Featherweight VeriFast
Formal Definition, Soundness Proof, Mechanisation

Frédéric Vogels, Bart Jacobs, and Frank Piessens

October 15, 2014

Notes

Contents

Notes

Contents

Notes

The Programming Language

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

Notes

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)
```


Notes

Contents

Notes

Example Concrete Execution

```
// s = 0, h = 0
pair := malloc(2);
// s = 0[pair := 100], h = {|mb(100, 2), 100 ↦ 42, 101 ↦ 24|}
[pair] := 0;
// s = 0[pair := 100], h = {|mb(100, 2), 100 ↦ 0, 101 ↦ 24|}
free(pair)
// s = 0[pair := 100], h = 0
```

where

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

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

$\{\!|e_1, \ldots, e_n, e_{n+1}|\!\} = \{\!|e_1, \ldots, e_n|\!\} + \{\!|e_{n+1}|\!\}$

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

Notes

Concrete Execution: Example Trace

```
routine range(i, n, r) =
s:0[i:5, n:8, r:41], h:h0⊎{|41↦77|}
if i = n then l := 0 else (
s:0[i:5, n:8, r:41], h:h0⊎{|41↦77|}
l := malloc(2);
s:0[i:5, n:8, r:41, l:50], h:h0⊎{|41↦77,mb(50, 2),50↦88,51↦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:h0⊎{|41↦77,mb(50, 2),50↦5,51↦60,
  mb(60, 2),60 ↦ 6,61 ↦ 70,mb(70, 2),70 ↦ 7,71 ↦ 0|}
);
[r] := l
s:0[i:5, n:8, r:41, l:50], h:h0⊎{|41 ↦ 50,mb(50, 2),50↦5,51↦60,
  mb(60, 2),60↦6,61↦70,mb(70, 2),70↦7,71↦0|}
where h0 : {|mb(30),30↦3,31↦40,mb(40, 2),40↦4|}
```


Notes

Concrete Execution States

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

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

Notes

Concrete Execution: Singleton Outcomes

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

Notes

Concrete Execution: Allocation

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

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

Notes

Concrete Execution: Demonic Choice

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

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

Notes

Concrete Execution: Failure, Nontermination, Angelic Choice

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

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

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

Notes

Concrete Execution

$$\mathit{CMutators} = \mathit{CStates} \to \mathit{Outcomes}(\mathit{CStates})$$

$$\mathsf{exec} \in \mathit{Commands} \to \mathit{CMutators}$$

Notes

Outcomes

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

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

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

Notes

Outcomes with Answers

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

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

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

Notes

Outcomes: Satisfaction, Coverage

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

$\phi\ \{Q\}$ ("outcome ϕ satisfies postcondition Q")

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

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

Notes

Outcomes: Sequential composition

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

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

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

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* $C_1; C_2 \Rrightarrow C_1'; C_2'$.

Lemma (Satisfaction)

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

Notes

Outcomes: Sequential composition (with Answers)

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

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

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

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*
$x \leftarrow C_1; C_2(x) \Rrightarrow x \leftarrow C_1'; C_2'(x)$

Lemma (Satisfaction)

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

Notes

Outcomes: Notations

$$\begin{array}{ll} \bigotimes \tilde{C} = \lambda \sigma.\ \bigotimes \{C \in \tilde{C}.\ C(\sigma)\} & \tilde{C} \subseteq \mathit{Mutators}(\mathcal{S}, \mathcal{S}', \mathcal{A}) \\ \bigoplus \tilde{C} = \lambda \sigma.\ \bigoplus \{C \in \tilde{C}.\ C(\sigma)\} & \tilde{C} \subseteq \mathit{Mutators}(\mathcal{S}, \mathcal{S}', \mathcal{A}) \end{array}$$

$$\begin{array}{l} \bigotimes i \in I.\ \phi_i = \bigotimes \{\phi_i \mid i \in I\} \\ \bigoplus i \in I.\ \phi_i = \bigoplus \{\phi_i \mid i \in I\} \end{array}$$

$$\begin{array}{ll} \bigotimes \mathsf{true}.\ \phi = \phi & \bigotimes \mathsf{false}.\ \phi = \top \\ \bigoplus \mathsf{true}.\ \phi = \phi & \bigoplus \mathsf{false}.\ \phi = \bot \end{array}$$

$$\begin{array}{l} \mathsf{yield}\ a = \lambda \sigma.\ \langle \sigma, a \rangle \\ \mathsf{noop} = \mathsf{yield}\ \mathsf{tt} \end{array}$$

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

Notes

Some Auxiliary Definitions

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

Notes

Concrete Execution of Commands

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

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

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

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

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

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

Notes

Concrete Execution of Commands

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

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

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

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

Notes

Concrete Execution of Commands

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

Notes

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)

Notes

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 (Assertions, Predicates, Routine contracts, Loop invariants) → scexec (Symbols, Path Condition, Fresh Symbols, Theorem Prover) → symexec

Definition (Soundness)

safe_program(c) ⇐ sc-safe_program(c) ⇐ sym-safe_program(c)

Notes

Contents

Notes

Annotations: Simple Example

```
routine swap(cell1, cell2)
  req cell1 ↦ ?v1 ∗ cell2 ↦ ?v2
  ens cell1 ↦ v2 ∗ cell2 ↦ v1
  =
    value1 := [cell1];
    value2 := [cell2];
    [cell1] := value2;
    [cell2] := value1
```


Notes

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
```


Notes

Annotations

$q \in \mathit{UserDefinedPredicates}$

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

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

Notes

Concrete Execution: Example Trace

routine range(i, n, r) =
s:**0**[i:5, n:8, r:41], h:$h_0 \uplus$ {41↦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↦77,mb(50, 2),50↦88,51↦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↦77,mb(50, 2),50↦5,51↦60,
mb(60, 2),60 ↦ 6,61 ↦ 70,mb(70, 2),70 ↦ 7,71 ↦ 0}
);
[r] := l
s:**0**[i:5, n:8, r:41, l:50], h:$h_0 \uplus$ {41 ↦ 50,mb(50, 2),50↦5,51↦60,
mb(60, 2),60↦6,61↦70,mb(70, 2),70↦7,71↦0}
where h_0 : {mb(30),30↦3,31↦40,mb(40, 2),40↦4}

Notes

Semiconcrete Execution: Example Trace

routine range(i, n, r)
req r ↦ ?dummy **ens** r ↦ ?list ∗ list(list)
s:**0**[i:5, n:8, r:41], h:**0**
produce(r ↦ ?dummy)
s:**0**[i:5, n:8, r:41], h:{41↦77}
if i = n **then** l := 0 **else** (
l := **malloc**(2);
s:**0**[i:5, n:8, r:41, l:50], h:{41↦77,mb(50, 2),50↦88,51↦99}
[l] := i; range(i + 1, n, l + 1)
consume(l+1↦?dummy); *produce*(l+1↦?list ∗ list(list))
s:**0**[i:5, n:8, r:41, l:50], h:{41↦77,mb(50, 2),50↦5,51↦60,list(60)}
); **close** list(l); [r] := l
s:**0**[i:5, n:8, r:41, l:50], h:{41↦50,list(50)}
consume(r ↦ ?list ∗ list(list))
s:**0**[i:5, n:8, r:41, l:50], h:**0**

Notes

Semiconcrete execution

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

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

Notes

Some Auxiliary Definitions

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

Notes

Producing Assertions

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

$$
\begin{aligned}
&\mathrm{produce}(p(\overline{e}, \overline{?x})) = \\
&\quad \overline{v} \leftarrow \mathrm{eval}(\overline{e});\ \bigotimes \overline{v}'.\ \mathrm{produce}(p(\overline{v}, \overline{v}'));\ \overline{x} := \overline{v}'
\end{aligned}
$$

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

$$
\begin{aligned}
&\mathrm{produce}(\textbf{if } b \textbf{ then } a \textbf{ else } a') = \\
&\quad \mathrm{assume}(b);\ \mathrm{produce}(a) \otimes \mathrm{assume}(\neg b);\ \mathrm{produce}(a')
\end{aligned}
$$

Notes

Consuming Assertions

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

$$
\begin{aligned}
&\mathrm{consume}(p(\overline{e}, \overline{?x})) = \\
&\quad \overline{v} \leftarrow \mathrm{eval}(\overline{e});\ \bigoplus \overline{v}'.\ \mathrm{consume}(p(\overline{v}, \overline{v}'));\ \overline{x} := \overline{v}'
\end{aligned}
$$

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

$$
\begin{aligned}
&\mathrm{consume}(\textbf{if } b \textbf{ then } a \textbf{ else } a') = \\
&\quad \mathrm{assume}(b);\ \mathrm{consume}(a) \otimes \mathrm{assume}(\neg b);\ \mathrm{consume}(a')
\end{aligned}
$$

Notes

Semiconcrete Execution of Commands

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

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

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

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

Notes

Semiconcrete Execution of Commands

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

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

Notes

Semiconcrete Execution of Commands

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

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

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

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

Notes

Semiconcrete Execution of Commands

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

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

Notes

Validity of Routines

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

Notes

Semiconcrete Execution: Program Safety

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

Notes

Soundness: The Consumption Relation

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

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

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

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

Notes

Soundness: The Production Relation

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

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

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

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

Notes

Soundness: Consumption and Production

Lemma (Consumption and Production and the Relations)

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

Lemma (Consumption Locality)

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

Lemma (Consumption Monotonicity)

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

Lemma (Production after Consumption (Relations))

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

Notes

Soundness: Consumption and Production

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

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

Lemma (Consumption and Production)

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

Notes

Soundness: Locality and Modifies

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

Lemma (Locality)

$\text{local scexec}(c)$

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

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

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

Lemma (Semiconcrete Execution Modifies Targets)

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

Notes

Soundness

Definition (Heap refinement)

$$\frac{h_{\text{c}} \lhd h \qquad \textbf{predicate}\ p(\overline{x}) = a \qquad (\mathbf{0}[\overline{x} := \overline{v}], h) \xrightarrow{a}_{\text{c}} (s', \mathbf{0})}{h_{\text{c}} \lhd \{\!|p(\overline{v})|\!\}}$$

$$h_{\text{c}} \lhd h_{\text{c}} \qquad \frac{h_{\text{c}} \lhd h \qquad h'_{\text{c}} \lhd h'}{h_{\text{c}} \uplus h'_{\text{c}} \lhd h \uplus h'}$$

Lemma (Open, Close)

$$h_{\text{c}} \lhd h \uplus \{\!|p(\overline{v})|\!\} \Leftrightarrow \exists s', h', (\mathbf{0}[\overline{x} := \overline{v}], h') \xrightarrow{a}_{\text{c}} (s', \mathbf{0}) \wedge h_{\text{c}} \lhd h \uplus h'$$

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

Notes

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. □

Notes

Soundness: Routine Calls

Proof.

Goal: $\mathsf{scexec}(r(\overline{e})); \kappa \Rrightarrow \kappa; \mathsf{exec}_{n+1}(r(\overline{e}))$.
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 $\mathsf{scexec}(\cdot)$).

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

□

Notes

Soundness: Loops

Proof.

Goal: $\mathsf{scexec}(\textbf{while}\ b\ \textbf{inv}\ a\ \textbf{do}\ c); \kappa \Rrightarrow \kappa; \mathsf{exec}_{n+1}(\textbf{while}\ b\ \textbf{inv}\ a\ \textbf{do}\ c)$.
Stp $\mathsf{m}_{\overline{x}}(s); \mathsf{cc}(a); \mathsf{h}(\overline{x}); (\mathsf{clh}; \mathsf{pc}(a); \mathsf{a}(b); \mathsf{sce}(c); \mathsf{cc}(a); \mathsf{lck} \otimes \mathsf{pc}(a); \mathsf{a}(\neg b)); \kappa \Rrightarrow \kappa; (\mathsf{a}(b); \mathsf{e}_n(c))^*; \mathsf{a}(\neg b)$.
Assume $\mathsf{m}_{\overline{x}}(s); \mathsf{h}(\overline{x}) \Rrightarrow \mathsf{pc}(a); \mathsf{a}(b); \mathsf{sce}(c); \mathsf{cc}(a)$.
Stp $\mathsf{m}_{\overline{x}}(s); \mathsf{cc}(a); \mathsf{h}(\overline{x}); \mathsf{pc}(a); \mathsf{a}(\neg b)); \kappa \Rrightarrow \kappa; (\mathsf{a}(b); \mathsf{e}_n(c))^*; \mathsf{a}(\neg b)$.
Stp $\mathsf{m}_{\overline{x}}(s); \mathsf{cc}(a); \mathsf{h}(\overline{x}); \mathsf{pc}(a) \Rrightarrow (\mathsf{a}(b); \mathsf{sce}(c))^*$ (by IH).
Stp $\mathsf{m}_{\overline{x}}(s); \mathsf{cc}(a); \mathsf{h}(\overline{x}); \mathsf{pc}(a) \Rrightarrow \mathsf{a}(b); \mathsf{sce}(c); \mathsf{m}_{\overline{x}}(s); \mathsf{cc}(a); \mathsf{h}(\overline{x}); \mathsf{pc}(a)$.

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

□

Notes

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 —(Assertions, Predicates, Routine contracts, Loop invariants)→ scexec —(Symbols, Path Condition, Fresh Symbols, Theorem Prover)→ symexec

Definition (Soundness)

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

Notes

Contents

Notes

Semiconcrete Execution: Example Trace

routine range(i, n, r)
req r ↦ ?dummy **ens** r ↦ ?list ∗ list(list)
s:**0**[i:5, n:8, r:41], h:**0**
produce(r ↦ ?dummy)
s:**0**[i:5, n:8, r:41], h:{41↦77}
if i = n **then** l := 0 **else** (
l := **malloc**(2);
s:**0**[i:5, n:8, r:41, l:50], h:{41↦77,mb(50, 2),50↦88,51↦99}
[l] := i; range(i + 1, n, l + 1)
consume(l+1↦?dummy); *produce*(l+1↦?list ∗ list(list))
s:**0**[i:5, n:8, r:41, l:50], h:{41↦77,mb(50, 2),50↦5,51↦60,list(60)}
); **close** list(l); [r] := l
s:**0**[i:5, n:8, r:41, l:50], h:{41↦50,list(50)}
consume(r ↦ ?list ∗ list(list))
s:**0**[i:5, n:8, r:41, l:50], h:**0**

Notes

Symbolic Execution: Example Trace

routine range(i, n, r)
req r ↦ ?dummy **ens** r ↦ ?list ∗ list(list)
Φ:{i,n,r}, s:**0**[i:i, n:n, r:r], h:**0** Φ:{..., ς, ...} = Φ:{..., ς = ς, ...}
sproduce(r ↦ ?dummy)
Φ:{i,n,r,d}, s:**0**[i:i, n:n, r:r], h:{r↦d}
if i = n **then** l := 0 **else** (
Φ:{i,n,r,d, i≠n}, s:**0**[i:i, n:n, r:r], h:{r↦d}
l := **malloc**(2);
Φ:{i,n,r,d,l,v,v', i≠n,0<l}, s:**0**[i:i, n:n, r:r, l:l], h:{r↦d,mb(l, 2),l↦v,l+1↦v'}
[l] := i; range(i + 1, n, l + 1)
sconsume(l+1↦?dummy); sproduce(l+1↦?list ∗ list(list))
Φ:{i,n,r,d,l,v,v',l', i≠n,0<l}, s:**0**[i:i, n:n, r:r, l:l],
h:{r↦d,mb(l, 2),l↦i,l+1↦l',list(l')}
); **close** list(l); [r] := l
Φ:{i,n,r,d,l,v,v',l', i≠n,0<l}, s:**0**[i:i, n:n, r:r, l:l], h:{r↦l,list(l)}
sconsume(r ↦ ?list ∗ list(list))
Φ:{i,n,r,d,l,v,v',l', i≠n,0<l}, s:**0**[i:i, n:n, r:r, l:l], h:**0**

Notes

Symbolic Execution

$$
\begin{array}{rl}
 & \varsigma \in \mathit{Symbols} \\
t, \hat{\ell}, \hat{v} \in \mathit{Terms} & ::= z \mid \varsigma \mid t + t \mid t - t \\
\varphi \in \mathit{Formulae} & ::= t = t \mid t < t \mid \neg\varphi \\
 & \\
\hat{s} \in \mathit{SStores} = & \mathit{Vars} \to \mathit{Terms} \\
\mathit{SPredicates} = & \{\mapsto, \mathrm{mb}\} \cup \mathit{UserDefinedPredicates} \\
\mathit{SChunks} = & \{p(\hat{\ell}, \hat{v}) \mid p \in \mathit{SPredicates}, \hat{\ell}, \hat{v} \in \mathit{Terms}\} \\
\hat{h} \in \mathit{SHeaps} = & \mathit{SChunks} \to \mathbb{N} \\
\mathit{PathConditions} = & \mathcal{P}(\mathit{Formulae}) \\
\mathit{SStates} = & \mathit{PathConditions} \times \mathit{SStores} \times \mathit{SHeaps} \\
\mathit{SMutators} = & \mathit{SStates} \to \mathit{Outcomes}(\mathit{SStates}) \\
 & \\
\mathit{sconsume}(a) \in & \mathit{Assertions} \to \mathit{SMutators} \\
\mathit{sproduce}(a) \in & \mathit{Assertions} \to \mathit{SMutators} \\
\mathit{symexec}(c) \in & \mathit{Commands} \to \mathit{SMutators}
\end{array}
$$

Notes

Auxiliary Mutators

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

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

where
$\epsilon(X)$ = some element of X

Notes

Producing Assertions

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

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

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

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

Notes

Consuming Assertions

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

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

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

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

Notes

Symbolic Execution of Commands

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

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

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

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

Notes

Symbolic Execution of Commands

$\mathsf{shavoc}(\overline{x}) = \overline{\hat{v}} \leftarrow \mathsf{fresh}; \overline{x} := \overline{\hat{v}}$
$\mathsf{sleakcheck} = \lambda(\Phi, \hat{s}, \hat{h}).\ \bigoplus \hat{h} = \mathbf{0}.\ \top$

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

Notes

Symbolic Execution of Commands

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

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

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

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

Notes

Symbolic Execution of Commands

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

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

Notes

Validity of Routines

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

Notes

Soundness of symbolic execution: Definitions

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

Notes

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)

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

Notes

Soundness of symbolic execution: Properties

Lemma (Soundness)

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

Notes

Soundness: fresh

Lemma (Soundness of fresh)

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

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.
□

Notes

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 $\Phi \not\vdash_{\mathrm{SMT}} \neg[\![b]\!]_{\hat{s}} \Rightarrow (\Phi \cup \{[\![b]\!]_{\hat{s}}\}, \hat{s}, \hat{h}) \triangleright \rho_I \{Q\}$.
Assume $[\![b]\!]_s = \mathsf{true}$.
Stp $(s, h) \in Q$.
Note $[\![\Phi]\!]_{I'} = \mathsf{true}$ and $[\![[\![b]\!]_{\hat{s}}]\!]_{I'} = \mathsf{true}$.
Hence $\Phi \not\vdash \neg[\![b]\!]_{\hat{s}}$.
Hence $\Phi \not\vdash_{\mathrm{SMT}} \neg[\![b]\!]_{\hat{s}}$ (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 $(s, h) \in Q$.

□

Notes

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 (Assertions, Predicates, Routine contracts, Loop invariants) → scexec (Symbols, Path Condition, Fresh Symbols, Theorem Prover) → symexec

Definition (Soundness)

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

Notes

Contents

Notes

Mechanised FVF versus FVF: Program Syntax

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

Notes

Mechanised FVF versus FVF: Consumption

FVF:

symexec($x := [e]$) =
  $\hat{\ell} \leftarrow$ seval(e); $\bigoplus \hat{v}$. sconsume($\hat{\ell} \mapsto \hat{v}$); sproduce($\hat{\ell} \mapsto \hat{v}$); $x := \hat{v}$
sconsume $\in$ *SHeaps* $\rightarrow$ *SOutcomes*(unit)

MFVF:

symexec($x := [e]$) =
  $\hat{\ell} \leftarrow$ seval(e); $[\hat{v}] \leftarrow$ sconsume($\mapsto, [\hat{\ell}], 1$); sproduce($\hat{\ell} \mapsto \hat{v}$); $x := \hat{v}$
sconsume $\in$ *SPredicates* $\rightarrow$ *Terms** $\rightarrow \mathbb{N} \rightarrow$ *SOutcomes*(*Terms**)

Notes

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).
```


Notes

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
```


Notes

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
```


Notes

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

Notes

MFVF: Full Coq Sources

Full Coq sources at

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

Notes

Notes

Notes