CS256/Winter 2009 Lecture #2

Zohar Manna

SPL (Simple Programming Language) Syntax

Basic Statements

- **skip**
- assignment

  $\underbrace{(u_1, \ldots, u_k)}_{\text{variables}} := \underbrace{(e_1, \ldots, e_k)}_{\text{expressions}}$
- **await** c

  (where c is a boolean expression)

  special case: **halt** $\equiv$ **await** F
- Communication by message-passing

  $\alpha \Leftarrow e$ (send)

  $\alpha \Rightarrow u$ (receive)

  (where α is a channel)
- Semaphore operations

  request r $(r > 0 \rightarrow r := r - 1)$

  release r $(r := r + 1)$

  (where r is an integer variable)

SPL (CON'T)

Schematic Statements

In Mutual-Exclusion programs:

- **noncritical**

  may not terminate

- **critical**

  terminates

In Producer-Consumer programs:

- **produce** x

  terminates – assign nonzero value to x

- **consume** y

  terminates

No program variables are modified by schematic statements. One exception:

“x” in **produce** x

SPL (CON'T)

Compound Statements

- Conditional
  if c **then** S_1 **else** S_2
  if c **then** S

- Concatenation
  $S_1;\ \cdots;\ S_k$

  `Example:`

  when c **do** $S \quad \equiv \quad$ **await** c; S

- Selection
  S_1 **or** $\cdots$ **or** S_k

- **while**
  while c **do** S

  `Example:`

  loop forever do $S \quad \equiv \quad$ **while** T **do** S

SPL (CON'T)

Compound Statements (Con't)

- Cooperation Statement

  ℓ: $[\underbrace{\ell_1\colon S_1;\ \widehat{\ell}_1\colon}_{\text{process}}] \parallel \cdots \parallel [\ell_k\colon S_k;\ \widehat{\ell}_k\colon];\ \widehat{\ell}\colon$

  $S_1, \ldots, S_k$ are parallel to one another
  interleaved execution.

  entry step: from ℓ to $\ell_1, \ell_2, \ldots, \ell_k$,
  exit step: from $\widehat{\ell_1}, \widehat{\ell_2}, \ldots, \widehat{\ell_k}$ to $\widehat{\ell}$.

- Block

  [$\underbrace{\textbf{local}\ \textit{declaration}}$; S]

  local *variable* , . . . , *variable* : *type* **where** $\underbrace{\varphi_i}_{y_1 = e_1,\ \ldots,\ y_n = e_n}$

SPL (CON'T)

<u>Basic types</u> – boolean, integer, character, ...

<u>Structured types</u> – array, list, set, ...

Static variable initialization
(variables get initialized at the
start of the execution)

Programs

$$P :: \left[declaration;\ P_1 :: \ [\ell_1 \colon S_1;\ \widehat{\ell}_1 \colon\ ] \parallel \cdots \parallel P_k :: \ [\ell_k \colon S_k;\ \widehat{\ell}_k \colon\ ] \right]$$

$P_1, \ldots, P_k$ are <u>top-level</u> processes
Variables in P called <u>program variables</u>

Declaration

$$\mathit{mode}\ \underbrace{\mathit{variable},\ \ldots,\ \mathit{variable}}_{\text{program variables}} \colon\ \mathit{type}\ \textbf{where}\ \varphi_i$$

$\downarrow$ $\downarrow$

in (not modified)
local
out

constraints on initial values

$\varphi_1 \wedge\ \ldots\ \wedge \varphi_n$ <u>data-precondition</u> of the program

Channel Declaration

- synchronous channels
  (no buffering capacity)

  mode $\alpha_1, \alpha_2, \ldots, \alpha_n$: **channel of** *type*

- asynchronous channels
  (unbounded buffering capacity)

  mode $\alpha_1, \alpha_2, \ldots, \alpha_n$: **channel [1..] of** *type*
  where φ_i

  - φ_i is optional

  - $\varphi_i = \Lambda$ (empty list) by default

Foundations for SPL Semantics

Labels

$$\ell : S$$

- Label ℓ identifies statement S
- Equivalence Relation $\sim_L$ between labels:
  - For ℓ: $[\ell_1 : S_1 ; \ldots ; \ell_k : S_k]$

    $\ell \ \sim_L \ \ell_1$
  - For ℓ: $[\ell_1 : S_1 \ \textbf{or} \ \ldots \ \textbf{or} \ \ell_k : S_k]$

    $\ell \ \sim_L \ \ell_1 \ \sim_L \ \cdots \ \sim_L \ \ell_k$
  - For ℓ: $[\textbf{local} \ \textit{declaration}; \ \ell_1 : S_1]$

    $\ell \ \sim_L \ \ell_1$

Note: For $\ell : [\ell_1 : S_1 || \dots || \ell_k : S_k]$
$\ell \not\sim_L \ell_1 \not\sim_L \ell_2 \not\sim_L \dots$
because of the entry step

`Example:`	In Figure 0.1
	$\ell_0 \sim_L \ell_1$
	$\ell_2 \sim_L \ell_3 \sim_L \ell_5$

$$
\begin{array}{lll}
\textbf{in} & a,\ b & : \textbf{integer where } a > 0,\ b > 0 \\
\textbf{local} & y_1,\ y_2 & : \textbf{integer where } y_1 = a,\ y_2 = b \\
\textbf{out} & g & : \textbf{integer}
\end{array}
$$

$$
\left[\begin{array}{l}
\ell_0: \left[\begin{array}{l}
\ell_1: \ \textbf{while } y_1 \neq y_2 \textbf{ do} \\
\qquad\qquad \ell_2: \left[\begin{array}{l}
\ell_3: \ \textbf{await } y_1 > y_2;\ \ell_4:\ y_1 := y_1 - y_2 \\
\quad \textbf{or} \\
\ell_5: \ \textbf{await } y_2 > y_1;\ \ell_6:\ y_2 := y_2 - y_1
\end{array}\right] \\
\ell_7: \ g := y_1
\end{array}\right] \\
\ell_8:
\end{array}\right]
$$

Figure 0.1

A Fully Labeled Program GCD-F

Locations

$[\ell]$

Identify site of control

- $[\ell]$ is the location corresponding to label ℓ.

- Multiple labels identifying different statements may identify the same location.
  $[\ell] \;=\; \{\ell' \mid \ell' \sim_L \ell\}$

Example: Fig 0.1: A fully labeled program

$[\ell_0] = [\ell_1] = \{\ell_0, \ell_1\}$ $\qquad$ $[\ell_6] = \{\ell_6\}$

$[\ell_2] = \{\ell_2, \ell_3, \ell_5\}$ $\qquad$ $[\ell_7] = \{\ell_7\}$

$[\ell_4] = \{\ell_4\}$ $\qquad$ $[\ell_8] = \{\ell_8\}$

Example: Fig 0.2: A partially labeled program

$\cancel{\ell_0}$

$\ell_3 \;\rightarrow\; \ell_2^a$

$\ell_5 \;\rightarrow\; \ell_2^b$

shortcut: label ℓ_2 "represents" $\{\ell_2,\ \ell_2^a,\ \ell_2^b\}$

$$
\begin{array}{llll}
\textbf{in} & a,\ b & : \textbf{integer where } a > 0,\ b > 0 \\
\textbf{local} & y_1,\ y_2 & : \textbf{integer where } y_1 = a,\ y_2 = b \\
\textbf{out} & g & : \textbf{integer}
\end{array}
$$

$$
\left[\begin{array}{l}
\ell_1\text{: } \textbf{while } y_1 \neq y_2 \textbf{ do} \\
\qquad \ell_2\text{: } \left[\begin{array}{l}
\ell_2^a\text{: } \textbf{await } y_1 > y_2;\ \ell_4\text{: } y_1 := y_1 - y_2 \\
\quad \textbf{or} \\
\ell_2^b\text{: } \textbf{await } y_2 > y_1;\ \ell_6\text{: } y_2 := y_2 - y_1
\end{array}\right] \\
\ell_7\text{: } g := y_1 \\
\ell_8\text{:}
\end{array}\right]
$$

Figure 0.2

A Partially Labeled Program GCD

Post Location

$$\ell\colon S;\ \widehat{\ell}\colon \qquad\qquad post(S) = [\widehat{\ell}]$$

- For $[\ell_1\colon S_1;\ \widehat{\ell}_1\colon\ ] \parallel \cdots \parallel [\ell_k\colon S_k;\ \widehat{\ell}_k\colon\ ]$

  $post(S_i) \ = \ [\widehat{\ell}_i]$, for every $i = 1, \ldots, k$

- For $S = [\ell_1\colon S_1; \ldots; \ell_k\colon S_k]$

  $post(S_i) \ = \ [\ell_{i+1}]$, for $i = 1, \ldots, k{-}1$
  $post(S_k) \ = \ post(S)$

- For $S = [\ell_1\colon S_1 \textbf{ or } \ldots \textbf{ or } \ell_k\colon S_k]$

  $post(S_1) \ = \ \cdots \ = \ post(S_k) \ = \ post(S)$

- For $S = [\textbf{if } c \textbf{ then } S_1 \textbf{ else } S_2]$

  $post(S_1) \ = \ post(S_2) \ = \ post(S)$

- For $[\ell : \textbf{while } c \textbf{ do } S']$

  $post(S') \ = \ [\ell]$

`Example:` Post Locations of Fig 0.2

$$post(\ell_1) = [\ell_7]$$

$$\begin{aligned} post(\ell_2) &= post(\ell_4) \\ &= post(\ell_6) = [\ell_1] \end{aligned}$$

$$post(\ell_2^a) = [\ell_4]$$

$$post(\ell_2^b) = [\ell_6]$$

$$post(\ell_7) = [\ell_8]$$

Ancestor

S is an ancestor of S'
if S' is a substatement of S

S is a common ancestor of S_1 and S_2
if it is an ancestor of both S_1 and S_2

S is a least common ancestor (LCA) of S_1 and S_2
if S is a common ancestor of S_1 and S_2
and any other common ancestor
of S_1 and S_2 is an ancestor of S

LCA is unique for given statements S_1 and S_2

`Example:` $\left[S_1;\ [S_2 \| S_3];\ S_4\right] \parallel S_5$

LCA of $S_2,\ S_3$	$[S_2 \| S_3]$
LCA of $S_2,\ S_4$	$\left[S_1;\ [S_2 \| S_3];\ S_4\right]$
LCA of $S_2,\ S_5$	$\left[S_1;\ [S_2 \| S_3];\ S_4\right] \parallel S_5$

Parallel Labels

- Statements S and $\widetilde{S}$ are parallel if their LCA is a cooperation statement that is different from statements S and $\widetilde{S}$

`Example:` $S = \left[S_1;\ [S_2 \| S_3];\ S_4\right] \| S_5$

Statements	LCA
S_2 parallel to S_3	$S_2 \parallel S_3$
S_2 parallel to S_5	S
S_2 not parallel to S_4	$[S_1;\ \cdots;\ S_4]$ not coop.
S_2 not parallel to $S_2 \parallel S_3$	$S_2 \parallel S_3$ same

- parallel labels – labels of parallel statements

Conflicting Labels

conflicting labels – not equivalent and not parallel

`Example:`

$$\begin{bmatrix} \ell_1\!: S_1; \\ \ell_2\!: \Big([\ell_3\!: S_3;\ \widehat{\ell}_3\!:] \parallel [\ell_4\!: S_4;\ \widehat{\ell}_4\!:]\Big); \\ \ell_5\!: S_5;\ \widehat{\ell}_5\!: \end{bmatrix} \parallel [\ell_6\!: S_6;\ \widehat{\ell}_6\!:]$$

ℓ_3 is parallel to each of $\{\ell_4, \widehat{\ell}_4, \ell_6, \widehat{\ell}_6\}$
and in conflict with each of
$\{\ell_1, \ell_2, \widehat{\ell}_3, \ell_5, \widehat{\ell}_5\}$

ℓ_6 and $\widehat{\ell}_6$ are in conflict with each other
but are parallel to each of
$\{\ell_1, \ell_2, \ell_3, \widehat{\ell}_3, \ell_4, \widehat{\ell}_4, \ell_5, \widehat{\ell}_5\}$

Critical References

Writing References:

$$\begin{array}{lrrr}
x := \ \ldots & \alpha \ \Rightarrow u & \textbf{produce}\ x & \textbf{request}\ r \\
\uparrow & \uparrow & \uparrow & \uparrow \\
 & & & \\
 & & & \textbf{release}\ \ r \\
 & & & \uparrow
\end{array}$$

Reading References: all other references

critical reference of a variable in S if:

- writing ref to a variable that has reading or writing refs in S' (parallel to S)
- reading reference to a variable that has writing references in S' (parallel to S)
- reference to a channel

Limited Critical References (LCR)

Statement obeys LCR restriction (LCR-Statement)
if each test (for await, conditional, while)
and entire statement (for assignment)
contains at most one critical reference.

`Example:` Fig 0.3
ℓ_2, m_1, m_3 are LCR-Statements
ℓ_1, m_2 violate the LCR-requirement

LCR-Program: only LCR-statements

Interleaved vs. Concurrent Execution

Claim : If P is an LCR program, then the interleaving computations of P and the concurrent executions of P give the same results.

Discussion & explanation: *Blue Book.*

$$P_1 :: \left[\begin{array}{l} \ell_1\colon\ \boxed{b} := \boxed{b} \cdot y_1 \\ \ell_2\colon\ \boxed{y_1} := y_1 - 1 \\ \ell_3\colon \end{array}\right] \quad \| \quad P_2 :: \left[\begin{array}{l} m_1\colon\ \textbf{await}\ \boxed{y_1} + y_2 \leq n \\ m_2\colon\ \boxed{b} := \boxed{b}/y_2 \\ m_3\colon\ y_2 := y_2 + 1 \\ m_4\colon \end{array}\right]$$

Figure 0.3

Critical references

SPL Semantics

Transition Semantics:

$$\begin{array}{ccc} \text{SPL P} & & \text{computation of P} \\ \downarrow & & \uparrow \\ \text{FTS } \Phi & \longrightarrow & \text{computation of } \Phi \end{array}$$

Given an SPL-program P, we can construct the corresponding FTS $\Phi = \langle V, \Theta, \mathcal{T}, \mathcal{J}, \mathcal{C} \rangle$:

- system variables V

  $Y = \{y_1, \ldots, y_n\}$ – program variables of P

  domains: as declared in P

  π – control variable

  domain: sets of locations in P

  $V = Y \cup \{\pi\}$

SPL Semantics (Con't)

Comments:

– For label ℓ, at_ℓ: $[\ell] \in \pi$

at'_ℓ: $[\ell] \in \pi'$

Note: When going from an SPL program to an FTS we lose the sequential nature of the program. We need to model control explicitly in the FTS: π can be viewed as a <u>program counter</u>.

SPL Semantics (Con't)

`Example:` Fig 0.1

$V = \{\pi, a, b, y_1, y_2, g\}$

π - ranges over subsets of

$\{[\ell_1], [\ell_2], [\ell_4], [\ell_6], [\ell_7], [\ell_8]\}$

$a, b, \ldots, g$ - range over integers

- <u>Initial Condition</u> Θ

  For $P :: \left[\text{dec}; \left[P_1 :: [\ell_1 \colon S_1;\ \widehat{\ell}_1 \colon] \parallel \cdots \parallel P_k :: [\ell_k \colon S_k;\ \widehat{\ell}_k \colon]\right]\right]$

  with data-precondition φ,

  $\Theta\colon\ \pi = \{[\ell_1], \ldots, [\ell_k]\} \ \wedge\ \varphi$

`Example:` Fig 0.1

$$\Theta\colon\ \pi = \{[\ell_1]\} \ \wedge\ \underbrace{a > 0 \ \wedge\ b > 0 \ \wedge\ y_1 = a \ \wedge\ y_2 = b}_{\text{data-precondition}}$$

$$
\begin{array}{lll}
\textbf{in} & a,\ b & : \textbf{integer where } a > 0,\ b > 0 \\
\textbf{local} & y_1,\ y_2 & : \textbf{integer where } y_1 = a,\ y_2 = b \\
\textbf{out} & g & : \textbf{integer}
\end{array}
$$

$$
\left[
\begin{array}{l}
\ell_1\!: \ \textbf{while } y_1 \neq y_2 \textbf{ do} \\
\qquad\qquad \ell_2\!: \left[
\begin{array}{l}
\ell_2^a\!: \ \textbf{await } y_1 > y_2;\ \ell_4\!: \ y_1 := y_1 - y_2 \\
\quad \textbf{or} \\
\ell_2^b\!: \ \textbf{await } y_2 > y_1;\ \ell_6\!: \ y_2 := y_2 - y_1
\end{array}
\right] \\
\ell_7\!: \ g := y_1 \\
\ell_8\!:
\end{array}
\right]
$$

Figure 0.2

A Partially Labeled Program GCD

SPL Semantics (Con't)

- <u>Transitions</u> $\mathcal{T}$

  $$\mathcal{T} = \{\tau_I\} \cup \left\{ \begin{array}{l} \text{transitions associated with} \\ \text{the statements of } P \end{array} \right\}$$

  where τ_I is the "idling transition"

  ρ_I: $V' = V$

  <u>abbreviation</u>

  - $pres(U)$: $\bigwedge_{u \in U} (u' = u)$ (where $U \subseteq V$)

    the value of $u \in U$ are preserved

  - $move(L, \widehat{L})$: $L \subseteq \pi \ \wedge \ \pi' = (\pi - L) \cup \widehat{L}$

    where L, $\widehat{L}$ are sets of locations

  - $move(\ell, \widehat{\ell})$: $move(\{[\ell]\}, \{[\widehat{\ell}]\})$

SPL Semantics (Con't)

We list the transitions (transition relations) associated with the statements of P

$\underline{\ell : S}$ $\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad$ $\underline{\rho_\ell}$

$\boxed{\textbf{Basic Statements}}$

$$\ell\text{: }\textbf{skip};\ \widehat{\ell}\text{:} \qquad \to \qquad move(\ell, \widehat{\ell}\,) \ \wedge\ pres(Y)$$

$$\ell\text{: }\overline{u} := \overline{e};\ \widehat{\ell}\text{:} \qquad \to \qquad move(\ell, \widehat{\ell}\,) \ \wedge\ \overline{u}' = \overline{e} \ \wedge\ pres\Big(Y - \{\overline{u}\}\Big)$$

SPL Semantics (Con't)

Basic Statements (Con't)

ℓ: **await** c; $\widehat{\ell}$: $\quad \rightarrow \quad move(\ell, \widehat{\ell}\,) \;\wedge\; c \;\wedge\; pres(Y)$

ℓ: **request** r; $\widehat{\ell}$: $\quad \rightarrow$

$$move(\ell, \widehat{\ell}\,) \;\wedge\; r > 0 \;\wedge\; r' = r - 1 \;\wedge\; pres\Big(Y - \{r\}\Big)$$

ℓ: **release** r; $\widehat{\ell}$: $\quad \rightarrow$

$$move(\ell, \widehat{\ell}\,) \;\wedge\; r' = r + 1 \;\wedge\; pres\Big(Y - \{r\}\Big)$$

SPL Semantics (Con't)

Basic Statements (Con't)

asynchronous send

$$\ell\colon\ \alpha \Leftarrow e;\ \widehat{\ell}\colon \qquad \to \qquad move(\ell, \widehat{\ell}\,) \ \wedge\ \alpha' = \alpha \bullet e \ \wedge\ pres\Big(Y - \{\alpha\}\Big)$$

asynchronous receive

$$\ell\colon\ \alpha \Rightarrow u;\ \widehat{\ell}\colon \qquad \to \qquad move(\ell, \widehat{\ell}\,) \ \wedge\ |\alpha| > 0 \ \wedge\ \alpha = u' \bullet \alpha' \ \wedge\ pres\Big(Y - \{u, \alpha\}\Big)$$

synchronous send-receive

$$\ell\colon\ \alpha \Leftarrow e;\ \widehat{\ell}\colon \qquad m\colon\ \alpha \Rightarrow u;\ \widehat{m}\colon$$

$$move\Big(\{\ell, m\}, \{\widehat{\ell}, \widehat{m}\}\Big) \ \wedge\ u' = e \ \wedge\ pres\Big(Y - \{u\}\Big)$$

SPL Semantics (Con't)

$\boxed{\textbf{Schematic Statements}}$ $\qquad\qquad\qquad \rho_\ell$

ℓ: **noncritical**; $\widehat{\ell}$: $\quad \rightarrow \quad move(\ell, \widehat{\ell}\,) \;\wedge\; pres\Big(Y\Big)$

(nontermination modeled by $\tau_\ell \notin \mathcal{J}$)

ℓ: **critical**; $\widehat{\ell}$: $\quad \rightarrow \quad move(\ell, \widehat{\ell}\,) \;\wedge\; pres\Big(Y\Big)$

SPL Semantics (Con't)

Compound Statements

$$\ell\colon \left[\textbf{if } c \textbf{ then } \ell_1\colon S_1 \textbf{ else } \ell_2\colon S_2\right];\ \widehat{\ell}\colon \rightarrow$$

$$\rho_\ell\colon \rho_\ell^{\mathrm{T}} \vee \rho_\ell^{\mathrm{F}} \text{ where}$$

$$\rho_\ell^{\mathrm{T}}\colon\ \mathit{move}(\ell, \ell_1)\ \wedge\ c\ \wedge\ \mathit{pres}(Y)$$

$$\rho_\ell^{\mathrm{F}}\colon\ \mathit{move}(\ell, \ell_2)\ \wedge\ \neg c\ \wedge\ \mathit{pres}(Y)$$

$$\ell\colon \left[\textbf{while } c \textbf{ do } [\widetilde{\ell}\colon \widetilde{S}\,]\right];\ \widehat{\ell}\colon \rightarrow$$

$$\rho_\ell\colon \rho_\ell^{\mathrm{T}} \vee \rho_\ell^{\mathrm{F}} \text{ where}$$

$$\rho_\ell^{\mathrm{T}}\colon\ \mathit{move}(\ell, \widetilde{\ell}\,)\ \wedge\ c\ \wedge\ \mathit{pres}(Y)$$

$$\rho_\ell^{\mathrm{F}}\colon\ \mathit{move}(\ell, \widehat{\ell}\,)\ \wedge\ \neg c\ \wedge\ \mathit{pres}(Y)$$

$$\ell\colon \left[[\ell_1\colon S_1;\ \widehat{\ell}_1\colon] \parallel \cdots \parallel [\ell_k\colon S_k;\ \widehat{\ell}_k\colon]\right];\ \widehat{\ell}\colon \rightarrow$$

$$\rho_\ell^{\mathrm{E}}\colon\ \mathit{move}\Big(\{\ell\},\ \{\ell_1, \ldots, \ell_k\}\Big)\ \wedge\ \mathit{pres}(Y)\ \text{(entry)}$$

$$\rho_\ell^{\mathrm{X}}\colon\ \mathit{move}\Big(\{\widehat{\ell}_1, \ldots, \widehat{\ell}_k\},\ \{\widehat{\ell}\,\}\Big)\ \wedge\ \mathit{pres}(Y)\ \text{(exit)}$$

Grouped Statements $\langle S \rangle$

executed in a single atomic step

`Example:`

$\langle x := y + 1;\ z := 2x + 1 \rangle$

$x' = y + 1 \quad \wedge \quad z' = 2y + 3$

the same as $(x, z) := (y + 1,\ 2y + 3)$

`Example:`

$$\underbrace{\langle a := 3; a := 5 \rangle}_{a' = 5}$$

$a = 3$ is never visible to the outside world, nor to other processes

SPL Semantics (Con't)

- Justice Set $\mathcal{J}$

  All transitions except

  τ_I and all transitions associated with **noncritical** statements

- Compassion Set $\mathcal{C}$

  All transitions associated with

  send, receive, request statements

Computations of Programs

$$\textbf{local } x\text{: } \textbf{integer where } x = 1$$

$$P_1 :: \begin{bmatrix} \ell_0\text{: } \begin{bmatrix} \ell_0^a\text{: } \textbf{await } x = 1 \\ \quad \textbf{or} \\ \ell_0^b\text{: } \textbf{skip} \end{bmatrix} \\ \ell_1\text{: } \end{bmatrix} \parallel P_2 :: \begin{bmatrix} m_0\text{: } \textbf{while } \mathrm{T} \textbf{ do} \\ \quad [m_1\text{: } x := -x] \end{bmatrix}$$

`Fig 0.4`	Process P_1 terminates in all computations.

$$\sigma\text{: } \langle \pi\text{: } \{\ell_0, m_0\}, x\text{: } 1\rangle \xrightarrow{m_0} \langle \pi\text{: } \{\ell_0, m_1\}, x\text{: } 1\rangle \xrightarrow{m_1}$$

$$\langle \pi\text{: } \{\ell_0, m_0\}, x\text{: } -1\rangle \xrightarrow{m_0} \langle \pi\text{: } \{\ell_0, m_1\}, x\text{: } -1\rangle \xrightarrow{m_1}$$

$$\langle \pi\text{: } \{\ell_0, m_0\}, x\text{: } 1\rangle \xrightarrow{m_0} \cdots$$

σ is $\boxed{\text{not}}$ a computation. Unjust towards ℓ_0^b
(enabled on all states but never taken)

Computations of Programs (Con't)

$$\textbf{local } x\text{: } \textbf{integer where } x = 1$$

$$P_1 :: \begin{bmatrix} \ell_0\text{: } \begin{bmatrix} \ell_0^a\text{: } \textbf{await } x = 1 \\ \textbf{or} \\ \ell_0^b\text{: } \textbf{await } x \neq 1 \end{bmatrix} \\ \ell_1\text{: } \end{bmatrix} \parallel P_2 :: \begin{bmatrix} m_0\text{: } \textbf{while } \mathrm{T} \textbf{ do} \\ \quad [m_1\text{: } x := -x] \end{bmatrix}$$

`Fig 0.5`	**skip** $\rightarrow$ **await** $x \neq 1$

$$\sigma\text{: } \langle \pi\text{: } \{\ell_0, m_0\}, x\text{: } 1\rangle \xrightarrow{m_0} \langle \pi\text{: } \{\ell_0, m_1\}, x\text{: } 1\rangle \xrightarrow{m_1}$$

$$\langle \pi\text{: } \{\ell_0, m_0\}, x\text{: } -1\rangle \xrightarrow{m_0} \langle \pi\text{: } \{\ell_0, m_1\}, x\text{: } -1\rangle \xrightarrow{m_1}$$

$$\langle \pi\text{: } \{\ell_0, m_0\}, x\text{: } 1\rangle \xrightarrow{m_0} \cdots$$

σ is a computation –
since none of the just transitions are
continually enabled.

Computations of Programs (Con't)

$$\textbf{local } x\text{: } \textbf{integer where } x = 1$$

$$P_1 :: \begin{bmatrix} \ell_0\text{: } \textbf{if } x = 1 \textbf{ then} \\ \quad \ell_1\text{: } \textbf{skip} \\ \textbf{else} \\ \quad \ell_2\text{: } \textbf{skip} \\ \ell_3\text{:} \end{bmatrix} \parallel P_2 :: \begin{bmatrix} m_0\text{: } \textbf{while } \mathrm{T} \textbf{ do} \\ \quad [m_1\text{: } x := -x] \end{bmatrix}$$

`Fig 0.6` Process P_1 terminates in all computations.

$$\sigma\text{: } \langle \pi\text{: } \{\ell_0, m_0\}, x\text{: } 1\rangle \xrightarrow{m_0} \langle \pi\text{: } \{\ell_0, m_1\}, x\text{: } 1\rangle \xrightarrow{m_1}$$
$$\langle \pi\text{: } \{\ell_0, m_0\}, x\text{: } -1\rangle \xrightarrow{m_0} \langle \pi\text{: } \{\ell_0, m_1\}, x\text{: } -1\rangle \xrightarrow{m_1}$$
$$\langle \pi\text{: } \{\ell_0, m_0\}, x\text{: } 1\rangle \xrightarrow{m_0} \cdots$$

σ is $\boxed{\text{not}}$ a computation –
since ℓ_0 is continually enabled,
but not taken.

Control Configurations

$L = \{[\ell_1], \ldots, [\ell_k]\}$ of P is called <u>conflict-free</u>
if no $[\ell_i]$ conflicts with $[\ell_j]$, for $i \neq j$.

L is called a (<u>control</u>) <u>configuration</u> of P
if it is a maximal conflict-free set.

Example:

$$\textbf{local } x\text{: } \textbf{integer where } x = 0$$

$$P_1 :: \begin{bmatrix} \ell_0\text{: } x := 1 \\ \ell_1\text{:} \end{bmatrix} \parallel P_2 :: \begin{bmatrix} m_0\text{: } \textbf{await } x = 1 \\ m_1\text{:} \end{bmatrix}$$

Configurations

$\{[\ell_0], [m_0]\}$, $\{[\ell_0], [m_1]\}$,
$\{[\ell_1], [m_0]\}$, $\{[\ell_1], [m_1]\}$

SPL Semantics (Con't)

<u>accessible configuration</u> –
appears as value of π in some accessible state

`Example:`
$\Big\{[\ell_0], [m_1]\Big\}$ does not appear in any accessible state

Is a given configuration accessible?

Undecidable

The Mutual-Exclusion Problem

$$\begin{array}{lcl}
\textbf{loop forever do} & & \textbf{loop forever do} \\
\quad\begin{bmatrix} \textbf{noncritical} \\ \cdots\cdots\cdots \\ \textbf{critical} \\ \cdots\cdots\cdots \end{bmatrix} & \| & \quad\begin{bmatrix} \textbf{noncritical} \\ \cdots\cdots\cdots \\ \textbf{critical} \\ \cdots\cdots\cdots \end{bmatrix}
\end{array}$$

Requirements:

- <u>Exclusion</u>

  While one of the processes is in its critical section, the other is not

- <u>Accessibility</u>

  Whenever a process is at the noncritical section exit, it must eventually reach its critical section

`Example:` mutual exclusion by semaphores

Fig. 0.7

$$\textbf{local } y\text{: } \textbf{integer where } y = 1$$

$$\underbrace{\left[\begin{array}{l} \ell_0\text{: } \textbf{loop forever do} \\ \qquad \left[\begin{array}{l} \ell_1\text{: } \textbf{noncritical} \\ \ell_2\text{: } \textbf{request } y \\ \ell_3\text{: } \textbf{critical} \\ \ell_4\text{: } \textbf{release } y \end{array}\right] \end{array}\right]}_{\text{P1}} \quad \| \quad \underbrace{\left[\begin{array}{l} m_0\text{: } \textbf{loop forever do} \\ \qquad \left[\begin{array}{l} m_1\text{: } \textbf{noncritical} \\ m_2\text{: } \textbf{request } y \\ m_3\text{: } \textbf{critical} \\ m_4\text{: } \textbf{release } y \end{array}\right] \end{array}\right]}_{\text{P2}}$$

Fig. 0.7 Program MUX-SEM

Message-Passing Programs

`Example:` Producer-Consumer Fig. 0.9

assumption:

channel <u>send</u> $\leq N$ values

$$\textbf{local } send, ack\text{: } \textbf{channel } [1..] \textbf{ of integer}$$
$$\textbf{where } send = \Lambda,\ ack = \underbrace{[1, \ldots, 1]}_{N}$$

$$Prod :: \left[\begin{array}{l} \textbf{local } x,\ t\text{: } \textbf{integer} \\ \ell_0\text{: } \textbf{loop forever do} \\ \qquad \left[\begin{array}{l} \ell_1\text{: } \textbf{produce } x \\ \ell_2\text{: } ack \Rightarrow t \\ \ell_3\text{: } send \Leftarrow x \end{array}\right] \end{array}\right] \quad || \quad Cons :: \left[\begin{array}{l} \textbf{local } y\text{: } \textbf{integer} \\ m_0\text{: } \textbf{loop forever do} \\ \qquad \left[\begin{array}{l} m_1\text{: } send \Rightarrow y \\ m_2\text{: } ack \Leftarrow 1 \\ m_3\text{: } \textbf{consume } y \end{array}\right] \end{array}\right]$$

Fig. 0.9 Program PROD-CONS