Exercises

- Introduce construct random(x) in the language and give its natural semantics
    - Is it possible to compile away "random" if we have the choose construct?

- Introduce parallelism and give its structural operational semantics
    - Natural semantics?

$$
\begin{array}{lcll}
Stmt & ::= & x := e & \text{(assignment)} \\
 & \mid & \textbf{assume } b & \\
 & \mid & \textbf{assert } b & \\
 & \mid & S_1; S_2 & \text{(sequence)} \\
 & \mid & \textbf{if } b \textbf{ then } S_1 \textbf{ else} S_2 & \\
 & \mid & \textbf{while } b \textbf{ do } S & \\
 & \mid & S_1 \textbf{ par } S_2 & \text{(parallelism)}
\end{array}
$$

$$x := 1 \textbf{ par } x := 2; x := x + 2;$$

Parallelism

$\langle x := 1 \text{ par } x := 2; x := x+2, [x \mapsto 0] \rangle$

$\Rightarrow \langle x := 1 \text{ par } x := x+2, [x \mapsto 2] \rangle$

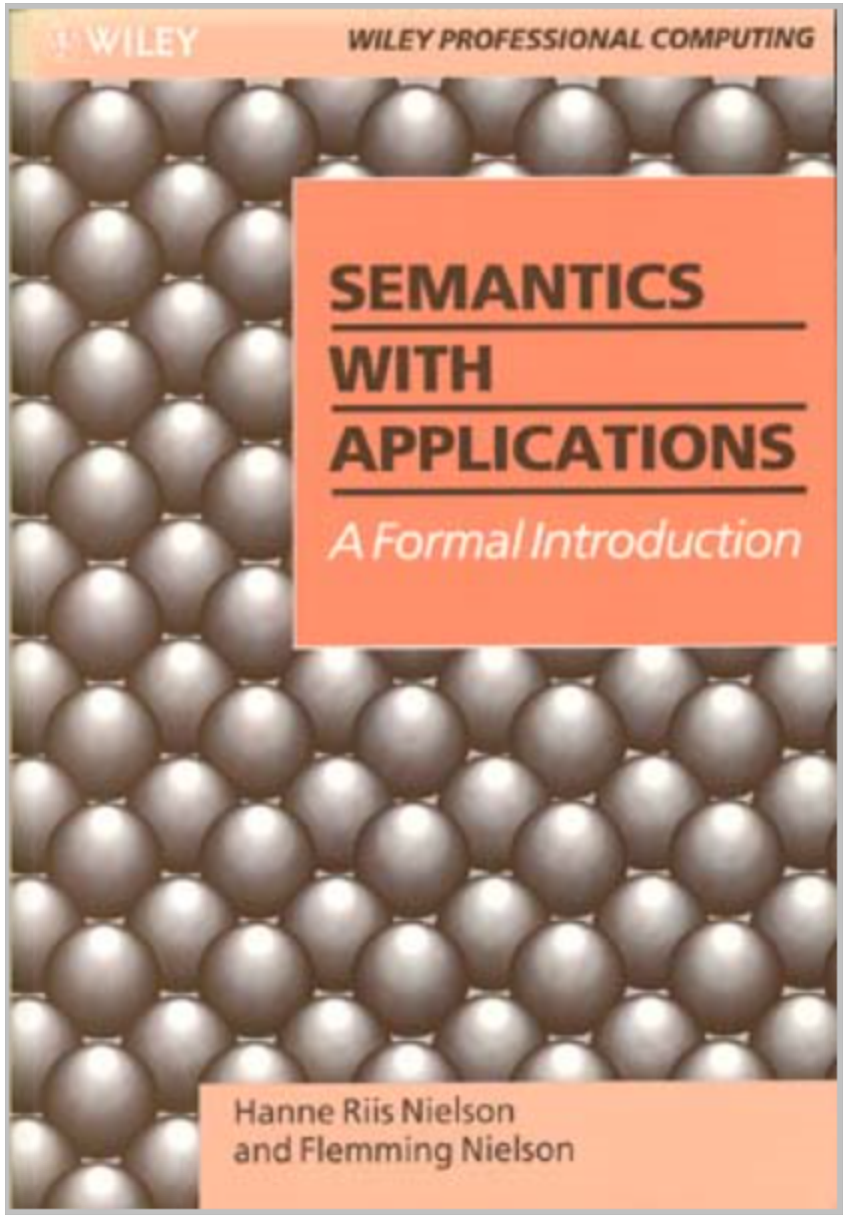
✓ $$\frac{\langle S_1, \sigma\rangle \Rightarrow \langle S_1', \sigma'\rangle}{\langle S_1 \textbf{ par } S_2, \sigma\rangle \Rightarrow \langle S_1' \textbf{ par } S_2, \sigma'\rangle}(\text{PAR})$$

✓ $$\frac{\langle S_1, \sigma\rangle \Rightarrow \sigma'}{\langle S_1 \textbf{ par } S_2, \sigma\rangle \Rightarrow \langle S_2, \sigma'\rangle}(\text{PAR})$$

✓ $$\frac{\langle S_2, \sigma\rangle \Rightarrow \langle S_2', \sigma'\rangle}{\langle S_1 \textbf{ par } S_2, \sigma\rangle \Rightarrow \langle S_1 \textbf{ par } S_2', \sigma'\rangle}(\text{PAR})$$

✓ $$\frac{\langle S_2, \sigma\rangle \Rightarrow \sigma'}{\langle S_1 \textbf{ par } S_2, \sigma\rangle \Rightarrow \langle S_1, \sigma'\rangle}(\text{PAR})$$

Not really possible with natural semantics because $\langle S, \sigma\rangle \longrightarrow \sigma'$ describes the complete execution of S. The complete executions of S_1 and S_2 is not enough to generate complete executions of S_1 **par** S_2.



Semantics With Applications: A Formal Introduction
by Hanne Riis Nielson, Flemming Nielson

Procedures

$$
\begin{array}{lll}
\mathit{Stmt} & ::= & x := e \quad \text{(assignment)} \\
& | & \textbf{assume } b \\
& | & \textbf{assert } b \\
& | & S_1; S_2 \quad \text{(sequence)} \\
& | & \textbf{if } b \textbf{ then } S_1 \textbf{ else } S_2 \\
& | & \textbf{while } b \textbf{ do } S \\
& | & \textbf{call } p
\end{array}
$$

local

$$
\mathit{ProcDecl} \quad ::= \quad \textbf{procedure } p \ \{ \ \textbf{let } V \textbf{ in } S \ \}
$$

V ≡ set of variables

$$
\mathit{Program} \quad ::= \quad \textbf{let } V \textbf{ in } \mathit{ProcDecl}^*; S
$$

$\textbf{let } \{x, y\} \textbf{ in } P \ x := 5; y := 1; \textbf{call } \mathit{fact};$ y = 5!

$P \equiv \textbf{procedure } \mathit{fact} \ \{ \ \textbf{let } \{\} \textbf{ in if } x = 1 \textbf{ then } \mathit{skip} \textbf{ else } y := y * x; x := x - 1; \textbf{call } \mathit{fact} \ \}$

Natural Semantics

Globals

Locals in scope

$$State \equiv (\overline{Var \to Int}, V_1, V_2)$$

For a map σ, let $\boxed{\sigma|_V}$ be its projection to V

only pass globals to p

new locals

$$\frac{\textbf{procedure}\ p\ \{\ \textbf{let}\ L'\ \textbf{in}\ S\ \} \qquad \langle S, (\sigma|_G, G, L')\rangle \longrightarrow (\sigma', G, L')}{\langle \textbf{call}\ p, (\sigma, G, L)\rangle \longrightarrow \sigma'|_G \cup \sigma|_L}\ (\textsc{Call})$$

$$\frac{\text{proc. } p\ \{\ \text{let}\ \{z\}\ \text{in}\ S\ \} \qquad \langle S, [x \mapsto 0], \{x\}, \{z\}\rangle \longrightarrow [x \mapsto 3, z \mapsto 2], \{x\}\{z\}}{\langle \text{call}\ p, ([x \mapsto 0, y \mapsto 1], \{x\}, \{y\})\rangle \longrightarrow [x \mapsto 3, y \mapsto 1], \{x\}, \{y\}}$$

Structural Operational Semantics

stack

$$State \equiv (Var \to Int, V_1, V_2)^+$$

value of globals

$$\frac{\textbf{procedure } p \ \{ \ \textbf{let } L' \ \textbf{in } S \ \}}{\langle \textbf{call } p, \Gamma(\sigma, G, L)\rangle \Rightarrow \langle S; \textbf{return}, \Gamma(\sigma, G, L)(\sigma|_G, G, L')\rangle} \ (\textsc{Call})$$

same

push a new stack frame

$$\frac{}{\langle \textbf{return}, \Gamma(\sigma, G, L)(\sigma', G, L')\rangle \Rightarrow \Gamma(\sigma|_L \cup \sigma'|_G, G, L)} \ (\textsc{Return})$$

pop

restore

globals

locals of callee

Symbolic Execution

$State \equiv Var \to Int$

$[x \mapsto 0, y \mapsto 1]$

$[x \mapsto 2, y \mapsto 1]$

$2 \leq 1?$ → No

$x := y + 1;$
$\textbf{assert } x > y$

$SymState \equiv Var \to Term$

$[x \mapsto a, y \mapsto b]$

$[x \mapsto b + 1, y \mapsto b]$

$b + 1 \leq b?$ → UNSAT

Symbolic Execution

Program Syntax

$$
\begin{array}{lcl}
Expr & ::= & n \in Int \\
 & | & x \in Var \\
 & | & e_1 + e_2 \\
 & | & e_1 - e_2
\end{array}
$$

$$
\begin{array}{lcl}
BoolC & ::= & true \quad | \quad false \\
 & | & e_1 = e_2 \\
 & | & e_1 < e_2 \\
 & | & \neg b \\
 & | & b_1 \wedge b_2 \\
 & | & b_1 \vee b_2
\end{array}
$$

$$
\begin{array}{lcl}
Term\ t & ::= & n \in Int \\
 & | & x \in Const \\
 & | & t_1 + t_2 \\
 & | & t_1 - t_2 \\
 & | & ite(\phi, t_1, t_2)
\end{array}
$$

Term : Int

$$
\begin{array}{lcl}
Formula\ \varphi & ::= & true \quad | \quad false \\
 & | & t_1 = t_2 \\
 & | & t_1 < t_2 \\
 & | & \neg \phi \\
 & | & \phi_1 \wedge \phi_2 \\
 & | & \phi_1 \vee \phi_2
\end{array}
$$

Formula : Bool

Symbolic Execution

$(\tilde{\varphi}, \sigma)$

$$Expr ::= n \in Int \mid x \in Var \mid e_1 + e_2 \mid e_1 - e_2$$

$$BoolC ::= true \mid false \mid e_1 = e_2 \mid e_1 < e_2 \mid \neg b \mid b_1 \wedge b_2 \mid b_1 \vee b_2$$

$$SymState : Formula \times Var \to Term$$
$$SymEval : Expr \times SymState \to Term$$

$$\begin{aligned}
SymEval(n, ss) &= n \\
SymEval(x, (\phi, \sigma)) &= \sigma(x) \\
SymEval(e_1 + e_2, ss) &= SymEval(e_1, ss) + SymEval(e_2, ss) \\
SymEval(e_1 - e_2, ss) &= SymEval(e_1, ss) - SymEval(e_2, ss)
\end{aligned}$$

constructs a term

$$SymBeval : BoolC \times SymState \to Formula$$

$$\begin{aligned}
SymBeval(e_1 = e_2, ss) &= SymEval(e_1, ss) = SymEval(e_2, ss)\ ?\ true : false \\
SymBeval(e_1 < e_2, ss) &= SymEval(e_1, ss) < SymEval(e_2, ss)\ ?\ true : false \\
SymBeval(\neg b, ss) &= \neg SymBeval(b, ss) \\
SymBeval(b_1 \wedge b_2, ss) &= SymBeval(b_1, ss) \wedge SymBeval(b_2, ss) \\
SymBeval(b_1 \vee b_2, ss) &= SymBeval(b_1, ss) \vee SymBeval(b_2, ss)
\end{aligned}$$

Symbolic Execution

$$
\begin{array}{lll}
Stmt & ::= & x := e \quad \text{(assignment)} \\
& | & \textbf{assume } b \\
& | & \textbf{assert } b \\
& | & S_1; S_2 \quad \text{(sequence)} \\
& | & \textbf{if } b \textbf{ then } S_1 \textbf{ else} S_2 \\
& | & \textbf{while } b \textbf{ do } S
\end{array}
$$

$$
\frac{}{\langle \underline{x := e}, (\phi, \sigma)\rangle \longrightarrow (\phi, \sigma[x \mapsto \underline{SymEval}(e, ss)])}\,(\textsc{Assign})
$$

$$
\frac{\langle S_1, ss\rangle \longrightarrow ss' \qquad \langle S_2, ss'\rangle \longrightarrow ss''}{\langle S_1; S_2, ss\rangle \longrightarrow ss''}\,(\textsc{Sequence})
$$

$$
\frac{}{\langle x := y; z := x + 4, (\phi, [x \mapsto a, y \mapsto b, z \mapsto c])\rangle \longrightarrow ???}
$$

$(\phi, [x \mapsto b, y \mapsto b, z \mapsto b+4])$

Symbolic Execution

$$
\begin{array}{lll}
Stmt & ::= & x := e \quad \text{(assignment)} \\
 & | & \textbf{assume } b \\
 & | & \textbf{assert } b \\
 & | & S_1; S_2 \quad \text{(sequence)} \\
 & | & \textbf{if } b \textbf{ then } S_1 \textbf{ else} S_2 \\
 & | & \textbf{while } b \textbf{ do } S
\end{array}
$$

(ϕ, σ)

if ϕ holds then my current state is σ

$(a = 0, x \mapsto a, y \mapsto b+a, z \mapsto b)$

Prove$(\neg(b + a = b) \wedge a = 0)$

UNSAT

$z := y$

assume $x = 0$

$y := y + z;$

assert $y = z;$

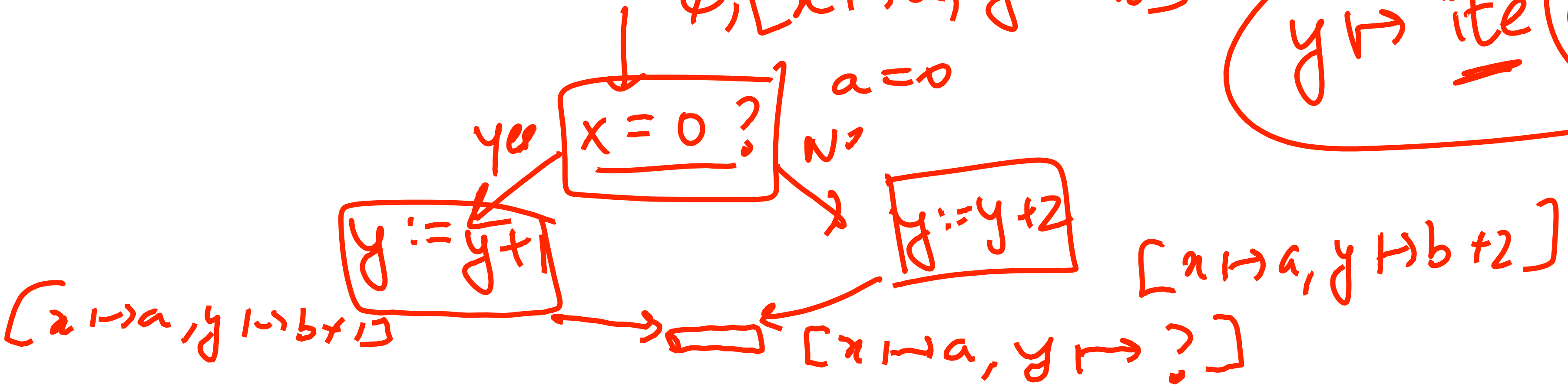
$$\frac{\textsc{Prove}(\neg SymBeval(b, \sigma) \wedge \phi)}{\langle \textbf{assert } b, (\phi, sigma) \rangle \longrightarrow (SymBeval(b, \sigma) \wedge \phi, \sigma)} (\textsc{Assert})$$

$$\frac{}{\langle \textbf{assume } b, (\phi, \sigma) \rangle \longrightarrow (SymBeval(b, \sigma) \wedge \phi, \sigma)} (\textsc{Assume})$$

Symbolic Execution

$$\frac{\langle S_1,(\phi,\sigma)\rangle \longrightarrow (\phi_1,\sigma_1) \qquad \langle S_2,(\phi,\sigma)\rangle \longrightarrow (\phi_2,\sigma_2)}{\langle \textbf{if } b \textbf{ then } S_1 \textbf{ else } S_2,(\phi,\sigma)\rangle \longrightarrow (ite(SymBeval(b),\phi_1,\phi_2), ite(SymBeval(b),\sigma_1,\sigma_2))}\ (\textsc{Ite})$$

if $x = 0$ then $y := y+1$ else $y := y+2$



$y \mapsto ite(a=0, b+1, b+2)$

random(x)

$$\frac{n \in \mathrm{Int}}{\langle \mathrm{random}(x), \sigma \rangle \to \sigma[x \mapsto n]}$$

$$\frac{m \text{ fresh constant}}{\langle \mathrm{random}(x), \phi, \sigma \rangle \longrightarrow (\phi, \sigma[x \mapsto m])}$$

$$\langle x \mapsto a, y \mapsto b \rangle \xrightarrow{\mathrm{random}(x)} \langle x \mapsto c, y \mapsto b \rangle$$

$$\big(\underbrace{\mathrm{random}(x) \,;\, \mathrm{assume}\ x == 5}_{[x \mapsto c]}\big) \equiv x := 5$$

$$(c == 5, x \mapsto c) \equiv (\mathrm{true}, x \mapsto 5)$$

$S_1 \,\square\, S_2$

$$\frac{\langle S_1, \sigma\rangle \to \sigma'}{\langle S_1 \square S_2, \sigma\rangle \to \sigma'}\ S_1 \qquad \frac{\langle S_2, \sigma\rangle \to \sigma'}{\langle S_1 \square S_2, \sigma\rangle \to \sigma'}\ S_2$$

$$\frac{\langle S_1, (\phi, \sigma)\rangle \to \phi_1, \sigma_1 \qquad \langle S_2, (\phi, \sigma)\rangle \to \phi_2, \sigma_2}{\langle S_1 \square S_2, (\phi, \sigma)\rangle \longrightarrow ite(b, \phi_1, \phi_2), ite(b, \sigma_1, \sigma_2)}$$

$(S_1 \square S_2) \equiv$ random(b); if b then S_1 else S_2

fresh Boolean constant.

$[x_0 \mapsto t]$

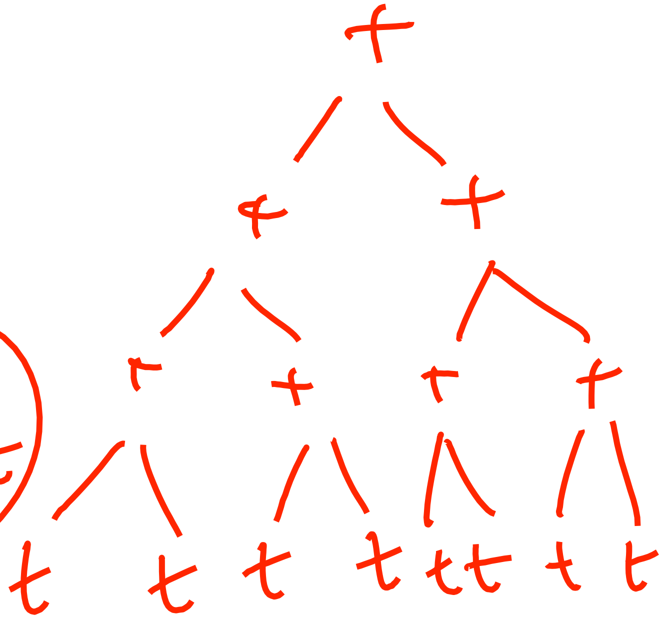
$x_1 := x_0 + x_0$ $\qquad x_1 \to t + t$

$x_2 := x_1 + x_1$ $\qquad x_2 \to t + t + t + t$

$x_3 := x_2 + x_2$

$\vdots$

$x_n := x_{n-1} + x_{n-1}$ $\qquad x_n \mapsto \sum^{2^n} t$



$+$
$(\;)$
$+$
$(\;)$
$+$
$(\;)$
t

$b_1 \ldots b_n \equiv 2^n$

$[x \mapsto t]$

if b_1 then $x := x+1$ else $x := x+2$; $\quad x \mapsto \mathrm{ite}(b_1, t+1, t+2)$

if b_2 then $x := x+1$ else $x := x+2$; $\quad x \mapsto \mathrm{ite}(b_2, \ +1, \ +2)$

$\vdots$

if b_n then $x := x+1$ else $x := x+2$; $\quad \vdots$

exponential blowup