CS256/winter2009—Lecture#09

Zohar Manna

Chapter 2

Invariance: Applications

Parameterized Programs

$$S:: \left[\begin{array}{l}\ell_0\text{: } \textbf{loop forever do} \\ \quad \left[\begin{array}{l}\ell_1\text{: } \textbf{noncritical} \\ \ell_2\text{: } \textbf{request } y \\ \ell_3\text{: } \textbf{critical} \\ \ell_4\text{: } \textbf{release } y\end{array}\right]\end{array}\right]$$

P^3:: [**local** y : **integer where** $y = 1$; $[S||S||S]$]

(with some renaming of labels of the S's.)

P^4:: [**local** y : **integer where** $y = 1$; $[S||S||S||S]$]

⋮

P^n:: ?

Mutual exclusion:

P^3: $\Box(\neg(at\_\ell_3 \wedge at\_m_3) \;\wedge\; \neg(at\_\ell_3 \wedge at\_k_3) \;\wedge$
$\neg(at\_m_3 \wedge \text{at\_}k_3))$

P^4: $\Box(\neg(\ldots) \;\wedge\; \ldots \;\wedge\; \neg(\ldots))$

P^n: ?

We want to deal with these programs,
i.e., programs with an arbitrary number of identical components, in a more uniform way.

Solution: parametrization

Syntax

Compound statements of variable size

cooperation: $\overset{M}{\underset{j=1}{\|}} S[j] \quad : \quad [\; S[1] \| \ldots \| S[M] \;]$

Selection: $\overset{M}{\underset{j=1}{\textbf{OR}}} S[j] \quad : \quad [\; S[1] \textbf{ or } \ldots \textbf{ or } S[M] \;]$

$S[j]$ is a parameterized statement.

In what ways can j appear in S?

- explicit variable in expression
  $\ldots := j + \ldots$
- explicit subscript in array x
  $\ldots := x[j] + \ldots$ or $x[j] := \ldots$
- implicit subscript of all local variables in $S[j]$
  z stands for $z[j]$
- implicit subscript of all labels in $S[j]$
  ℓ_3 stands for $\ell_3[j]$

`Example:` Program PAR-SUM (Fig. 2.1)

(parallel sum of squares) $M \geq 1$

$$\begin{array}{lll} \textbf{in} & M & : \textbf{integer where } M \geq 1 \\ & x & : \textbf{array } [1..M] \textbf{ of integer} \\ \textbf{out} & z & : \textbf{integer where } z = 0 \end{array}$$

$$\overset{M}{\underset{j=1}{||}} \; P[j] :: \left[\begin{array}{l} \textbf{local } y\text{: } \textbf{integer} \\ \ell_0\text{: } y := x[j] \\ \ell_1\text{: } z := z + y \cdot y \\ \ell_2\text{:} \end{array}\right]$$

$$\boxed{z \; = \; x[1]^2 + x[2]^2 + \ldots + x[M]^2}$$

Program PAR-SUM-E (Fig. 2.2)

(Explicit subscripted parameterized statements of PAR-SUM)

$$\begin{array}{lll} \textbf{in} & M & : \textbf{integer where } M \geq 1 \\ & x & : \textbf{array } [1..M] \textbf{ of integer} \\ \textbf{out} & z & : \textbf{integer where } z = 0 \end{array}$$

$$\overset{M}{\underset{j=1}{||}} \; P[j] :: \left[\begin{array}{l} \textbf{local } y[j]\text{: } \textbf{integer} \\ \ell_0[j]\text{: } y[j] := x[j] \\ \ell_1[j]\text{: } z := z + y[j] \cdot y[j] \\ \ell_2[j]\text{:} \end{array}\right]$$

We <u>write</u> the short version,
but we <u>reason</u> about this one.

Parameterized transition systems

The number M of processes is not fixed, so there is an unbounded number of transitions. To finitely represent these, we use parameterization of transition relations.

`Example:` PAR-SUM

The unbounded number of transitions associated with ℓ_0 are represented by a single transition relation using parameter j:

$$\begin{aligned} \rho_{\ell_0}[j]: \quad & \mathit{move}(\ell_0[j], \ell_1[j]) \ \wedge \\ & y'[j] = x[j] \ \wedge \\ & \mathit{pres}(\{x, z\}) \end{aligned}$$

where $j = 1 \ldots M$.

Array Operations

Arrays (explicit or implicit) are treated as variables that range over functions:

$[1 \ldots M] \ \mapsto$ integers

Representation of array operations in transition relations:

- Retrieval: $y[k]$
  to retrieve the value of the kth element of array y

- Modification: $\mathit{update}(y, k, e)$
  the resulting array agrees with y on all i, $i \neq k$, and $y[k] = e$

Properties of *update*

$$update(y,k,e)[k] = e$$

$$update(y,k,e)[j] = y[j] \text{ for } j \neq k$$

Example: PAR-SUM

The proper representation of the transition relation for $\ell_0[j]$ is

$$\rho_0[j]: \quad move(\ell_0[j],\ \ell_1[j]) \wedge$$
$$y' = update(y,\ j,\ x[j]) \wedge$$
$$pres(\{x,z\})$$

Parameterized Programs: Specification

Notation:

- $L_i \ = \ \{j \mid \ell_i[j] \in \pi\} \quad \subseteq \{1,\ldots,M\}$

  The set of indices of processes that currently reside at ℓ_i
- $N_i \ = \ |L_i|$

  The number of processes currently residing at ℓ_i

Example: $L_i \ = \ \{3,5\}$ means $\ell_i[3], \ell_i[5] \ \in \pi$
and we have $N_i = 2$

Invariant:
$\square(N_i \ \geq \ 0)$

Abbreviations:

$$L_{i_1,i_2,\ldots,i_k} = L_{i_1} \cup L_{i_2} \cup \ldots \cup L_{i_k}$$
$$L_{i..j} = L_i \cup L_{i+1} \cup \ldots \cup L_j$$
$$N_{i_1,i_2,\ldots,i_k} = |L_{i_1,i_2,\ldots,i_k}|$$
$$N_{i..j} = |L_{i..j}|$$

Parameterized Programs: Specification (Con'd)

<u>`Example:`</u> Program MPX-SEM (Fig 2.3) $M \geq 2$
(multiple mutual exclusion by semaphores)
where

$$j \oplus_M 1 = (j \bmod M) + 1 = \begin{cases} j+1 & \text{if } j < M \\ 1 & \text{if } j = M \end{cases}$$

Elaboration for $M = 2$:
Program MPX-SEM-2 (Fig 2.4)

mutual exclusion:

$$\square \underbrace{\forall i, j \in [1..M] \,.\, i \neq j \,.\, \neg(at\_\ell_3[i] \;\wedge\; at\_\ell_3[j])}_{\psi}$$

abbreviated as

$$\boxed{\square(N_3 \leq 1)}$$

i.e., the number of processes simultaneously residing at ℓ_3 is always less than or equal to 1.

Note: $\neg(at\_\ell_3[i] \wedge at\_\ell_3[j])$ can be expressed as $at\_\ell_3[i] + at\_\ell_3[j] \leq 1$.

Program MPX-SEM (Fig. 2.3)

in M: **integer where** $M \geq 2$
local y : **array** $[1..M]$ **of integer**
where $y[1] = 1,\ y[j] = 0$ for $2 \leq j \leq M$

$$\overset{M}{\underset{j=1}{\Big|\Big|}} \; P[j] :: \left[\begin{array}{l} \ell_0\text{: } \textbf{loop forever do} \\ \quad \left[\begin{array}{l} \ell_1\text{: } \textbf{noncritical} \\ \ell_2\text{: } \textbf{request } y[j] \\ \ell_3\text{: } \textbf{critical} \\ \ell_4\text{: } \textbf{release } y[j \oplus_M 1] \end{array} \right] \end{array} \right]$$

Program MPX-SEM-2 (Fig. 2.4)

local y: **array** [1..2] **of integer where** $y[1] = 1,\ y[2] = 0$

$$P[1] :: \left[\begin{array}{l} \ell_0[1]:\ \textbf{loop forever do} \\ \quad \left[\begin{array}{l} \ell_1[1]:\ \textbf{noncritical} \\ \ell_2[1]:\ \textbf{request}\ y[1] \\ \ell_3[1]:\ \textbf{critical} \\ \ell_4[1]:\ \textbf{release}\ y[2] \end{array}\right] \end{array}\right]$$

$$||$$

$$P[2] :: \left[\begin{array}{l} \ell_0[2]:\ \textbf{loop forever do} \\ \quad \left[\begin{array}{l} \ell_1[2]:\ \textbf{noncritical} \\ \ell_2[2]:\ \textbf{request}\ y[2] \\ \ell_3[2]:\ \textbf{critical} \\ \ell_4[2]:\ \textbf{release}\ y[1] \end{array}\right] \end{array}\right]$$

Parameterized Programs: Verification

Objective: prove $\{\varphi\}\tau[i]\{\varphi\}$ in a uniform way for all $i \in [1..M]$

Example: Program MPX-SEM (Fig 2.3) $M \geq 2$

Prove mutual exclusion:

$$\square(\underbrace{N_3 \leq 1}_{\varphi})$$

The assertion φ is not inductive, therefore we prove the invariance of

$$\varphi_1:\quad \forall j\,.\,y[j]\ \geq\ 0$$

$$\varphi_2:\quad \left(N_{3,4} + \sum_{j=1}^{M} y[j]\right) =\ 1$$

where $N_{3,4}$ = Number of processes currently residing at ℓ_3 or at ℓ_4

Example: Program MPX-SEM (Con't)

Then φ can be deducted by monotonicity:

$$\varphi_1 \ \wedge \ \varphi_2 \ \rightarrow \ \underbrace{N_3 \leq 1}_{\varphi}$$

since

$$N_3 \ \leq \ N_{3,4} \ \underset{\varphi_2}{=} \ 1 - \sum_{j=1}^{M} y[j] \ \underset{\varphi_1}{\leq} \ 1$$

- Proof of $\square(\underbrace{\forall j \,.\, y[j] \geq 0}_{\varphi_1})$

B1:

$$\underbrace{\ldots \ \wedge \ y[1] = 1 \ \wedge \ (\forall j \,.\, 2 \leq j \leq M \,.\, y[j] = 0)}_{\Theta} \rightarrow \underbrace{\forall j \,.\, y[j] \geq 0}_{\varphi_1}$$

Note: $\forall j \,.\, y[j] \geq 0$ stands for $\forall j . i \leq j \leq M \,.\, y[j] \geq 0$

Example: Program MPX-SEM (Con't)

B2:
The only transitions that interfere with φ_1 are $\tau_{\ell_2}[i]$ and $\tau_{\ell_4}[i]$.

$$\rho_{\ell_2}[i]\text{:} \ \mathit{move}(\ell_2[i], \ell_3[i]) \ \wedge \ y[i] > 0 \ \wedge$$
$$y' = \mathit{update}(y, i, y[i]{-}1)$$

$$\rho_{\ell_4}[i]\text{:} \ \mathit{move}(\ell_4[i], \ell_0[i]) \ \wedge$$
$$y' = \mathit{update}(y, i \oplus_M 1, y[i \oplus_M 1]{+}1)$$

$\rho_{\ell_2}[i]$ implies

$$y[i] > 0 \ \wedge y'[i] = y[i] - 1 \ \wedge \ \forall j \,.\, j \neq i \,.\, y'[j] = y[j]$$

$\rho_{\ell_4}[i]$ implies

$$y'[i \oplus_M 1] = y[i \oplus_M 1] + 1 \ \wedge$$
$$\forall j (j \neq i \oplus_M 1) \ y'[j] = y[j]$$

We therefore have

$$\underbrace{\forall j \,.\, y[j] \geq 0}_{\varphi_1} \wedge \left\{ \begin{array}{l} \rho_{\ell_2}[i] \\ \rho_{\ell_4}[i] \end{array} \right\} \ \rightarrow \ \underbrace{\forall j \,.\, y'[j] \geq 0}_{\varphi_1'}$$

- Proof of $\square \underbrace{\left(N_{3,4} + \left(\sum_{j=1}^{M} y[j]\right) = 1\right)}_{\varphi_2}$

B1:

$$\underbrace{\left(\begin{array}{l} \pi = \{\ell_0[1], \ldots, \ell_0[M]\} \wedge \\ y[1] = 1 \ \wedge \ (\forall j \, . \, 2 \leq j \leq M \, . \, y[j] = 0) \end{array}\right)}_{\Theta}$$

$$\rightarrow \underbrace{N_{3,4} + \left(\sum_{j=1}^{M} y[j]\right) = 1}_{\varphi_2}$$

B2: Verification conditions:

$\rho_{\ell_2}[i]$ implies:

$$N'_{3,4} = N_{3,4} + 1$$

$$\left(\sum_{j=1}^{M} y'[i]\right) = \left(\sum_{j=1}^{M} y[i]\right) - 1$$

$\rho_{\ell_4}[i]$ implies:

$$N'_{3,4} = N_{3,4} - 1$$

$$\left(\sum_{j=1}^{M} y'[i]\right) = \left(\sum_{j=1}^{M} y[i]\right) + 1$$

Therefore

$$\underbrace{N_{3,4} + \left(\sum_{j=1}^{M} y[i]\right) = 1}_{\varphi_2} \wedge \left\{\begin{array}{l} \rho_{\ell_2}[i] \\ \rho_{\ell_4}[i] \end{array}\right\}$$

$$\rightarrow \underbrace{N'_{3,4} + \left(\sum_{j=1}^{M} y'[i]\right) = 1}_{\varphi'_2}$$

Parameterized Programs: Examples

Example: READERS-WRITERS (Fig 2.11)
(readers-writers with generalized semaphores)

where

$$\textbf{request}\ (y,c)\ =\ \langle \textbf{await}\ y \geq c;\ y := y - c\rangle$$

$$\textbf{release}\ (y,c)\ =\ \langle y := y + c\rangle$$

$$\Box\ \underbrace{\forall i,j \in [1..M]\,.\, i \neq j\,.\, at\_\ell_6[i]\ \rightarrow\ \neg(at\_\ell_6[j]\ \vee\ at\_\ell_3[j])}_{\psi}$$

- φ_1 and φ_2 are inductive

  φ_1: $\quad y \geq 0$

  φ_2: $\quad N_{3,4} + M \cdot N_{6,7} + y = M$

- Therefore

  $$N_{6,7} > 0 \underset{\varphi_1,\varphi_2}{\rightarrow} (N_{6,7} = 1\ \wedge\ N_{3,4} = 0)$$

  Thus,

  $$\Box\,\psi$$

Program READ-WRITE(Fig. 2.11)

in $\quad M$: **integer where** $M \geq 1$
local y : **integer where** $y = M$

$$\mathop{\Big|\Big|}_{i=1}^{M} P[i] :: \left[\begin{array}{l} \ell_0\text{: \textbf{loop forever do}} \\ \quad \left[\begin{array}{l} \ell_1\text{: \textbf{noncritical}} \\ \left[\begin{array}{l} R :: \left[\begin{array}{l} \ell_2\text{: \textbf{request} } (y,1) \\ \ell_3\text{: \textbf{read}} \\ \ell_4\text{: \textbf{release} } (y,1) \end{array}\right] \\ \quad \textbf{or} \\ W :: \left[\begin{array}{l} \ell_5\text{: \textbf{request} } (y,M) \\ \ell_6\text{: \textbf{write}} \\ \ell_7\text{: \textbf{release} } (y,M) \end{array}\right] \end{array}\right] \end{array}\right] \end{array}\right]$$

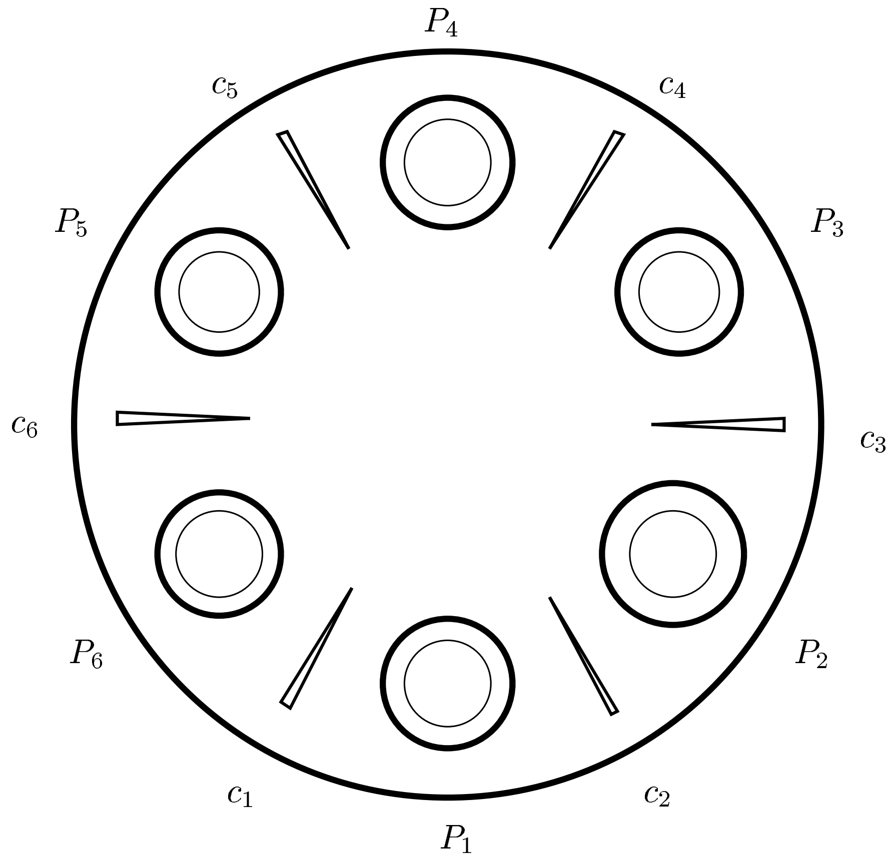
`Example:` **The Dining Philosophers Problem**

(multiple resource allocation)

Fig 2.14

- M philosophers are seated at a round table
- Each philosopher alternates between a "thinking" phase and "eating" phase
- M chopsticks, one between every two philosophers
- A philosopher needs 2 chopsticks (left & right) to eat

Dining philosophers setup (Fig. 2.14)



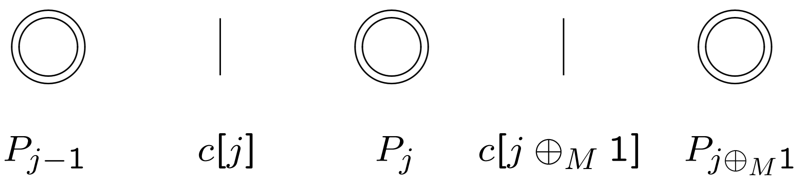
Program DINE (Fig. 2.15)
(A simple solution to the dining philosophers problem)

Philosopher P_i	-	process $P[i]$
"thinking" phase	-	noncritical
"eating" phase	-	critical

For philosopher j,

- $c[j]$ represents availability of left chopstick
  ($c[j] = 1$ iff chopstick is available)

- $c[j \oplus_M 1]$.............right chopstick



Program DINE (Fig. 2.15)

in M: **integer where** $M \geq 2$
local c : **array** $[1..M]$ **of integer where** $c = 1$

$$\overset{M}{\underset{j=1}{||}} P[j] :: \left[\begin{array}{l} \ell_0\text{: } \textbf{loop forever do} \\ \quad \left[\begin{array}{l} \ell_1\text{: } \textbf{noncritical} \\ \ell_2\text{: } \textbf{request } c[j] \\ \ell_3\text{: } \textbf{request } c[j \oplus_M 1] \\ \ell_4\text{: } \textbf{critical} \\ \ell_5\text{: } \textbf{release } c[j] \\ \ell_6\text{: } \textbf{release } c[j \oplus_M 1] \end{array}\right] \end{array}\right]$$

Specification: Chopstick Exclusion

$$\Box \underbrace{\forall j \in [1..M] \,.\, \neg(at\_\ell_4[j] \ \wedge \ at\_\ell_4[j \oplus_M 1])}_{\psi}$$

Mutual exclusion between every two adjacent philosophers

Proof:

- φ_0 and φ_1 are inductive

$$\varphi_0\colon \ \forall j \in [1..M] \,.\, c[j] \ \geq \ 0$$

$$\varphi_1\colon \ \forall j \in [1..M] \,.\, at\_\ell_{4..6}[j] \ + \ at\_\ell_{3..5}[j \oplus_M 1] \ + \ c[j \oplus_M 1] \ = \ 1$$

- Then,

$$at\_\ell_4[j] + at\_\ell_4[j \oplus_M 1]$$

$$\leq at\_\ell_{4..6}[j] + at\_\ell_{3..5}[j \oplus_M 1]$$

$$\underset{\varphi_1}{=} 1 - c[j \oplus_M 1] \ \underset{\varphi_0}{\leq} \ 1$$

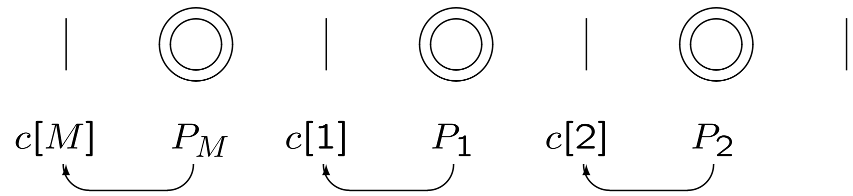
Chopstick Exclusion OK

Problem: possible deadlock ("starvation")

$P[1]$ $\quad \ell_2$: **request** $c[1]$; $\quad \ell_3$: **request** $c[2]$
$\qquad\qquad\qquad\qquad\qquad\qquad\quad\uparrow$

.
.
.

$P[M]$ $\quad \ell_2$: **request** $c[M]$; $\quad \ell_3$: **request** $c[1]$
$\qquad\qquad\qquad\qquad\qquad\qquad\quad\uparrow$



Solution: One Philosopher Excluded
(keeping the symmetry)

- Two-room philosophers' world (Fig 2.18)

  Philosophers are "thinking" at the library
  "eating" at the dining hall

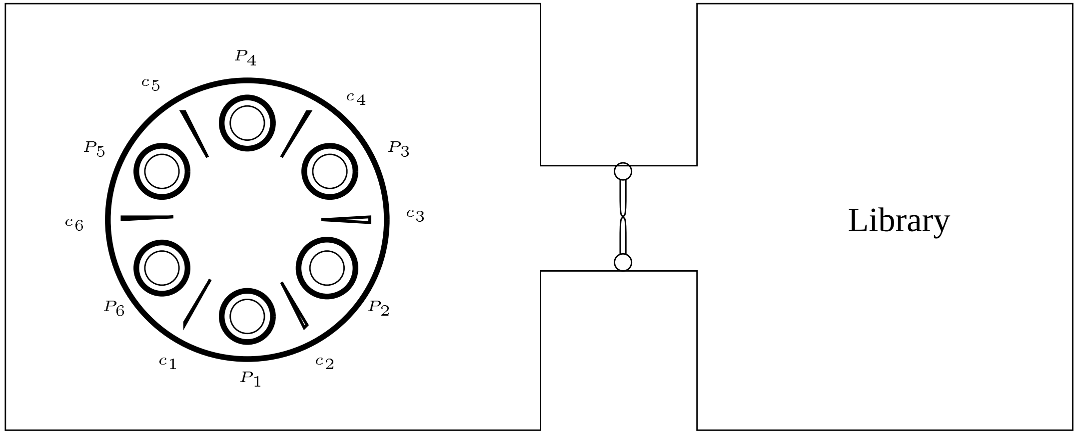
  When a philosopher finishes "eating"
  he returns to the library to "think"

- Program DINE-EXCL (Fig 2.17)

  Additional semaphore variable r
  "door keeper" (initally $r = M-1$)

  No more than $M-1$ philosophers are
  admitted to the dining hall at the same time.

Two-room philosopher's world (Fig. 2.18)



Program DINE-EXCL (Fig. 2.17)

$$
\begin{array}{ll}
\textbf{in} & M\colon \textbf{integer where } M \geq 2 \\
\textbf{local } c & :\textbf{array } [1..M] \textbf{ integer where } c = 1 \\
\phantom{\textbf{local }} r & :\textbf{integer where } r = M - 1
\end{array}
$$

$$
\overset{M}{\underset{j=1}{\Big|\Big|}} P[j] :: \left[\begin{array}{l}
\ell_0\colon \textbf{loop forever do} \\
\qquad \left[\begin{array}{l}
\ell_1\colon \textbf{noncritical} \\
\ell_2\colon \textbf{request } r \\
\ell_3\colon \textbf{request } c[j] \\
\ell_4\colon \textbf{request } c[j \oplus_M 1] \\
\ell_5\colon \textbf{critical} \\
\ell_6\colon \textbf{release } c[j] \\
\ell_7\colon \textbf{release } c[j \oplus_M 1] \\
\ell_8\colon \textbf{release } r
\end{array}\right]
\end{array}\right]
$$

Properties of DINE-EXCL:

- chopstick exclusion
  A safety property (in text)

- starvation-free
  progress (next book)

- accessibility $\ell_2[j] \Rightarrow \Diamond\, \ell_5[j]$
  progress (next book)

Chapter 3

Precedence

Proving Precedence Properties

nested waiting-for formulas

are of the form

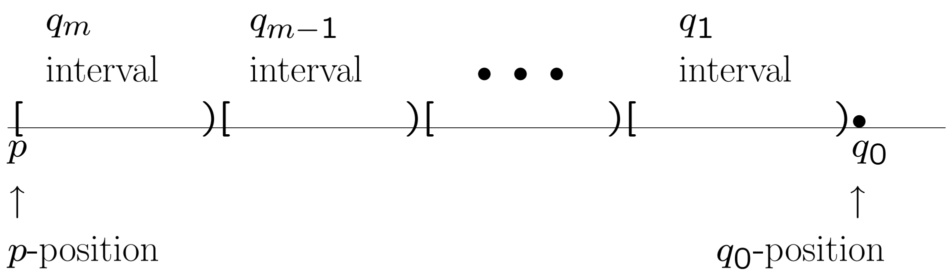
$$p \;\Rightarrow\; q_m \,\mathcal{W}\, (q_{m-1} \;\cdots\; (q_1 \,\mathcal{W}\, q_0) \ldots)$$

also written

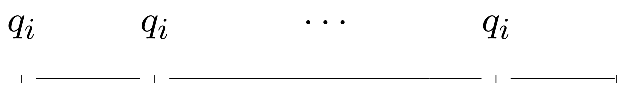
$$\boxed{p \;\Rightarrow\; q_m \,\mathcal{W}\, q_{m-1} \;\cdots\; q_1 \,\mathcal{W}\, q_0}$$

for assertions $p, q_0, q_1, \ldots, q_m$.

Models that satisfy these formulas

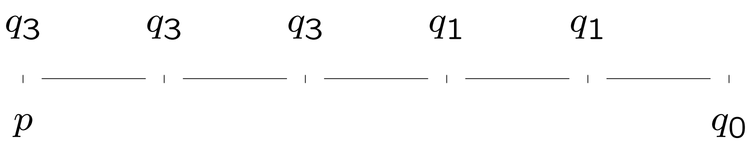


q_i-interval

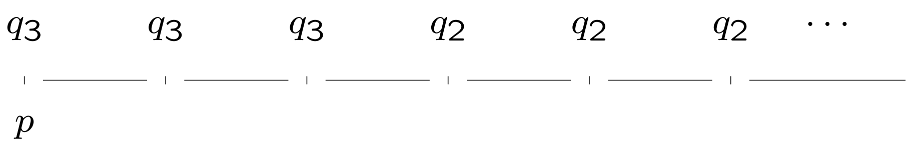


- May be empty

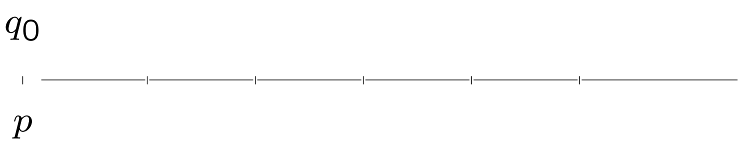
  e.g. $p \Rightarrow q_3 \mathcal{W} q_2 \mathcal{W} q_1 \mathcal{W} q_0$



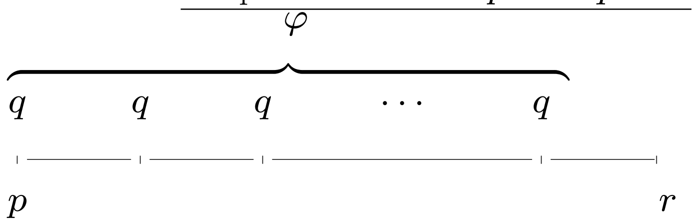
- May extend to infinity



Note: The following is OK



Simple Precedence: $p \Rightarrow q \mathcal{W} r$



can be reduced to first-order VCs by verification rule WAIT:

> **Rule wait** (general waiting-for)
>
> For assertions p, q, r, φ
>
> W1. $p \rightarrow \varphi \vee r$
>
> W2. $\varphi \rightarrow q$
>
> W3. $\{\varphi\} \mathcal{T} \{\varphi \vee r\}$
>
> ---
>
> $p \Rightarrow q \mathcal{W} r$

Recall: To show $P \Vdash \{\varphi\} \ \mathcal{T} \ \{\varphi \vee r\}$,
we have to show that for every $\tau \in \mathcal{T}$

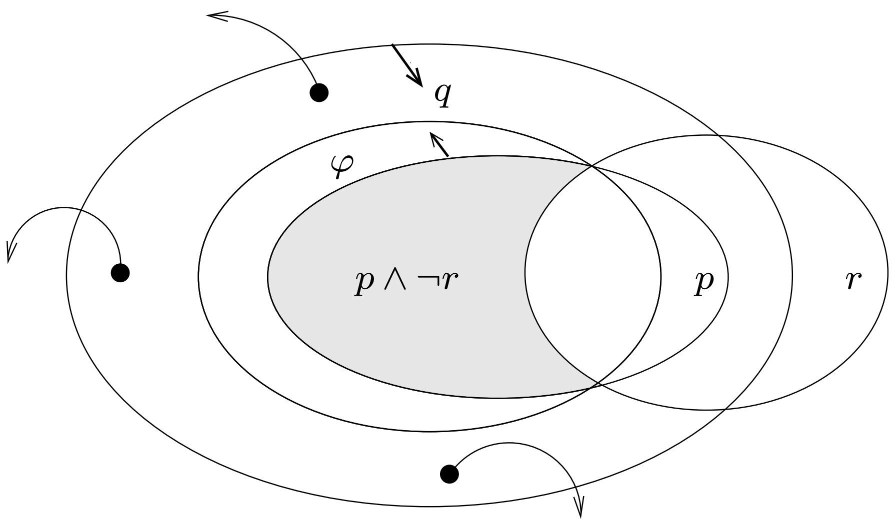
$$\rho_\tau \wedge \varphi \rightarrow \varphi' \vee r'$$

is P-state valid.

Intermediate Assertion φ

W1. $p \rightarrow \varphi \vee r$ "φ weakens $p \wedge \neg r$"
i.e., $p \wedge \neg r \rightarrow \varphi$

W2. $\varphi \rightarrow q$ "φ strengthens q"



Example: **Program mux-pet1 (Fig. 3.4)**

We proved mutual exclusion

ψ_1: $\square \neg(at\_\ell_4 \ \wedge \ at\_m_4)$

Using invariants

χ_0: $s = 1 \vee s = 2$

χ_1: $y_1 \leftrightarrow at\_\ell_{3..5}$

χ_2: $y_2 \leftrightarrow at\_m_{3..5}$

χ_3: $at\_\ell_3 \wedge at\_m_4 \rightarrow y_2 \wedge s = 1$

χ_4: $at\_\ell_4 \wedge at\_m_3 \rightarrow y_1 \wedge s = 2$

Example: Program mux-pet1 (Fig. 3.4)

(Peterson's Algorithm for mutual exclusion)

$$
\begin{array}{ll}
\textbf{local} & y_1, y_2\text{: } \textbf{boolean} \quad \textbf{where } y_1 = \mathrm{F}, y_2 = \mathrm{F} \\
 & s \quad\;\; : \textbf{integer} \quad \textbf{where } s = 1
\end{array}
$$

$$
P_1 :: \quad \ell_0 : \textbf{loop forever do} \left[\begin{array}{ll}
\ell_1 : & \textbf{noncritical} \\
\ell_2 : & (y_1,\ s) := (\mathrm{T},\ 1) \\
\ell_3 : & \textbf{await } (\neg y_2) \vee (s \neq 1) \\
\ell_4 : & \textbf{critical} \\
\ell_5 : & y_1 := \mathrm{F}
\end{array}\right]
$$

$$||$$

$$
P_2 :: \quad m_0 : \textbf{loop forever do} \left[\begin{array}{ll}
m_1 : & \textbf{noncritical} \\
m_2 : & (y_2,\ s) := (\mathrm{T},\ 2) \\
m_3 : & \textbf{await } (\neg y_1) \vee (s \neq 2) \\
m_4 : & \textbf{critical} \\
m_5 : & y_2 := \mathrm{F}
\end{array}\right]
$$

We want to prove simple precedence

$$\psi_2 : \quad \underbrace{at\_\ell_3 \wedge at\_m_{0..2}}_{p} \Rightarrow \underbrace{\neg at\_m_4}_{q} \ \mathcal{W} \ \underbrace{at\_\ell_4}_{r}$$

We try to find an assertion φ such that W1 – W3 of rule WAIT hold

Let

$$\varphi : \ at\_\ell_3 \wedge (at\_m_{0..2} \vee (at\_m_3 \wedge s = 2))$$

W1:

$$\underbrace{at\_\ell_3 \ \wedge \ at\_m_{0..2}}_{p} \ \rightarrow$$

$$\underbrace{at\_\ell_3 \ \wedge \ (at\_m_{0..2} \ \vee \ \cdots)}_{\varphi} \ \vee \ \underbrace{\cdots}_{r}$$

W2:

$$\underbrace{\cdots \ \wedge \ (at\_m_{0..2} \ \vee \ (at\_m_3 \ \wedge \ \cdots))}_{\varphi} \ \rightarrow \ \underbrace{\neg at\_m_4}_{q}$$

W3:

$$\rho_{\mathcal{T}} \wedge \underbrace{at\_\ell_3 \ \wedge \ (at\_m_{0..2} \ \vee \ (at\_m_3 \ \wedge \ s = 2))}_{\varphi} \ \rightarrow$$

$$\underbrace{at'\_\ell_3 \ \wedge \ (at'\_m_{0..2} \ \vee \ (at'\_m_3 \ \wedge \ s' = 2))}_{\varphi'} \ \vee \ \underbrace{at'\_\ell_4}_{r'}$$

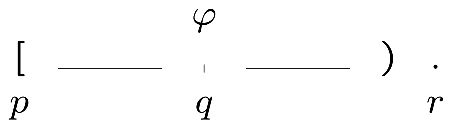
Check:

ℓ_3, m_2: OK

m_3: disabled (with the help of the invariant $at\_\ell_{3..5} \leftrightarrow y_1$, we have $y_1 = \text{T}$).

Proving precedence properties:

Systematic derivation of intermediate assertions



Recall:

Rule WAIT (general waiting-for)

For assertions p, q, r, φ

W1. $p \ \rightarrow \ \varphi \vee r$

W2. $\varphi \ \rightarrow \ q$

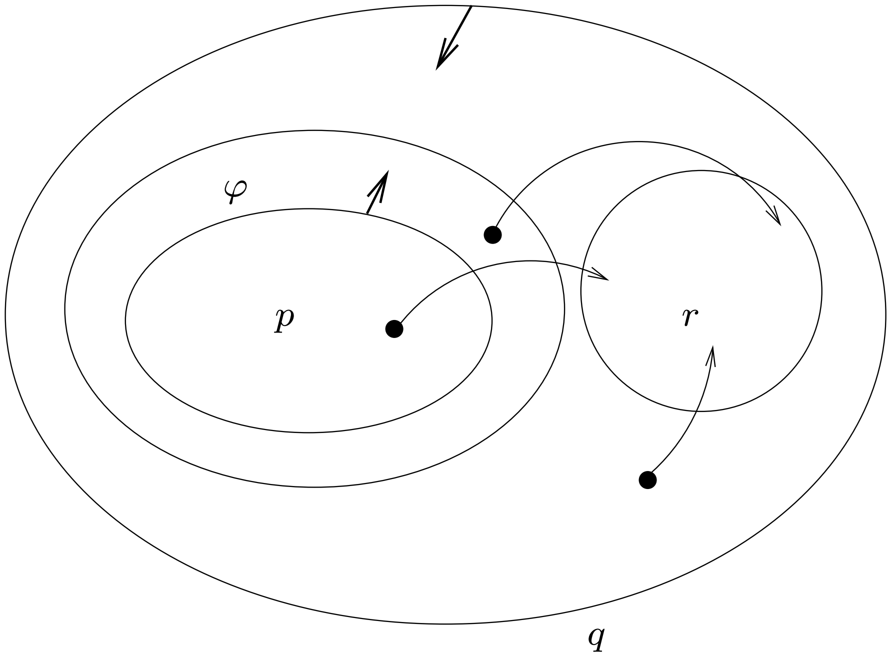
W3. $\{\varphi\}\mathcal{T}\{\varphi \vee r\}$

$p \ \Rightarrow \ q \, \mathcal{W} \, r$

How to find φ?

Escape Transition

Transition that leads to r-state.



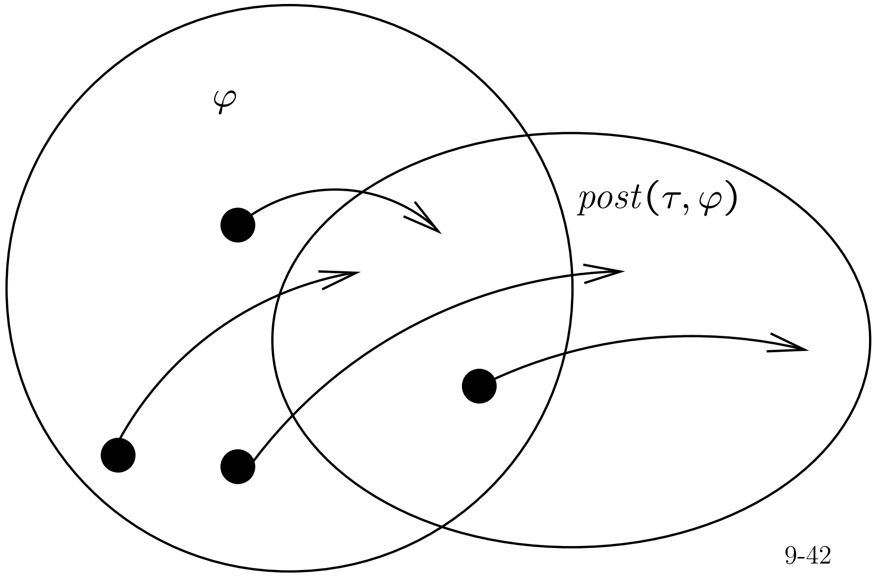
Forward propagation

Weaken $p \wedge \neg r$ until it becomes an assertion preserved under all nonescape transitions.

Based on postcondition:

$$\Psi(V) = post(\tau, \varphi): \quad \exists V^0 . \varphi(V^0) \wedge \rho_\tau(V^0, V)$$

$post(\tau, \varphi)$ characterizes all states that are τ-successors of a φ-state.



Example: Postcondition

$V = \{x, y\}$,

$\rho_\mathcal{T} : x' = x + y \wedge y' = x$,

$\Phi : x = y$

Then $post(\tau, \Phi)$ is given by

$$\exists x^0, y^0 : \underbrace{x^0 = y^0}_{\Phi(V^0)} \wedge \underbrace{x = x^0 + y^0 \wedge y = x^0}_{\rho_\mathcal{T}(V^0, V)},$$

which can be simplified to

$\Psi : x = y + y$.

Forward Propagation: Algorithm

Φ_t - characterizes all states that can be reached from a $(p \wedge \neg r)$-state without taking an escape transition.

1. $\Phi_0 \;=\; p \wedge \neg r$
2. Repeat

   $\Phi_{k+1} = \Phi_k \vee post(\tau, \Phi_k)$

   for any non-escape transition τ

   Until

   $post(\tau, \Phi_t) \rightarrow \Phi_t$ [may use invariants]

   for all non-escape transitions τ

If this terminates (it may not), Φ_t is a good assertion to be used in rule WAIT.
Satisifies W1, W3, but check W2.

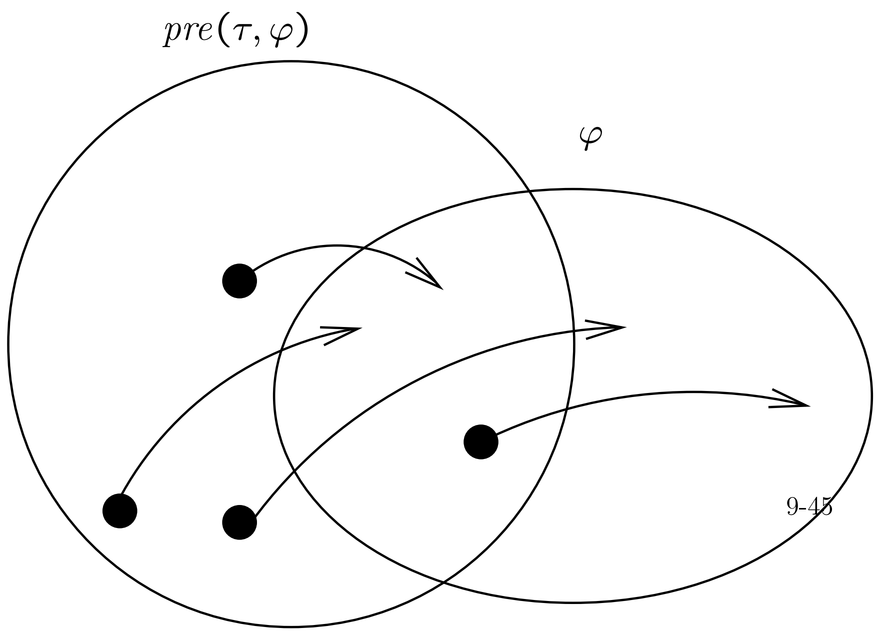
Backward propagation

Strengthen q until it becomes an assertion preserved under all nonescape transitions.

Based on precondition:

$$pre(\tau, \varphi): \ \forall V' . \rho_\tau(V, V') \rightarrow \varphi(V')$$

$pre(\tau, \varphi)$ characterizes all states all of whose τ-successors satisfy φ.



Example: Precondition

For Peterson's Algorithm, consider

$$\Gamma_0 : \underbrace{\neg at\_m_4}$$

and calculate $pre(m_3, \Gamma_0)$:

$$\forall V' : \underbrace{at\_m_3 \wedge (\neg y_1 \vee s \neq 2) \wedge at\_m_4{}' \wedge \cdots}_{\rho_{m_3}(V,V')} \rightarrow \underbrace{\neg at\_m_4{}'}_{\Gamma_0(V')}.$$

P-equivalent to

$$at\_m_3 \rightarrow (y_1 \wedge s = 2).$$

Backward Propagation: Algorithm

Γ_f - characterizes all states that can reach a q-state without taking an escape transition

1. $\Gamma_0 = q$
2. Repeat

   $\Gamma_{k+1} = \Gamma_k \wedge pre(\tau, \Gamma_k)$

   for any non-escape transition τ

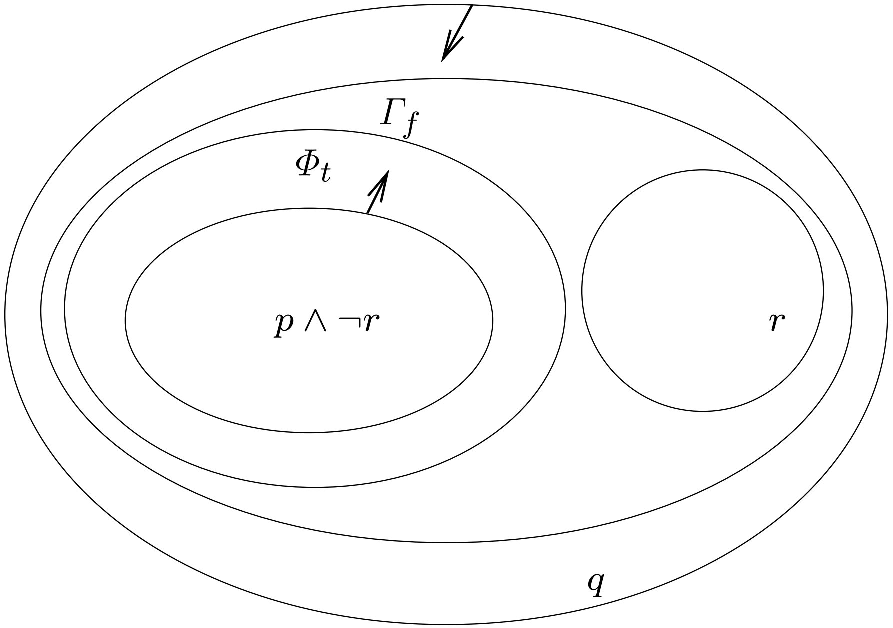
   Until

   $\Gamma_f \rightarrow pre(\tau, \Gamma_f)$ [may use invariants]

   for all non-escape transitions τ

If this terminates (it may not), Γ_f is a good assertion to be used in rule WAIT.
Satisfies W2, W3, but check W1.

Backward vs. Forward



If $p \Rightarrow q\ \mathcal{W}\ r$ is P-valid

$$\Phi_t \rightarrow \Gamma_f$$

is P-state valid.

Example: Program mux-pet1 (Fig. 3.4)
(Peterson's Algorithm for mutual exclusion)

$$
\begin{array}{l}
\textbf{local}\ \ y_1, y_2 : \ \textbf{boolean}\ \ \textbf{where}\ y_1 = \mathrm{F}, y_2 = \mathrm{F} \\
\qquad\quad\ s \quad\ : \ \textbf{integer}\ \ \ \textbf{where}\ s = 1 \\
\\
\qquad \ell_0 : \ \textbf{loop forever do} \\
P_1 :: \qquad\qquad \left[\begin{array}{ll}
\ell_1 : & \textbf{noncritical} \\
\ell_2 : & (y_1,\ s) := (\mathrm{T},\ 1) \\
\ell_3 : & \textbf{await}\ (\neg y_2) \vee (s \neq 1) \\
\ell_4 : & \textbf{critical} \\
\ell_5 : & y_1 := \mathrm{F}
\end{array}\right] \\
|| \\
\qquad m_0 : \ \textbf{loop forever do} \\
P_2 :: \qquad\qquad \left[\begin{array}{ll}
m_1 : & \textbf{noncritical} \\
m_2 : & (y_2,\ s) := (\mathrm{T},\ 2) \\
m_3 : & \textbf{await}\ (\neg y_1) \vee (s \neq 2) \\
m_4 : & \textbf{critical} \\
m_5 : & y_2 := \mathrm{F}
\end{array}\right]
\end{array}
$$

Example: Forward Propagation

$$\underbrace{at\_\ell_3 \wedge at\_m_{0..2}}_{p} \Rightarrow \underbrace{\neg at\_m_4}_{q}\ \mathcal{W}\ \underbrace{at\_\ell_4}_{r}$$

Start with

$$\Phi_0 :\ \underbrace{at\_\ell_3 \wedge at\_m_{0..2}}_{p}.$$

and calculate $post(m_2, \Phi_0)$:

$$\exists \underbrace{(\pi^0, y_1^0, y_2^0, s^0)}_{V^0} :\ \underbrace{(at\_\ell_3)^0 \wedge (at\_m_{0..2})^0}_{\Phi_0(V^0)} \wedge$$
$$\underbrace{(at\_m_2)^0 \wedge at\_m_3 \wedge ((at\_\ell_3)^0 \leftrightarrow at\_\ell_3) \wedge s = 2 \wedge \cdots}_{\rho_{m_2}(V^0, V)}$$

P-equivalent to

$$\Psi_1 : at\_\ell_3 \wedge at\_m_3 \wedge s = 2,$$

using the invariant $\varphi_1 :\ y_1 \leftrightarrow at\_\ell_{3..5}$.

Thus,

$$\Phi_1 :\ \underbrace{at\_\ell_3 \wedge at\_m_{0..2}}_{\Phi_0} \vee \underbrace{at\_\ell_3 \wedge at\_m_3 \wedge s = 2}_{\Psi_1},$$

Example: Forward Propagation (cont.)

i.e.,

$$\boxed{at\_\ell_3 \ \wedge\ (at\_m_{0..2} \ \vee\ (at\_m_3 \wedge s = 2))}$$

Φ_1 is preserved under all transitions except the escape transition ℓ_3, so the process converges.

Example: Backward Propagation

Start with

$$\Gamma_0 :\ \underbrace{\neg at\_m_4}_{q}.$$

We calculated $pre(m_3, \Gamma_0)$ above, which is P-equivalent to

$$\Delta_1 :\ at\_m_3 \rightarrow (y_1 \wedge s = 2).$$

Thus,

$$\Gamma_1 :\ \underbrace{\neg at\_m_4}_{\Gamma_0} \ \wedge\ \underbrace{at\_m_3 \rightarrow (y_1 \wedge s = 2)}_{\Delta_1}.$$

Consider transition τ_{m_2}, and calculate $pre(m_2, \Gamma_1)$:

$$\forall V' :\ \underbrace{at\_m_2 \wedge at\_m_3{}' \wedge y_1' = y_1 \wedge s' = 2 \wedge \cdots}_{\rho_{m_2}}$$
$$\rightarrow \underbrace{\neg at\_m_4{}' \ \wedge\ (at\_m_3{}' \rightarrow (y_1' \wedge s' = 2))}_{\Gamma_1'}.$$

P-equivalent to

$$\Delta_2 :\ at\_m_2 \rightarrow y_1.$$

Example: Backward Propagation (Cont'd)

Thus,

$$\Gamma_2 : \neg at\_m_4 \land (at\_m_3 \to s = 2) \land (at\_m_{2,3} \to y_1).$$

Considering transitions τ_{m_1}, τ_{m_0}, and τ_{m_5} leads to the following sequence:

$$\Gamma_3 : \neg at\_m_4 \land (at\_m_3 \to s = 2) \land (at\_m_{1..3} \to y_1)$$

$$\Gamma_4 : \neg at\_m_4 \land (at\_m_3 \to s = 2) \land (at\_m_{0..3} \to y_1)$$

$$\Gamma_5 : \neg at\_m_4 \land (at\_m_3 \to s = 2) \land (at\_m_{0..3,5} \to y_1)$$

By the control invariant $at\_m_{0..5}$, Γ_5 can be simplified to

$$\Gamma_5 : \neg at\_m_4 \land (at\_m_3 \to s = 2) \land y_1.$$

Example: Backward Propagation (Cont'd)

Calculating $pre(\ell_5, \Gamma_5)$,

$$\forall V' : \underbrace{at\_\ell_5 \land y_1' = \text{F} \land \cdots}_{\rho_{\ell_5}} \to \underbrace{\neg at\_m_4{}' \land (at\_m_3{}' \to s' = 2) \land y_1'}_{\Gamma_5'},$$

gives

$$\Delta_6 : at\_\ell_5 \to \text{F}.$$

Propagating $\Gamma_5 \land \Delta_6$ via τ_{ℓ_4} gives

$$\Delta_7 : at\_\ell_4 \to \text{F}.$$

Hence,

$$\boxed{\Gamma_7 : \neg at\_m_4 \land (at\_m_3 \to s = 2) \land at\_\ell_3,}$$

using the invariant $\varphi_1 : y_1 \leftrightarrow at\_\ell_{3..5}$ for simplifications. The assertion is preserved under all but the escape transitions, ending the process.