_{\text{p}} (s',h')}$$

Soundness: Consumption and Production

Lemma (Consumption and Production and the Relations)

$$\mathsf{consume}(a) = \lambda\sigma.\ \bigoplus \sigma', \sigma \xrightarrow{a}_{\mathsf{c}} \sigma'.\ \langle\sigma'\rangle$$
$$\mathsf{produce}(a) = \lambda\sigma.\ \bigotimes \sigma', \sigma \xrightarrow{a}_{\mathsf{p}} \sigma'.\ \langle\sigma'\rangle$$

Lemma (Consumption Locality)

$$(s, h) \xrightarrow{a}_{\mathsf{c}} (s', h') \Rightarrow (s, h \uplus h'') \xrightarrow{a}_{\mathsf{c}} (s', h' \uplus h'')$$

Lemma (Consumption Monotonicity)

$$(s, h) \xrightarrow{a}_{\mathsf{c}} (s', h') \Rightarrow \exists h''.\ h = h' \uplus h'' \wedge (s, h'') \xrightarrow{a}_{\mathsf{c}} (s', \mathbf{0})$$

Lemma (Production after Consumption (Relations))

$$(s, h) \xrightarrow{a}_{\mathsf{c}} (s', \mathbf{0}) \Rightarrow (s, h'') \xrightarrow{a}_{\mathsf{p}} (s', h'' \uplus h)$$

Soundness: Consumption and Production

Lemma (Consumption and Production (with Post-stores))

$s_1 \leftarrow \mathsf{with}(s, \mathsf{consume}(a); \mathsf{store});$
$s_2 \leftarrow \mathsf{with}(s, \mathsf{produce}(a); \mathsf{store});$
$C(s_1, s_2)$
$\Rrightarrow$
$\bigoplus s'.\ C(s', s')$

Lemma (Consumption and Production)

$\mathsf{with}(s, \mathsf{consume}(a)); \mathsf{with}(s, \mathsf{produce}(a)) \Rrightarrow \mathsf{noop}$

Soundness: Locality and Modifies

$$\text{local } C \Leftrightarrow C;, \text{produce}(h) \Rrightarrow \text{produce}(h); C$$

Lemma (Locality)

local scexec(c)

$$s \overset{\overline{x}}{\sim} s' \Leftrightarrow s[\overline{x} := 0] = s'[\overline{x} := 0]$$

$$\text{modified}_{\overline{x}}(s') = \lambda(s, h).\ \bigoplus s \overset{\overline{x}}{\sim} s'.\ \text{noop}$$

$$\text{modifies}_{\overline{x}}\ C \Leftrightarrow \forall s.\ \text{modified}_{\overline{x}}(s); C \Rrightarrow C;, \text{modified}_{\overline{x}}(s)$$

Lemma (Semiconcrete Execution Modifies Targets)

$\text{modifies}_{\text{targets}(c)}$ scexec(c)

Soundness

Definition (Heap refinement)

$$\frac{h_c \triangleleft h \qquad \textbf{predicate } p(\overline{x}) = a \qquad (\mathbf{0}[\overline{x} := \overline{v}], h) \xrightarrow{a}_c (s', \mathbf{0})}{h_c \triangleleft \{\!|p(\overline{v})|\!\}}$$

$$h_c \triangleleft h_c \qquad \frac{h_c \triangleleft h \qquad h'_c \triangleleft h'}{h_c \uplus h'_c \triangleleft h \uplus h'}$$

Lemma (Open, Close)

$$h_c \triangleleft h \uplus \{\!|p(\overline{v})|\!\} \Leftrightarrow \exists s', h', (\mathbf{0}[\overline{x} := \overline{v}], h') \xrightarrow{a}_c (s', \mathbf{0}) \wedge h_c \triangleleft h \uplus h'$$

where **predicate** $p(\overline{x}) = a$

Soundness

$$\kappa = \lambda(s, h).\ \bigotimes h_c \in \mathit{CHeaps}, h_c \triangleleft h.\ \langle (s, h_c) \rangle$$

Lemma (Soundness of Semiconcrete Execution)

If $\forall r.\ \mathsf{valid}(r)$, *then*

$$\mathsf{scexec}(c); \kappa \Rrightarrow \kappa; \mathsf{exec}(c)$$

Proof.

It is sufficient to prove

$$\forall n, c.\ \mathsf{scexec}(c); \kappa \Rrightarrow \kappa; \mathsf{exec}_n(c)$$

By induction on n. The base case is trivial. Assume $\forall c.\ \mathsf{scexec}(c); \kappa \Rrightarrow \kappa; \mathsf{exec}_n(c)$. The goal is $\forall c.\ \mathsf{scexec}(c); \kappa \Rrightarrow \kappa; \mathsf{exec}_{n+1}(c)$. By case analysis on c. □

Soundness: Routine Calls

Proof.

Goal: $\mathsf{scexec}(\mathsf{r}(\overline{e})); \kappa \Rrightarrow \kappa; \mathsf{exec}_{n+1}(r(\overline{e}))$.
S.t.p.: $\mathsf{w}(s, \mathsf{c}(a); \mathsf{p}(a')); \kappa \Rrightarrow \kappa; \mathsf{w}(s, \mathsf{e}_n(c))$.
S.t.p.: $\mathsf{w}(s, \mathsf{c}(a); \mathsf{p}(a')) \Rrightarrow \mathsf{w}(s, \mathsf{sce}(c))$ (by IH).
Note: $\mathsf{noop} \Rrightarrow \mathsf{w}(s, s' \leftarrow \mathsf{ws}(s, \mathsf{p}(a)); \mathsf{sce}(c); \mathsf{w}(s', \mathsf{c}(a')))$
(by $\mathsf{valid}(r)$ and local $\mathsf{scexec}(\cdot)$).

$$
\begin{aligned}
& \mathsf{w}(s, \mathsf{c}(a); \mathsf{p}(a')) \\
\Rrightarrow\ & s_1 \leftarrow \mathsf{ws}(s, \mathsf{c}(a)); \mathsf{w}(s_1, \mathsf{p}(a')) \\
\Rrightarrow\ & s_1 \leftarrow \mathsf{ws}(s, \mathsf{c}(a)); \mathsf{noop}; \mathsf{w}(s_1, \mathsf{p}(a')) \\
\Rrightarrow\ & s_1 \leftarrow \mathsf{ws}(s, \mathsf{c}(a)); \mathsf{w}(s, s' \leftarrow \mathsf{ws}(s, \mathsf{p}(a)); \mathsf{sce}(c); \mathsf{w}(s', \mathsf{c}(a'))); \mathsf{w}(s_1, \mathsf{p}(a')) \\
\Rrightarrow\ & s_1 \leftarrow \mathsf{ws}(s, \mathsf{c}(a)); s_2 \leftarrow \mathsf{ws}(s, \mathsf{p}(a)); \mathsf{w}(s, \mathsf{sce}(c)); \mathsf{w}(s_2, \mathsf{c}(a')); \mathsf{w}(s_1, \mathsf{p}(a')) \\
\Rrightarrow\ & \mathsf{w}(s, \mathsf{sce}(c)); \mathsf{w}(s'', \mathsf{c}(a')); \mathsf{w}(s'', \mathsf{p}(a')) \\
\Rrightarrow\ & \mathsf{w}(s, \mathsf{sce}(c))
\end{aligned}
$$

Soundness: Loops

Proof.

Goal: $\mathsf{scexec}(\textbf{while } b \textbf{ inv } a \textbf{ do } c); \kappa \Rrightarrow \kappa; \mathsf{exec}_{n+1}(\textbf{while } b \textbf{ inv } a \textbf{ do } c)$.

$\mathrm{Stp}\ \mathsf{m}_{\overline{x}}(s); \mathsf{cc}(a); \mathsf{h}(\overline{x}); (\mathsf{clh}; \mathsf{pc}(a); \mathsf{a}(b); \mathsf{sce}(c); \mathsf{cc}(a); \mathsf{lck} \otimes \mathsf{pc}(a); \mathsf{a}(\neg b)); \kappa \Rrightarrow \kappa; (\mathsf{a}(b); \mathsf{e}_n(c))^*; \mathsf{a}(\neg b)$.

$\mathrm{Assume}\ \mathsf{m}_{\overline{x}}(s); \mathsf{h}(\overline{x}) \Rrightarrow \mathsf{pc}(a); \mathsf{a}(b); \mathsf{sce}(c); \mathsf{cc}(a)$.

$\mathrm{Stp}\ \mathsf{m}_{\overline{x}}(s); \mathsf{cc}(a); \mathsf{h}(\overline{x}); \mathsf{pc}(a); \mathsf{a}(\neg b)); \kappa \Rrightarrow \kappa; (\mathsf{a}(b); \mathsf{e}_n(c))^*; \mathsf{a}(\neg b)$.

$\mathrm{Stp}\ \mathsf{m}_{\overline{x}}(s); \mathsf{cc}(a); \mathsf{h}(\overline{x}); \mathsf{pc}(a) \Rrightarrow (\mathsf{a}(b); \mathsf{sce}(c))^*\ (\mathrm{by\ IH})$.

$\mathrm{Stp}\ \mathsf{m}_{\overline{x}}(s); \mathsf{cc}(a); \mathsf{h}(\overline{x}); \mathsf{pc}(a) \Rrightarrow \mathsf{a}(b); \mathsf{sce}(c); \mathsf{m}_{\overline{x}}(s); \mathsf{cc}(a); \mathsf{h}(\overline{x}); \mathsf{pc}(a)$.

$$
\begin{array}{ll}
 & \mathsf{m}_{\overline{x}}(s); \mathsf{cc}(a); \mathsf{h}(\overline{x}); \mathsf{pc}(a) \\
\Rrightarrow & \mathsf{m}_{\overline{x}}(s); \mathsf{cc}(a); \mathsf{m}_{\overline{x}}(s); \mathsf{h}(\overline{x}); \mathsf{h}(\overline{x}); \mathsf{pc}(a) \\
\Rrightarrow & \mathsf{m}_{\overline{x}}(s); \mathsf{cc}(a); \mathsf{pc}(a); \mathsf{a}(b); \mathsf{sce}(c); \mathsf{cc}(a); \mathsf{h}(\overline{x}); \mathsf{pc}(a) \\
\Rrightarrow & \mathsf{m}_{\overline{x}}(s); \mathsf{a}(b); \mathsf{sce}(c); \mathsf{cc}(a); \mathsf{h}(\overline{x}); \mathsf{pc}(a) \\
\Rrightarrow & \mathsf{a}(b); \mathsf{sce}(c); \mathsf{m}_{\overline{x}}(s); \mathsf{cc}(a); \mathsf{h}(\overline{x}); \mathsf{pc}(a)
\end{array}
$$

Solving the Verification Problem

	exec	scexec	symexec
Recursion	Yes	No	No
Looping	Yes	No	No
Branching	Infinite	Infinite	Finite
Is Algorithm	No	No	Yes

exec $\xrightarrow{\text{Assertions, Predicates, Routine contracts, Loop invariants}}$ scexec $\xrightarrow{\text{Symbols, Path Condition, Fresh Symbols, Theorem Prover}}$ symexec

Definition (Soundness)

$$\text{safe\_program}(c) \Leftarrow \text{sc-safe\_program}(c) \Leftarrow \text{sym-safe\_program}(c)$$

Contents

Semiconcrete Execution: Example Trace

routine range(i, n, r)

req $r \mapsto ?dummy$ **ens** $r \mapsto ?list * list(list)$

s:**0**[i:5, n:8, r:41], h:**0**

produce($r \mapsto ?dummy$)

s:**0**[i:5, n:8, r:41], h:$\{\!\{41 \mapsto 77\}\!\}$

if i = n **then** l := 0 **else** (

l := **malloc**(2);

s:**0**[i:5, n:8, r:41, l:50], h:$\{\!\{41 \mapsto 77, mb(50, 2), 50 \mapsto 88, 51 \mapsto 99\}\!\}$

[l] := i; range(i + 1, n, l + 1)

consume($l+1 \mapsto ?dummy$); *produce*($l+1 \mapsto ?list * list(list)$)

s:**0**[i:5, n:8, r:41, l:50], h:$\{\!\{41 \mapsto 77, mb(50, 2), 50 \mapsto 5, 51 \mapsto 60, list(60)\}\!\}$

); **close** list(l); [r] := l

s:**0**[i:5, n:8, r:41, l:50], h:$\{\!\{41 \mapsto 50, list(50)\}\!\}$

consume($r \mapsto ?list * list(list)$)

s:**0**[i:5, n:8, r:41, l:50], h:**0**

Symbolic Execution: Example Trace

routine range(i, n, r)

req r $\mapsto$?dummy **ens** r $\mapsto$?list $*$ list(list)

Φ:{i,n,r}, s:**0**[i:i, n:n, r:r], h:**0** $\qquad \Phi:\{\ldots,\varsigma,\ldots\} = \Phi:\{\ldots,\varsigma=\varsigma,\ldots\}$

sproduce(r $\mapsto$?dummy)

Φ:{i,n,r,d}, s:**0**[i:i, n:n, r:r], h:$\{\!|r\mapsto d|\!\}$

if i = n **then** l := 0 **else** (

Φ:{i,n,r,d, $i\neq n$}, s:**0**[i:i, n:n, r:r], h:$\{\!|r\mapsto d|\!\}$

l := **malloc**(2);

Φ:{i,n,r,d,l,v,v', $i\neq n$,$0<l$}, s:**0**[i:i, n:n, r:r, l:l], h:$\{\!|r\mapsto d,\mathrm{mb}(l,2),l\mapsto v,l+1\mapsto v'|\!\}$

[l] := i; range(i + 1, n, l + 1)

sconsume(l+1$\mapsto$?dummy); sproduce(l+1$\mapsto$?list $*$ list(list))

Φ:{i,n,r,d,l,v,v',l', $i\neq n$,$0<l$}, s:**0**[i:i, n:n, r:r, l:l],
h:$\{\!|r\mapsto d,\mathrm{mb}(l,2),l\mapsto i,l+1\mapsto l',\mathrm{list}(l')|\!\}$

); **close** list(l); [r] := l

Φ:{i,n,r,d,l,v,v',l', $i\neq n$,$0<l$}, s:**0**[i:i, n:n, r:r, l:l], h:$\{\!|r\mapsto l,\mathrm{list}(l)|\!\}$

sconsume(r $\mapsto$?list $*$ list(list))

Φ:{i,n,r,d,l,v,v',l', $i\neq n$,$0<l$}, s:**0**[i:i, n:n, r:r, l:l], h:**0**

Symbolic Execution

$$
\begin{array}{rl}
 & \varsigma \in \mathit{Symbols} \\
t, \hat{\ell}, \hat{v} \in \mathit{Terms} & ::= z \mid \varsigma \mid t + t \mid t - t \\
\varphi \in \mathit{Formulae} & ::= t = t \mid t < t \mid \neg\varphi
\end{array}
$$

$$
\begin{array}{rl}
\hat{s} \in \mathit{SStores} = & \mathit{Vars} \rightarrow \mathit{Terms} \\
\mathit{SPredicates} = & \{\mapsto, \mathsf{mb}\} \cup \mathit{UserDefinedPredicates} \\
\mathit{SChunks} = & \{p(\overline{\hat{v}}) \mid p \in \mathit{SPredicates}, \overline{\hat{v}} \in \mathit{Terms}\} \\
\hat{h} \in \mathit{SHeaps} = & \mathit{SChunks} \rightarrow \mathbb{N} \\
\mathit{PathConditions} = & \mathcal{P}(\mathit{Formulae}) \\
\mathit{SStates} = & \mathit{PathConditions} \times \mathit{SStores} \times \mathit{SHeaps} \\
\mathit{SMutators} = & \mathit{SStates} \rightarrow \mathit{Outcomes}(\mathit{SStates})
\end{array}
$$

$$
\begin{array}{rl}
\mathit{sconsume}(a) \in & \mathit{Assertions} \rightarrow \mathit{SMutators} \\
\mathit{sproduce}(a) \in & \mathit{Assertions} \rightarrow \mathit{SMutators} \\
\mathit{symexec}(c) \in & \mathit{Commands} \rightarrow \mathit{SMutators}
\end{array}
$$

Auxiliary Mutators

$$\mathsf{sassume}(\varphi) = \lambda(\Phi, \hat{s}, \hat{h}).\ \bigotimes \Phi \not\vdash_{\mathrm{SMT}} \neg\varphi.\ \langle(\Phi \cup \{\varphi\}, \hat{s}, \hat{h})\rangle$$
$$\mathsf{sassume}(b) = \hat{s} \leftarrow \mathsf{sstore}; \mathsf{sassume}([\![b]\!]_{\hat{s}})$$
$$\mathsf{sassert}(b) = \lambda(\Phi, \hat{s}, \hat{h}).\ \bigoplus \Phi \vdash_{\mathrm{SMT}} [\![b]\!]_{\hat{s}}.\ \langle(\Phi, \hat{s}, \hat{h})\rangle$$
$$\mathsf{Used}(\Phi) = \{\varsigma \in \mathit{Symbols} \mid \varsigma = \varsigma \in \Phi\}$$
$$\mathsf{fresh}(\Phi) = \epsilon(\{\varsigma \mid \varsigma \in \mathit{Symbols} \wedge \varsigma \notin \mathsf{Used}(\Phi)\})$$
$$\mathsf{fresh} = \lambda(\Phi, \hat{s}, \hat{h}).\ \textbf{let}\ \varsigma = \mathsf{fresh}(\Phi)\ \textbf{in}\ \langle(\Phi \cup \{\varsigma = \varsigma\}, \hat{s}, \hat{h}), \varsigma\rangle$$

$$\bigoplus t.\ C(t) = \Phi \leftarrow \mathsf{pc}; \bigoplus t \in \mathit{Terms}, \mathrm{FS}(t) \subseteq \mathsf{Used}(\Phi).\ C(t)$$
$$\mathsf{sconsume}(\hat{h}') = \lambda(\Phi, \hat{s}, \hat{h}).\ \bigoplus \hat{h}'' \leq \hat{h}, \Phi \vdash_{\mathrm{SMT}} \hat{h}'' = \hat{h}'.\ \langle(\Phi, \hat{s}, \hat{h} - \hat{h}'')\rangle$$
$$\mathsf{sproduce}(\hat{h}') = \lambda(\Phi, \hat{s}, \hat{h}).\ \langle(\Phi, \hat{s}, \hat{h} \uplus \hat{h}')\rangle$$

where

$$\epsilon(X) \quad = \quad \text{some element of } X$$

Producing Assertions

$$\mathsf{sproduce}(b) = \mathsf{sassume}(b)$$

$$\mathsf{sproduce}(p(\overline{e}, \overline{?x})) = \\ \quad \overline{\hat{v}} \leftarrow \mathsf{seval}(e); \overline{\hat{v}}' \leftarrow \mathsf{fresh}; \mathsf{sproduce}(p(\overline{\hat{v}}, \overline{\hat{v}}')); \overline{x} := \overline{\hat{v}}'$$

$$\mathsf{sproduce}(a * a') = \mathsf{sproduce}(a); \mathsf{sproduce}(a')$$

$$\mathsf{sproduce}(\textbf{if } b \textbf{ then } a \textbf{ else } a') = \\ \quad \mathsf{sassume}(b); \mathsf{sproduce}(a) \otimes \mathsf{sassume}(\neg b); \mathsf{sproduce}(a')$$

Consuming Assertions

$$\mathsf{sconsume}(b) = \mathsf{sassert}(b)$$

$$\mathsf{sconsume}(p(\overline{e}, \overline{?x})) = \\ \quad \overline{\hat{v}} \leftarrow \mathsf{seval}(e); \bigoplus \overline{\hat{v}}'.\ \mathsf{sconsume}(p(\overline{\hat{v}}, \overline{\hat{v}}')); \overline{x} := \overline{\hat{v}}'$$

$$\mathsf{sconsume}(a * a') = \mathsf{sconsume}(a); \mathsf{sconsume}(a')$$

$$\mathsf{sconsume}(\textbf{if } b \textbf{ then } a \textbf{ else } a') = \\ \quad \mathsf{sassume}(b); \mathsf{sconsume}(a) \otimes \mathsf{sassume}(\neg b); \mathsf{sconsume}(a')$$

Symbolic Execution of Commands

$$\mathsf{symexec}(x := e) = \hat{v} \leftarrow \mathsf{seval}(e); x := \hat{v}$$

$$\mathsf{symexec}(c; c') = \mathsf{symexec}(c); \mathsf{symexec}(c')$$

$$\mathsf{symexec}(\textbf{if } b \textbf{ then } a \textbf{ else } a') = \mathsf{sassume}(b); \mathsf{symexec}(c) \otimes \mathsf{sassume}(\neg b); \mathsf{symexec}(c')$$

$$\mathsf{symexec}(r(\overline{e})) = \overline{\hat{v}} \leftarrow \mathsf{eval}(\overline{e}); \mathsf{with}(\mathbf{0}[\overline{x} := \overline{\hat{v}}], \mathsf{sconsume}(a); \mathsf{sproduce}(a'))$$
$$\text{where } \textbf{routine } r(\overline{x}) \textbf{ req } a \textbf{ ens } a'$$

Symbolic Execution of Commands

$$\text{shavoc}(\overline{x}) = \overline{\hat{v}} \leftarrow \text{fresh}; \overline{x} := \overline{\hat{v}}$$

$$\text{sleakcheck} = \lambda(\Phi, \hat{s}, \hat{h}).\ \bigoplus \hat{h} = \mathbf{0}.\ \top$$

$$\begin{array}{l}
\text{symexec}(\textbf{while}\ b\ \textbf{inv}\ a\ \textbf{do}\ c) = \\
\quad \hat{s} \leftarrow \text{sstore}; \text{with}(\hat{s}, \text{sconsume}(a)); \\
\quad \text{shavoc}(\text{targets}(c)); \\
\quad (\\
\qquad \text{sheap} := \mathbf{0}; \\
\qquad \hat{s} \leftarrow \text{sstore}; \text{with}(\hat{s}, \text{sproduce}(a)); \\
\qquad \text{sassume}(b); \text{symexec}(c); \\
\qquad \hat{s} \leftarrow \text{sstore}; \text{with}(\hat{s}, \text{sconsume}(a)); \\
\qquad \text{sleakcheck} \\
\quad \otimes \\
\qquad \hat{s} \leftarrow \text{sstore}; \text{with}(\hat{s}, \text{sproduce}(a)) \\
\qquad \text{sassume}(\neg b); \\
\quad)
\end{array}$$

Symbolic Execution of Commands

$$\text{symexec}(x := \textbf{malloc}(n)) =$$
$$\quad \hat{\ell}, \hat{v}_1, \ldots, \hat{v}_n \leftarrow \text{fresh}; \text{sassume}(0 < \hat{\ell});$$
$$\quad \text{sproduce}(\text{mb}(\hat{\ell}, n), \hat{\ell} \mapsto \hat{v}_1, \ldots, \hat{\ell} + n - 1 \mapsto \hat{v}_n); x := \hat{\ell}$$

$$\text{symexec}(x := [e]) =$$
$$\quad \hat{\ell} \leftarrow \text{seval}(e); \bigoplus \hat{v}.\ \text{sconsume}(\hat{\ell} \mapsto \hat{v}); \text{sproduce}(\hat{\ell} \mapsto \hat{v}); x := \hat{v}$$

$$\text{symexec}([e] := e') =$$
$$\quad \hat{\ell}, \hat{v} \leftarrow \text{seval}(e, e'); \bigoplus \hat{v}'.\ \text{sconsume}(\hat{\ell} \mapsto \hat{v}'); \text{sproduce}(\hat{\ell} \mapsto \hat{v})$$

$$\text{symexec}(\textbf{free}(e)) = \hat{\ell} \leftarrow \text{seval}(e);$$
$$\quad \bigoplus n, \hat{v}_1, \ldots, \hat{v}_n.\ \text{sconsume}(\text{mb}(\hat{\ell}, n), \hat{\ell}_1 \mapsto \hat{v}_1, \ldots, \hat{\ell}_n \mapsto \hat{v}_n)$$

Symbolic Execution of Commands

$$\text{symexec}(\textbf{open}\ p(\overline{e})) = \overline{\hat{v}} \leftarrow \text{eval}(\overline{e});\ \text{sconsume}(p(\overline{\hat{v}}));\ \text{with}(\mathbf{0}[\overline{x} := \overline{\hat{v}}], \text{sproduce}(a))\ \text{where}\ \textbf{predicate}\ p(\overline{x}) = a$$

$$\text{symexec}(\textbf{close}\ p(\overline{e})) = \overline{\hat{v}} \leftarrow \text{eval}(\overline{e});\ \text{with}(\mathbf{0}[\overline{x} := \overline{\hat{v}}], \text{sconsume}(a));\ \text{sproduce}(p(\overline{\hat{v}}))\ \text{where}\ \textbf{predicate}\ p(\overline{x}) = a$$

Validity of Routines

$$
\begin{array}{l}
\mathsf{svalid}(r) = \\
\quad (\emptyset, \mathbf{0}, \mathbf{0}) \triangleright \\
\quad \overline{\hat{v}} \leftarrow \mathsf{fresh}; \\
\quad \mathsf{with}(\mathbf{0}[\overline{x} := \overline{\hat{v}}], \\
\qquad \hat{s}' \leftarrow \mathsf{with}(\mathbf{0}[\overline{x} := \overline{\hat{v}}], \mathsf{sproduce}(a); \mathsf{sstore}); \\
\qquad \mathsf{symexec}(c); \\
\qquad \mathsf{with}(\hat{s}', \mathsf{sconsume}(a')) \\
\quad); \\
\quad \mathsf{sleakcheck} \\
\quad \{\mathrm{true}\} \\
\quad \text{where } \textbf{routine } r(\overline{x}) \textbf{ req } a \textbf{ ens } a' = c
\end{array}
$$

Soundness of symbolic execution: Definitions

$$
\begin{array}{rcl}
I \in \mathit{Interps} & = & \mathit{Symbols} \rightharpoonup \mathbb{Z} = \mathit{Symbols} \to \mathbb{Z} \cup \{\mathsf{undef}\} \\
\mathrm{dom}\, I & = & \{\varsigma \mid I(\varsigma) \neq \mathsf{undef}\} \\
I \subseteq I' & = & \forall \varsigma.\ I(\varsigma) = \mathsf{undef} \vee I(\varsigma) = I'(\varsigma) \\
I((\Phi, \hat{s}, \hat{h})) & = & \begin{cases} (s, h) & \text{if } \mathrm{dom}\, I = \mathsf{Used}(\Phi) \wedge [\![\Phi, \hat{s}, \hat{h}]\!]_I = \mathsf{true}, s, h \\ \mathsf{undef} & \text{otherwise} \end{cases} \\
\rho_I & = & \lambda \hat{\sigma}.\ \bigotimes I' \supseteq I, \sigma, I'(\hat{\sigma}) = \sigma.\ \langle \sigma \rangle \\
C \rightsquigarrow_I C' & = & C;, \rho_I \Rrightarrow \rho_I; C' \\
C(\cdot) \rightsquigarrow_I C'(\cdot) & = & \forall I' \supseteq I, t, v, [\![t]\!]_{I'} = v.\ C(t) \rightsquigarrow_{I'} C'(v)
\end{array}
$$

Soundness of symbolic execution: SMT Solver

$$\Phi \vDash \varphi \quad \Leftrightarrow \quad \forall I.\ [\![\Phi]\!]_I = \mathsf{true} \Rightarrow [\![\varphi]\!]_I = \mathsf{true}$$

Assumption (SMT Solver Soundness)

$$\Phi \vdash_{\mathrm{SMT}} \varphi \quad \Rightarrow \quad \Phi \vDash \varphi$$

Soundness of symbolic execution: Properties

Lemma (Soundness)

$$
\begin{aligned}
C(\cdot) \rightsquigarrow_I C'(\cdot) &\Rightarrow \hat{v} \leftarrow \text{fresh}; C(\hat{v}) \rightsquigarrow_I \bigotimes v.\ C'(v) \\
C(\cdot) \rightsquigarrow_I C'(\cdot) &\Rightarrow \bigoplus \hat{v}.\ C(\hat{v}) \rightsquigarrow_I \bigoplus v.\ C'(v) \\
\text{sassume}(b), \text{sassert}(b) &\rightsquigarrow_I \text{assume}(b), \text{assert}(b) \\
[\![\hat{h}]\!]_I = h \Rightarrow \text{sconsume}(\hat{h}), \text{sproduce}(\hat{h}) &\rightsquigarrow_I \text{consume}(h), \text{produce}(h) \\
\text{sconsume}(a), \text{sproduce}(a) &\rightsquigarrow_I \text{consume}(a), \text{produce}(a) \\
\text{symexec}(c) &\rightsquigarrow_I \text{scexec}(c) \\
\text{svalid}(r) &\Rightarrow \text{valid}(r) \\
\text{sym-safe\_program}(c) &\Rightarrow \text{sc-safe\_program}(c)
\end{aligned}
$$

Soundness: fresh

Lemma (Soundness of fresh)

$C(\cdot) \rightsquigarrow_I C'(\cdot) \Rightarrow \hat{v} \leftarrow \mathsf{fresh}; C(\hat{v}) \rightsquigarrow_I \bigotimes v.\ C'(v)$

Proof.

Assume $C(\cdot) \rightsquigarrow_I C'(\cdot)$. Fix Φ, $\hat{s}$, $\hat{h}$, Q.
Let $\varsigma := \epsilon(\{\varsigma \mid \varsigma \notin \mathsf{Used}(\Phi)\})$.
Assume $(\Phi \cup \{\varsigma = \varsigma\}, \hat{s}, \hat{h}) \triangleright C(\varsigma); \rho_I\ \{Q\}$.
Stp $(\Phi, \hat{s}, \hat{h}) \triangleright \rho_I; \bigotimes v.\ C'(v)\ \{Q\}$.
Fix $I' \supseteq I, s, h$ st $I'((\Phi, \hat{s}, \hat{h})) = (s, h)$. Fix v.
Stp $(s, h) \triangleright C'(v)\ \{Q\}$.
Let $I'' := I'[\varsigma := v]$. We have $I''((\Phi \cup \{\varsigma = \varsigma), \hat{s}, \hat{h})) = (s, h)$.
Stp $(\Phi \cup \{\varsigma = \varsigma\}, \hat{s}, \hat{h}) \triangleright \rho_{I''}; C'(v)\ \{Q\}$.
By $\rho_I \Rrightarrow \rho_{I''}$ and first assumption.

Soundness: sassume

Lemma (Soundness of sassume)

$\mathsf{sassume}(b) \rightsquigarrow_I \mathsf{assume}(b)$

Proof.

Fix $\Phi, \hat{s}, \hat{h}, Q, I' \supseteq I, s, h$ st $I'((\Phi, \hat{s}, \hat{h})) = (s, h)$.
Assume $\Phi \models_{\mathrm{SMT}} \neg[\![b]\!]_{\hat{s}} \Rightarrow (\Phi \cup \{[\![b]\!]_{\hat{s}}\}, \hat{s}, \hat{h}) \triangleright \rho_I \{Q\}$.
Assume $[\![b]\!]_s = \mathsf{true}$.
Stp $(s, h) \in Q$.
Note $[\![\Phi]\!]_{I'} = \mathsf{true}$ and $[\![[\![b]\!]_{\hat{s}}]\!]_{I'} = \mathsf{true}$.
Hence $\Phi \not\models \neg[\![b]\!]_{\hat{s}}$.
Hence $\Phi \not\models_{\mathrm{SMT}} \neg[\![b]\!]_{\hat{s}}$ (by soundness SMT solver).
Hence $(\Phi \cup \{[\![b]\!]_{\hat{s}}\}, \hat{s}, \hat{h}) \triangleright \rho_I \{Q\}$.
Note $I'((\Phi \cup \{[\![b]\!]_{\hat{s}}\}, \hat{s}, \hat{h})) = (s, h)$.
Thus $(s, h) \in Q$.

Solving the Verification Problem

	exec	scexec	symexec
Recursion	Yes	No	No
Looping	Yes	No	No
Branching	Infinite	Infinite	Finite
Is Algorithm	No	No	Yes

exec $\xrightarrow{\text{Assertions, Predicates, Routine contracts, Loop invariants}}$ scexec $\xrightarrow{\text{Symbols, Path Condition, Fresh Symbols, Theorem Prover}}$ symexec

Definition (Soundness)

$$\text{safe\_program}(c) \Leftarrow \text{sc-safe\_program}(c) \Leftarrow \text{sym-safe\_program}(c)$$

Contents

Mechanised FVF versus FVF: Program Syntax

$$
\begin{array}{rl}
 & z \in \mathbb{Z}, n \in \mathbb{N} \\
 & x \in \mathit{Vars} \\
e ::= & z \mid x \mid e + e \\
b ::= & e = e \mid e < e \mid \neg b \\
c ::= & x := e \mid (c; c) \mid \textbf{if } b \textbf{ then } c \textbf{ else } c \mid \textbf{skip} \mid \textbf{message } \mathit{text} \\
 & \mid x := r(\overline{e}) \mid x := \textbf{malloc}(n) \mid x := [e] \mid [e] := e \mid \textbf{free}(e) \\
 & \mid \textbf{while } b \textbf{ inv } a \textbf{ do } c \mid \textbf{open } q(\overline{e}, \overline{?\_}) \mid \textbf{close } q(\overline{e}) \\
\mathit{rdef} ::= & \textbf{routine } r(\overline{x}) = c \\
 & \\
 & q \in \mathit{UserDefinedPredicates} \\
p ::= & \mapsto \mid \mathsf{mb} \mid q \\
a ::= & b \mid p(\overline{e}, \overline{?x}) \mid a * a \mid \textbf{if } b \textbf{ then } a \textbf{ else } a \\
\mathit{preddef} ::= & \textbf{predicate } q(\overline{x}) = a \\
\mathit{rspec} ::= & \textbf{routine } r(\overline{x}) \textbf{ req } a \textbf{ ens } a
\end{array}
$$

Mechanised FVF versus FVF: Consumption

FVF:

$\mathsf{symexec}(x := [e]) =$

$\quad \hat{\ell} \leftarrow \mathsf{seval}(e); \bigoplus \hat{v}.\ \mathsf{sconsume}(\hat{\ell} \mapsto \hat{v}); \mathsf{sproduce}(\hat{\ell} \mapsto \hat{v}); x := \hat{v}$

$\mathsf{sconsume} \in \mathit{SHeaps} \to \mathit{SOutcomes}(\mathsf{unit})$

MFVF:

$\mathsf{symexec}(x := [e]) =$

$\quad \hat{\ell} \leftarrow \mathsf{seval}(e); [\hat{v}] \leftarrow \mathsf{sconsume}(\mapsto, [\hat{\ell}], 1); \mathsf{sproduce}(\hat{\ell} \mapsto \hat{v}); x := \hat{v}$

$\mathsf{sconsume} \in \mathit{SPredicates} \to \mathit{Terms}^* \to \mathbb{N} \to \mathit{SOutcomes}(\mathit{Terms}^*)$

Executability: Outcomes

```
Inductive type_name := n_Empty_set | n_bool | n_Z | n_T(T : Type).

Fixpoint ltype_name(n : type_name) : Type := match n with
  | n_Empty_set ⇒ Empty_set
  | n_bool ⇒ bool
  | n_Z ⇒ Z
  | n_T T ⇒ T
  end.

Inductive set(X : Type) := set_(n : type_name)(f : ltype_name n → X).

Inductive outcome(S A : Type) :=
| single(s : S)(a : A)
| demonic(os : set (outcome S A))
| angelic(os : set (outcome S A))
| message(msg : string)(o : outcome S A).
```

MFVF: Symbolic Execution is Executable

```
Definition p :=
  predicate list(l) =
    if l = 0 then 0 = 0 else mb(l, 2) ∗ l ↦ _ ∗ l + 1 ↦ ?next ∗ list(next)

Compute svalid_routine [p] [] [l] list(l) list(result)
  (
    close list(b);
    while ¬(a = 0) inv list(a) ∗ list(b) do (
      open list(a);
      n := [a + 1]; [a + 1] := b; b := a; a := t;
      close list(b)
    );
    open list(a);
    result := b
  ).
ok
```

MFVF: Concrete Execution is Semi-Executable

```
Definition atZ z o := match o with
  | Some (demonic (set_ n_Z o')) ⇒ Some (o' z)
  | _ ⇒ None end.
Definition isSingle o :=
  match o with Some (single _ _) ⇒ true | _ ⇒ false end.
Definition isFail o := match o with
  | Some (angelic (set_ n_Empty_set _)) ⇒ true
  | _ ⇒ false end.

Definition o := cstate0 ▷ exec [] (x := malloc(1); [42] := 123).

Compute Some o ▷ atZ 2 ▷ atZ 42 ▷ isSingle.
true

Compute Some o ▷ atZ 2 ▷ atZ 43 ▷ isFail.
true
```

MFVF: Soundness

```
Theorem soundness rspecs pdefs rdefs c :
  svalid_program rspecs pdefs rdefs c →
  cvalid_program rdefs c.
Proof.
...
Qed.

Print Assumptions soundness.
Coq.Sets.Ensembles.Extensionality_Ensembles
Coq.Logic.Classical_Prop.classic
Coq.Logic.IndefiniteDescription.constructive_indefinite_description
Coq.Logic.FunctionalExtensionality.functional_extensionality_dep
```

MFVF: Full Coq Sources

Full Coq sources at

http://www.cs.kuleuven.be/~bartj/fvf/