Verification

Bernd Finkbeiner, Sven Schewe,
Rayna Dimitrova, Lars Kuhtz,
Anne Proetzsch

Winter Semester 2007/2008

Announcements

- Deadline for HISPOS registration: 01.12.2007
- Next Lecture: Thursday, HS 003, 12:15-13:45
- First Tutorial:
  Wednesday 14:15-15:45 Room 015 Building E 1 3
  Fridays 10:00-11:30, Room 013 Building E 1 3

Setting: Reactive Systems

Recap

- Ongoing interaction
- Concurrent and distributed
- Generalization of sequential systems
- Computational model: Fair Transition Systems
- Specification logic: Linear Time Temporal Logic (LTL)

This Lecture

- Application Language:
  Simple Programming Language (SPL)

Transition Systems

Review

- Vocabulary: a set of typed variables $\mathcal{V}$
- Set of all states: Σ
- A (finite) set of variables $V \subseteq \mathcal{V}$
  (data variables + control variables)
- Initial condition θ
- A (finite) set of transitions $\mathcal{T}$

Transitions

Review

For each $\tau \in \mathcal{T}$: $\tau : \Sigma \mapsto 2^{\Sigma}$

(each transition is a function from states to sets of states)

- s' is a τ-successor of s if $s' \in \tau(s)$
- τ is represented by the transition relation ρ(τ) (next-state relation)

  V values of variables in the current state

  V' values of variables in the next state

Enabled/Disabled/Taken Transitions

Review

- A transition τ
  - is enabled on s if $\tau(s) \neq \{\}$
  - is disabled on s if $\tau(s) = \{\}$
- For an infinite sequence of states

  $\sigma: s_0, s_1, s_2, \ldots$

  a transition τ

  - is enabled at position k if it is enabled on s_k
  - is taken at position k if s_{k+1} is a τ-successor of s_k

The Interleaving Model

Review

Infinite sequence of states

$$\sigma: s_0, s_1, s_2, \ldots$$

is a run of a transition system, if it satisfies the following:

- Initiality: s_0 satisfies θ
- Consecution: For each i= 0,1, …

there is a transition $\tau \in \mathcal{T}$ s.t. $s_{i+1} \in \tau(s_i)$

Idling Transition

Review

- What if no transition is enabled?
- We implicitly assume that there is an idling transition (stuttering transition) τ_I

$$\rho(\tau_I) : V = V'$$

- The idling transition is always enabled.

Reachable States

Review

For a transition system Φ,
a state s is Φ-accessible if there is a run

σ: $s_0, s_1, s_2, \ldots$

with $s=s_i$, for some i.

A transition system Φ is finite-state if the set of all Φ-accessible states is finite.

Fair Transition Systems

Review

$$\Phi = (V, \theta, \mathcal{T}, \mathcal{J}, \mathcal{C})$$

- $\mathcal{J} \subseteq \mathcal{T}$: set of just (weakly fair) transitions
- $\mathcal{C} \subseteq \mathcal{T}$: set of compassionate (strongly fair) transitions

- Justice: for each just transition it is not the case that the transition is continually enabled but only taken at finitely many positions.
- Compassion: for each compassionate transition it is not the case that the transition is enabled at infinitely many positions but only taken at finitely many positions.

Computations

Review

An infinite sequence of states

$$\sigma: s_0, s_1, s_2, \ldots$$

is a computation of a fair transition system, if it satisfies:

- Initiality
- Consecution
- Justice
- Compassion

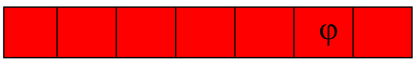
Fairness = Justice + Compassion
Computation = Run + Fairness

LTL

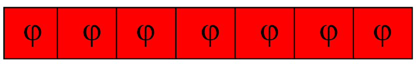
Review

- Assertion language: FO over interpreted symbols
- Boolean connectives: $\vee, \wedge, \neg$
- Modal operators:

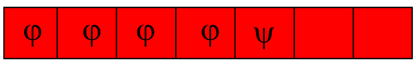
$\Diamond \varphi$ Eventually



$\Box \varphi$ Henceforth

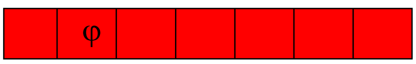


$\varphi \mathcal{U} \psi$ Until



$\varphi \mathcal{W} \psi$ Wait-for $\Box \varphi \vee \varphi \mathcal{U} \psi$

$\bigcirc \varphi$ Next



Abbreviations

Review

- $p \Rightarrow q$ stands for $\Box(p \rightarrow q)$

  (entailment)

- $p \approx q$ stands for $\Box(p \leftrightarrow q)$

  (congruence)

- $q_1 \mathcal{W} q_2 \mathcal{W} q_3 \mathcal{W} q_4$ stands for $q_1 \mathcal{W} (q_2 \mathcal{W} (q_3 \mathcal{W} q_4))$

  (nested waiting-for)

Satisfiability / Validity

Review

- For a temporal formula p and sequence σ,

  $\sigma \models p$ iff $(\sigma,0) \models p$

- The formula p is satisfiable if $\sigma \models p$ for some sequence σ

- The formula p is valid if $\sigma \models p$ for all sequences σ

P-Validity

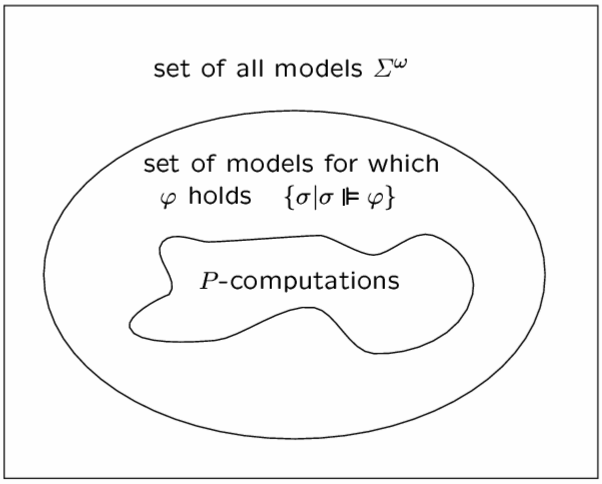
Review

- A LTL formula φ is valid over a program P, written P ⊨φ,

  if φ holds in the first state of every computation of P.

P-Validity

Review



P-Validity

Review

	general	program P
state formula q	$\Vdash q$ state valid „q holds in all states"	$P \Vdash q$ P-state valid „q holds in all P-accessible states"
temporal formula φ.	$\models \varphi$ Valid „φ holds in the first position of every sequence"	$P \models \varphi$ P-valid „φ holds in the first position of every P-computation"

P-Validity

- For state formulas:

$$\begin{array}{rcl} \Vdash q & \longleftrightarrow & \models \Box q \\ P \Vdash q & \longleftrightarrow & P \models \Box q \\ \Vdash q & \longrightarrow & P \Vdash q \end{array}$$

- For temporal formulas:

$$\models \psi \longrightarrow P \models \psi$$

Specification of Properties

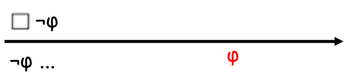
- Property Π: set of sequences
- Π is specified by temporal formula φ if for every sequence σ,

$$\sigma \in \Pi \quad \text{iff} \quad \sigma \models \varphi.$$

- Program P has property Π if all computations of P are in Π.
- If P has property Π, and Π is specified by φ, then φ is P-valid.

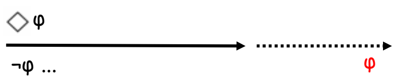
Safety versus Liveness

- "Nothing bad ever happens"
- All finite prefixes satisfy a certain requirement (does not depend on limit behavior)
- Counter-examples "are finite"



- Provable by induction over reachable states.
- Can not distinguish runs and computations
- Example:
  $\Box(\varphi \rightarrow \bigcirc\psi)$

- "Something good eventually happens"
- Does not depend on finite prefixes
- No finite counter-examples



- Proof requires assumptions about nondeterministic choices (Justice and/or Compasion)
- Example:
  $\Box\Diamond(\neg\,\text{enabled}(\tau) \text{ or } \text{taken}(\tau))$

Safety vs. Liveness (Examples)

- $\varphi \mathcal{W} \psi$ Safety
- $(\Diamond \varphi) \mathcal{U} \psi$ Liveness
- $(\Diamond \varphi) \Rightarrow (\Box \psi)$ Safety
- $\text{request} \Rightarrow \Diamond \text{grant}$ Liveness
- $\varphi \mathcal{U} \psi$ Safety and Liveness

Simple Programming Language

SPL: Simple Programming Language

local y_1, y_2: boolean where $y_1 = \mathrm{F}, y_2 = \mathrm{F}$
s : integer where $s = 1$

$P_1 ::$

ℓ_0 : loop forever do

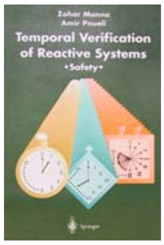
$$\left[\begin{array}{ll} \ell_1: & \text{noncritical} \\ \ell_2: & (y_1,\ s) := (\mathrm{T},\ 1) \\ \ell_3: & \text{await } (\neg y_2) \lor (s = 2) \\ \ell_4: & \text{critical} \\ \ell_5: & y_1 := \mathrm{F} \end{array}\right]$$

$||$

$P_2 ::$

m_0 : loop forever do

$$\left[\begin{array}{ll} m_1: & \text{noncritical} \\ m_2: & (y_2,\ s) := (\mathrm{T},\ 2) \\ m_3: & \text{await } (\neg y_1) \lor (s = 1) \\ m_4: & \text{critical} \\ m_5: & y_2 := \mathrm{F} \end{array}\right]$$



SPL Syntax: Basic Statements

- **skip**
- assignment

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

  boolean expression

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

  $\alpha \Leftarrow e$ (send)

  $\updownarrow$ channel

  $\alpha \Rightarrow u$ (receive)
- Semaphore operations

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

  integer variable

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

SPL Syntax: Schematic Statements

- **noncritical**

  may not terminate

- **critical**

  terminates

- **produce** x

  terminates – assign nonzero value to x

- **consume** y

  terminates

SPL Syntax: 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 $\equiv$ **await** c; S

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

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

  Example:
  loop forever do S $\equiv$ **while** $\top$ **do** S

SPL Syntax: Compound Statements (cont'd)

- Cooperation Statement

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

  $S_1;\ \cdots;\ S_k$ 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

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

  $\textbf{local}\ \ variable,\ \ldots,\ variable\ :\ type\ \textbf{where}\ \underbrace{\varphi_i}_{}$

  $$y_1 = e_1,\ \ldots, y_n = e_n$$

SPL Syntax: Grouped Statements

$\langle S \rangle$
executed in a single atomic step

- S can contain only statements that are guaranteed to terminate:
  - no **while** statements, no schematic statements

- S can contain no communication statements

(To simplify presentation. More general case in Manna/Pnueli book)

SPL Syntax: Grouped Statements

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 Syntax: Programs

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

$P_1, \ldots, P_k$ are top-level processes

Variables in P called program variables

Declaration

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

Data-precondition:

$\varphi_1 \wedge \ldots \wedge \varphi_n$

$mode$ ↓

in (not modified)
local
out

φ_i ↓ constraints on initial values

SPL Syntax: 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
  - $\varphi_i = \varepsilon$ (empty list) by default

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: $[\ell_1 : S_1$ **or** $\ldots$ **or** $\ell_k : S_k]$
    $\ell \sim_L \ell_1 \sim_L \cdots \sim_L \ell_k$

  - For ℓ: [**local** *declaration*; $\ell_1 : S_1$]
    $\ell \sim_L \ell_1$

Locations

$[\ell]$

Identify site of control

- Multiple labels identifying different statements may identify the same location.
  $[\ell] = \{\ell' \mid \ell' \sim_L \ell\}$
- $[\ell]$ is the location corresponding to label ℓ.

Example

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

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

$\ell_0 \sim_L \ell_1$
$\ell_2 \sim_L \ell_3 \sim_L \ell_5$

$[\ell_0] = \{\ell_0, \ell_1\}$	$[\ell_6] = \{\ell_6\}$
$[\ell_2] = \{\ell_2, \ell_3, \ell_5\}$	$[\ell_7] = \{\ell_7\}$
$[\ell_4] = \{\ell_4\}$	$[\ell_8] = \{\ell_8\}$

Post Location

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

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

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

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

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

  $post(S_k) = post(S)$

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

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

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

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

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

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

Example

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

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

$post(\ell_2) = post(\ell_4) = post(\ell_6) = [\ell_1]$

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

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

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

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
is an ancestor of S

LCA is unique for given statements S_1 and S_2

Parallel Labels

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

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	$\underbrace{[S_1;\ \cdots;\ S_4]}$ not cooperation

- parallel labels – labels of parallel statements

Conflicting Labels

Conflicting: not equivalent and not parallel

Example:

$$\left[\begin{array}{l} \ell_1\colon S_1; \\ \ell_2\colon \left([\ell_3\colon S_3;\ \widehat{\ell}_3\colon] \parallel [\ell_4\colon S_4;\ \widehat{\ell}_4\colon]\right); \\ \ell_5\colon S_5;\ \widehat{\ell}_5\colon \end{array}\right] \parallel [\ell_6\colon S_6;\ \widehat{\ell}_6\colon]$$

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

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

Writing references:

$$\underset{\uparrow}{x} := \ldots \qquad \alpha \Rightarrow \underset{\uparrow}{u} \qquad \textbf{produce}\ \underset{\uparrow}{x} \qquad \textbf{request}\ \underset{\uparrow}{r}$$

$$\textbf{release}\ \underset{\uparrow}{r}$$

(all other references are reading references)

Limited Critical References

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

Example:

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

ℓ_2, m_1, m_3 are LCR-Statements
ℓ_1, m_2 violate the LCR-requirement

LCR Program: only LCR statements

SPL Semantics

Transition Semantics:

SPL P	computation of P
↓	↑
FTS Φ	computation of Φ

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

- <u>system variables</u> 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
  $V = Y \cup \{\pi\}$

Program counter

For label ℓ,

at_ℓ: $[\ell] \in \pi$
at'_ℓ: $[\ell] \in \pi'$

SPL Semantics

- Initial Condition Θ

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

  data-precondition φ

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

- Transitions $\mathcal{T}$

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

  where τ_I is the "idling transition"

  $\rho_I\colon \ V' = V$

Some Abbreviations

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

  the value of $u \in U$ are preserved

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

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

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

Basic Statements

$\ell : S$		ρ_ℓ
ℓ: **skip**; $\widehat{\ell}$:	$\rightarrow$	$move(\ell, \widehat{\ell}) \;\wedge\; pres(Y)$
ℓ: $\overline{u} := \overline{e}$; $\widehat{\ell}$:	$\rightarrow$	$move(\ell, \widehat{\ell}) \;\wedge\; \overline{u}' = \overline{e} \;\wedge\; pres\big(Y - \{\overline{u}\}\big)$
ℓ: **await** c; $\widehat{\ell}$:	$\rightarrow$	$move(\ell, \widehat{\ell}) \;\wedge\; c \;\wedge\; pres(Y)$
ℓ: **request** r; $\widehat{\ell}$:	$\rightarrow$	$move(\ell, \widehat{\ell}) \;\wedge\; r > 0 \;\wedge\; r' = r - 1 \;\wedge\; pres\big(Y - \{r\}\big)$
ℓ: **release** r; $\widehat{\ell}$:	$\rightarrow$	$move(\ell, \widehat{\ell}) \;\wedge\; r' = r + 1 \;\wedge\; pres\big(Y - \{r\}\big)$

Basic Statements (cont'd)

asynchronous send

$$\ell:\ \alpha \Leftarrow e;\ \widehat{\ell}: \quad \rightarrow \quad move(\ell, \widehat{\ell}) \;\wedge\; \alpha' = \alpha \bullet e \;\wedge\; pres\big(Y - \{\alpha\}\big)$$

asynchronous receive

$$\ell:\ \alpha \Rightarrow u;\ \widehat{\ell}: \quad \rightarrow \quad move(\ell, \widehat{\ell}) \;\wedge\; |\alpha| > 0 \;\wedge\; \alpha = u' \bullet \alpha' \;\wedge\; pres\big(Y - \{u, \alpha\}\big)$$

synchronous send-receive

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

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

SPL Semantics: Schematic Statements

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

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

ℓ: **produce** x; $\hat{\ell}$: $\quad\rightarrow\quad move(\ell, \hat{\ell}) \;\wedge\; x' \neq 0 \;\wedge\; pres\big(Y - \{x\}\big)$

ℓ: **consume** y; $\hat{\ell}$: $\quad\rightarrow\quad move(\ell, \hat{\ell}) \;\wedge\; pres\big(Y\big)$

Noncritical section doesn't need to terminate.

Modeled by

$\tau_\ell \notin \mathcal{J}$

SPL Semantics: Compound Statements

ℓ: $\big[$**if** c **then** $\ell_1: S_1$ **else** $\ell_2: S_2\big]$; $\hat{\ell}$: $\rightarrow$

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

ρ_ℓ^{T}: $move(\ell, \ell_1) \;\wedge\; c \;\wedge\; pres(Y)$

ρ_ℓ^{F}: $move(\ell, \ell_2) \;\wedge\; \neg c \;\wedge\; pres(Y)$

ℓ: $\big[$**while** c **do** $[\tilde{\ell}: \tilde{S}]\big]$; $\hat{\ell}$: $\rightarrow$

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

ρ_ℓ^{T}: $move(\ell, \tilde{\ell}) \;\wedge\; c \;\wedge\; pres(Y)$

ρ_ℓ^{F}: $move(\ell, \hat{\ell}) \;\wedge\; \neg c \;\wedge\; pres(Y)$

ℓ: $\big[[\ell_1: S_1;\ \hat{\ell}_1:] \,\|\, \cdots \,\|\, [\ell_k: S_k;\ \hat{\ell}_k:]\big]$; $\hat{\ell}$: $\rightarrow$

ρ_ℓ^{E}: $move\big(\{\ell\},\ \{\ell_1, \ldots, \ell_k\}\big) \wedge pres(Y)$ (entry)

ρ_ℓ^{X}: $move\big(\{\hat{\ell}_1, \ldots, \hat{\ell}_k\},\ \{\hat{\ell}\}\big) \wedge pres(Y)$ (exit)

SPL Semantics: Grouped Statements

$$\ell\colon \langle S\rangle;\ \hat{\ell}\colon \quad \rightarrow \quad move(\ell, \hat{\ell}) \ \wedge \delta(S)$$

δ: data transformation relation:

skip;	$pres(Y)$
$\overline{u} := \overline{e}$;	$\overline{u}' = \overline{e} \ \wedge \ pres\big(Y - \{\overline{u}\}\big)$
...	
$[S_1; S_2]$;	$\exists Y'' : \ (\delta(S_1)(Y, Y'') \wedge \delta(S_2)(Y'', Y'))$

Justice and Compassion

- 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

Examples

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

$$\textbf{local } x\colon \textbf{integer where } x = 1$$

1. $P_1 :: \begin{bmatrix} \ell_0\colon \begin{bmatrix} \ell_0^a\colon \textbf{ await } x = 1 \\ \textbf{or} \\ \ell_0^b\colon \textbf{ skip} \end{bmatrix} \\ \ell_1\colon \end{bmatrix} \parallel P_2 :: \begin{bmatrix} m_0\colon \textbf{ while } \mathrm{T} \textbf{ do} \\ \quad [m_1\colon x := -x] \end{bmatrix}$ — no computation

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

3. $P_1 :: \begin{bmatrix} \ell_0\colon \textbf{ if } x = 1 \textbf{ then} \\ \quad \ell_1\colon \textbf{ skip} \\ \textbf{else} \\ \quad \ell_2\colon \textbf{ skip} \\ \ell_3\colon \end{bmatrix} \parallel P_2 :: \begin{bmatrix} m_0\colon \textbf{ while } \mathrm{T} \textbf{ do} \\ \quad [m_1\colon x := -x] \end{bmatrix}$ — no computation