Parameterized Programs

\[ S : \]
\[ \ell_0: \text{loop forever} \]
\[ \ell_1: \text{noncritical} \]
\[ \ell_2: \text{request } y \]
\[ \ell_3: \text{critical} \]
\[ \ell_4: \text{release } y \]

\[ P^3: [\text{local } y: \text{integer where } y = 1; [S||S||S] \] 
(with some renaming of labels of the \( S \)'s.)

\[ P^4: [\text{local } y: \text{integer where } y = 1; [S||S||S||S] \] 

\[ P^n: ? \]

Mutual exclusion:

\[ P^3: \square (\neg(at_{-\ell_3} \land at_{-m_3}) \land \neg(at_{-\ell_3} \land at_{-k_3}) \land \neg(at_{-m_3} \land at_{-k_3})) \]

\[ P^4: \square (\neg(\ldots) \land \ldots \land \neg(\ldots)) \]

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:
\[ \bigwedge_{j=1}^{M} S[j] : [S[1]|\ldots||S[M]] \]

Selection:
\[ \bigvee_{j=1}^{M} S[j] : [S[1] \text{ or } \ldots \text{ or } S[M]] \]

\( S[j] \) is a parameterized statement.

In what ways can \( j \) appear in \( S \)?

- explicit variable in expression
  \[ \ldots := j + \ldots \]

- explicit subscript in array \( x \)
  \[ \ldots := x[j] + \ldots \text{ or } x[j] := \ldots \]

- implicit subscript of all local variables in \( S[j] \)
  \( z \) stands for \( z[j] \)

- implicit subscript of all labels in \( S[j] \)
  \( \ell_3 \) stands for \( \ell_3[j] \)
Program PAR-SUM-E (Fig. 2.2)
(Explicit subscripted parameterized statements of PAR-SUM)

in M: integer where M ≥ 1
x : array [1..M] of integer
out z : integer where z = 0

\[ M \bigg\| j=1 \bigg\| P[j] :: \]  
local y: integer \]  
\[ \ell_0: y := x[j] \]  
\[ \ell_1: z := z + y \cdot y \]  
\[ \ell_2: \]

\[ z = x[1]^2 + x[2]^2 + \ldots + x[M]^2 \]

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 \( \ell_0 \) are represented by a single transition relation using parameter j:

\[ \rho_{\ell_0[j]}: \text{move}(\ell_0[j], \ell_1[j]) \land \]  
\[ y'[j] = x[j] \land \]  
\[ \text{pres} \{(x, z)\} \]  

where \( j = 1 \ldots M \).

Array Operations

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

\[ [1 \ldots M] \mapsto \text{integers} \]

Representation of array operations in transition relations:

- **Retrieval**: \( y[k] \) 
  to retrieve the value of the \( k \)th element of array \( y \)

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

\[
\text{update}(y, k, e)[k] = e
\]

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

**Example: PAR-SUM**

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

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

**Parameterized Programs: Specification**

**Notation:**

- \(L_i = \{j \mid \ell_i[j] \in \pi\}\) \subseteq \{1, \ldots, M\}
  - The set of indices of processes that currently reside at \(\ell_i\)
- \(N_i = |L_i|\)
  - The number of processes currently residing at \(\ell_i\)

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

**Invariant:**

\(\Box(N_i \geq 0)\)

**Abbreviations:**

\[
\begin{align*}
L_{i_1, i_2, \ldots, i_k} &= L_{i_1} \cup L_{i_2} \cup \ldots \cup L_{i_k} \\
L_{i..j} &= L_i \cup L_{i+1} \cup \ldots \cup L_j \\
N_{i_1, i_2, \ldots, i_k} &= |L_{i_1, i_2, \ldots, i_k}| \\
N_{i..j} &= |L_{i..j}|
\end{align*}
\]

**Program mpx-sem (Fig. 2.3)**

**Example:** Program mpx-sem (Fig. 2.3) \(M \geq 2\)

(multiple mutual exclusion by semaphores)

where

\[
\begin{align*}
\psi & = (j \oplus_M 1 = (j \mod M) + 1 = \left\{ \begin{array}{ll} 
  j + 1 & \text{if } j < M \\
  1 & \text{if } j = M 
\end{array} \right. \\
\end{align*}
\]

Elaboration for \(M = 2\):

Program mpx-sem-2 (Fig 2.4)

\[
\begin{align*}
\Box(N_3 \leq 1)
\end{align*}
\]

i.e., the number of processes simultaneously residing at \(\ell_3\) is always less than or equal to 1.

Note: \(\neg(at_{\ell_3[i]} \land at_{\ell_3[j]})\) can be expressed as \(at_{\ell_3[i]} + at_{\ell_3[j]} \leq 1\).
Example: Program mpx-sem (Con’t)

Then ϕ can be deducted by monotonicity:

\[ \phi_1 \land \phi_2 \rightarrow N_3 \leq 1 \]

since

\[ N_3 \leq N_3,4 = ... M \cdot y[j] = 0 \]

Example: Program mpx-sem-2 (Fig 2.4)

Program mpx-sem-2 (Fig. 2.4)


\[
\begin{align*}
P[1] &:: [\ell_0[1]: \text{loop forever do}] \\
&\quad [\ell_1[1]: \text{noncritical}] \\
&\quad [\ell_2[1]: \text{request } y[1]] \\
&\quad [\ell_3[1]: \text{critical}] \\
&\quad [\ell_4[1]: \text{release } y[2]] \\
\| \\
P[2] &:: [\ell_0[2]: \text{loop forever do}] \\
&\quad [\ell_1[2]: \text{noncritical}] \\
&\quad [\ell_2[2]: \text{request } y[2]] \\
&\quad [\ell_3[2]: \text{critical}] \\
&\quad [\ell_4[2]: \text{release } y[1]] 
\end{align*}
\]

Parameterized Programs: Verification

Objective: prove \{ \phi \} \tau[i] \{ \phi \} in a uniform way for all \( i \in [1..M] \)

Example: Program mpx-sem (Fig 2.3) \( M \geq 2 \)

Prove mutual exclusion:

\[ \Box(N_3 \leq 1) \]

The assertion \( \phi \) is not inductive, therefore we prove the invariance of

\[ \phi_1: \quad \forall j . y[j] \geq 0 \]

\[ \phi_2: \quad (N_{3,4} + \sum_{j=1}^{M} y[j]) = 1 \]

where \( N_{3,4} = \text{Number of processes currently residing at } \ell_3 \text{ or at } \ell_4 \)

Example: Program mpx-sem (Con’t)

B2:
The only transitions that interfere with \( \phi_1 \) are \( \tau_{\ell_2}[i] \) and \( \tau_{\ell_4}[i] \).

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

\[ \rho_{\ell_4}[i]: \text{move}(\ell_4[i], \ell_0[i]) \land y' = \text{update}(y, i \oplus M 1, y[i \oplus M 1] + 1) \]

\( \rho_{\ell_2}[i] \) implies

\[ y[i] > 0 \land y'[i] = y[i] - 1 \land \forall j . j \neq i . y'[j] = y[j] \]

\( \rho_{\ell_4}[i] \) implies

\[ y'[i \oplus M 1] = y[i \oplus M 1] + 1 \land \forall j . j \neq i \oplus M 1 . y'[j] = y[j] \]

We therefore have

\[ \forall j . y[j] \geq 0 \land \{ \rho_{\ell_2}[i] \} \rightarrow \forall j . y'[j] \geq 0 \]

9-15

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

9-14
Proof of $\square \left( N_{3,4} + \left( \sum_{j=1}^{M} y[j] \right) = 1 \right)$

B1: 

\[
\begin{aligned}
\pi &= \{ \ell_0[1], \ldots, \ell_0[M] \} \land \\
y[1] &= 1 \land (\forall j. 2 \leq j \leq M . y[j] = 0)
\end{aligned}
\]

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

B2: Verification conditions:

$\rho \ell_2[i]$ implies:

\[
N'_{3,4} = N_{3,4} + 1
\]

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

Parameterized Programs: Examples

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

where

\[
\begin{align*}
\text{request} (y, c) &= \langle \text{await } y \geq c; \ y := y - c \rangle \\
\text{release} (y, c) &= \langle y := y + c \rangle
\end{align*}
\]

\[
\begin{aligned}
\forall i,j \in [1..M], i \neq j . at_{-\ell_6[i]} &\rightarrow \neg (at_{-\ell_6[j]} \lor at_{-\ell_3[j]})
\end{aligned}
\]

- $\varphi_1$ and $\varphi_2$ are inductive

$\varphi_1$: $y \geq 0$

$\varphi_2$: $N_{3,4} + M \cdot N_{6,7} + y = M$

Therefore

\[
N_{6,7} > 0 \rightarrow (N_{6,7} = 1 \land N_{3,4} = 0)
\]

Thus, $\varphi_1, \varphi_2$

Program read-write(Fig. 2.11)

\[
\begin{aligned}
\ell_0: \text{loop forever do} \\
\ell_1: \text{noncritical} \\
\text{in } M: \text{integer where } M \geq 1 \\
\text{local } y: \text{integer where } y = M
\end{aligned}
\]

\[
\begin{aligned}
R \::& \ell_2: \text{request } (y,1) \\
\ell_3: \text{read} \\
\ell_4: \text{release } (y,1)
\end{aligned}
\]

or

\[
\begin{aligned}
W \::& \ell_6: \text{write} \\
\ell_7: \text{release } (y,M)
\end{aligned}
\]
**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

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


\[
\begin{array}{cccc}
\ell_0: & \text{loop forever do} & \\
\ell_1: & \text{noncritical} & \\
\ell_2: & \text{request } c[j] & \\
\ell_3: & \text{request } c[j \oplus_M 1] & \\
\ell_4: & \text{critical} & \\
\ell_5: & \text{release } c[j] & \\
\ell_6: & \text{release } c[j \oplus_M 1] & \\
\end{array}
\]

9-24
Specification: Chopstick Exclusion

\[ \forall j \in [1..M] . \neg (\text{at}_{-\ell 4}[j] \land \text{at}_{-\ell 4}[j + M - 1]) \]

Mutual exclusion between every two adjacent philosophers

Proof:

- \( \varphi_0 \) and \( \varphi_1 \) are inductive
  - \( \varphi_0: \forall j \in [1..M]. \ c[j] \geq 0 \)
  - \( \varphi_1: \forall j \in [1..M]. \ \text{at}_{-\ell 4..6}[j] + \text{at}_{-\ell 3..5}[j + M - 1] + c[j + M - 1] = 1 \)

- Then,
  \[ \text{at}_{-\ell 4}[j] + \text{at}_{-\ell 4}[j + M - 1] \leq \text{at}_{-\ell 4..6}[j] + \text{at}_{-\ell 3..5}[j + M - 1] \]
  \[ = 1 - c[j + M - 1] \leq 1 \]

\( \varphi_1 \) \( \varphi_0 \)

Chopstick Exclusion OK

Problem: possible deadlock ("starvation")

- \( P[1] \) \( \ell_2: \text{request } c[1]; \ \ell_3: \text{request } c[2] \)
- \( P[M] \) \( \ell_2: \text{request } c[M]; \ \ell_3: \text{request } c[1] \)

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” (initially \( r = M - 1 \))
  No more than \( M - 1 \) philosophers are admitted to the dining hall at the same time.

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

\[ p \Rightarrow q_m W (q_{m-1} \cdots (q_1 W q_0) \cdots) \]

also written

\[ p \Rightarrow q_m W q_{m-1} \cdots q_1 W q_0 \]

for assertions \( p, q_0, q_1, \ldots, q_m \).

Models that satisfy these formulas

\[
\begin{array}{cccc}
q_m & q_{m-1} & \cdots & q_1 \\
\text{interval} & \text{interval} & \cdots & \text{interval} \\
\{p\} & \{\} & \cdots & \{q_0\} \\
\uparrow & \uparrow & \cdots & \uparrow \\
p\text{-position} & \text{ } & \cdots & q_0\text{-position}
\end{array}
\]
Intermediate Assertion \( \varphi \)

W1. \( p \rightarrow \varphi \lor r \)  
\( \text{"} \varphi \text{ weakens } p \land \neg r \text{"} \)

W2. \( \varphi \rightarrow q \)  
\( \text{"} \varphi \text{ strengthens } q \" \)

Example: Program \( \text{mux-pet1} \) (Fig. 3.4)

We proved mutual exclusion
\( \psi_1: \square \neg (at_{\ell_4} \land at_{m_4}) \)

Using invariants

\( \chi_0: \ s = 1 \lor s = 2 \)

\( \chi_1: \ y_1 \leftrightarrow at_{\ell_{3..5}} \)

\( \chi_2: \ y_2 \leftrightarrow at_{m_{3..5}} \)

\( \chi_3: \ at_{\ell_3} \land at_{m_4} \rightarrow y_2 \land s = 1 \)

\( \chi_4: \ at_{\ell_4} \land at_{m_3} \rightarrow y_1 \land s = 2 \)
Example: Program mux-pet1 (Fig. 3.4) (Peterson’s Algorithm for mutual exclusion)

local \( y_1, y_2 \): boolean where \( y_1 = F, y_2 = F \)
\( s \): integer where \( s = 1 \)

\( \ell_0 \): loop forever do
\[
\begin{align*}
\ell_1 & : \text{noncritical} \\
\ell_2 & : (y_1, s) := (T, 1) \\
\ell_3 & : \text{await } (\neg y_2) \lor (s \neq 1) \\
\ell_4 & : \text{critical} \\
\ell_5 & : y_1 := F
\end{align*}
\]

\( P_1 \) ::

\( m_0 \): loop forever do
\[
\begin{align*}
m_1 & : \text{noncritical} \\
m_2 & : (y_2, s) := (T, 2) \\
m_3 & : \text{await } (\neg y_1) \lor (s = 2) \\
m_4 & : \text{critical} \\
m_5 & : y_2 := F
\end{align*}
\]

We want to prove simple precedence

\[
\psi_2: \at_{\ell_3} \land \at_{m_{0..2}} \Rightarrow \neg \at_{m_{0..2}} \at_{\ell_3}
\]

We try to find an assertion \( \varphi \) such that
\( W1 – W3 \) of rule \textsc{wait} hold

Let
\[
\varphi : \at_{\ell_3} \land (\at_{m_{0..2}} \lor (\at_{m_3} \land s = 2))
\]

Proving precedence properties:
Systematic derivation of intermediate assertions

\[
\begin{align*}
\psi_2: \at_{\ell_3} \land \at_{m_{0..2}} & \Rightarrow \neg \at_{m_{0..2}} \at_{\ell_3} \\
\Rightarrow \at_{\ell_3} \land (\at_{m_{0..2}} \lor (\at_{m_3} \land s = 2)) & \Rightarrow \neg \at_{m_{0..2}}
\end{align*}
\]

Recall:

\textbf{Rule} \textsc{wait} (general waiting-for)

For assertions \( p, q, r, \varphi \)
\[
\begin{align*}
W1. & \quad p \rightarrow \varphi \lor r \\
W2. & \quad \varphi \rightarrow q \\
W3. & \quad \{\varphi\}T\{\varphi \lor r\} \\
& \quad p \Rightarrow q \ W r
\end{align*}
\]

How to find \( \varphi \)?
Escape Transition

Transition that leads to r-state.

Forward propagation

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

Based on postcondition:

$$
\Psi(V) = post(\tau, \varphi) : \exists V^0 . \varphi(V^0) \land \rho_{\tau}(V^0, V)
$$

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

Example: Postcondition

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

$$
\rho_{\tau} : x' = x + y \land y' = x,
$$

$\Phi : x = y$

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

$$
\exists x^0, y^0 : y^0 = x^0 \land x = x^0 + y^0 \land y = x^0, \tag{\varphi(V^0)}
\rho_{\tau}(V^0, V)
$$

which can be simplified to

$$
\Psi : x = y + y.
$$

Forward Propagation: Algorithm

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

1. $\Phi_0 = p \land \neg r$

2. Repeat

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

for any non-escape transition $\tau$

Until

$$
post(\tau, \Phi_t) \rightarrow \Phi_t \quad \text{[may use invariants]}
$$

for all non-escape transitions $\tau$

If this terminates (it may not), $\Phi_t$ is a good assertion to be used in rule WAIT.

Satisfies W1, W3, but check W2.
Backward Propagation

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

Based on precondition:
\[ \text{pre}(\tau, \varphi) : \forall V'. \rho_\tau(V, V') \rightarrow \varphi(V') \]

$\text{pre}(\tau, \varphi)$ characterizes all states all of whose $\tau$-successors satisfy $\varphi$.

Example: Precondition

For Peterson’s Algorithm, consider
\[ \Gamma_0 : \neg \text{at}_m \]

and calculate $\text{pre}(m_3, \Gamma_0)$:
\[ \forall V' : \text{at}_m \wedge (\neg y_1 \vee s \neq 2) \wedge \text{at}_m' \wedge \cdots \rightarrow \neg \text{at}_m'. \]

$P$-equivalent to
\[ \text{at}_m \rightarrow (y_1 \wedge s = 2). \]

Backward Propagation: Algorithm

1. $\Gamma_0 = q$
2. Repeat
\[ \Gamma_{k+1} = \Gamma_k \wedge \text{pre}(\tau, \Gamma_k) \]
for any non-escape transition $\tau$

Until
\[ \Gamma_f \rightarrow \text{pre}(\tau, \Gamma_f) \]
[may use invariants]
for all non-escape transitions $\tau$

If this terminates (it may not), $\Gamma_f$ is a good assertion to be used in rule \textsc{Wait}.

Satisfies W2, W3, but check W1.

Backward vs. Forward

If $p \Rightarrow q \ W r$ 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)

\[ y_1, y_2 : \text{boolean where } y_1 = F, y_2 = F \]
\[ s : \text{integer where } s = 1 \]

\[ \ell_0 : \text{loop forever do} \]
\[ \begin{align*}
\ell_1 & : \text{noncritical} \\
\ell_2 & : (y_1, s) := (T, 1) \\
\ell_3 & : \text{await } (\neg y_2) \lor (s \neq 1) \\
\ell_4 & : \text{critical} \\
\ell_5 & : y_1 := F
\end{align*} \]

\[ P_1 : \]

\[ m_0 : \text{loop forever do} \]
\[ \begin{align*}
m_1 & : \text{noncritical} \\
m_2 & : (y_2, s) := (T, 2) \\
m_3 & : \text{await } (\neg y_1) \lor (s \neq 2) \\
m_4 & : \text{critical} \\
m_5 & : y_2 := F
\end{align*} \]

Example: Forward Propagation (cont.)
\[ i.e., \]
\[ at_{\ell 3} \land (at_{m0 \cdot 2} \lor (at_{m3} \land s = 2)) \]

\( \phi_1 \) is preserved under all transitions except the escape transition \( \ell_3 \), so the process converges.

Example: Forward Propagation

\[ at_{\ell 3} \land at_{m0 \cdot 2} \Rightarrow \neg at_{m4} \quad \text{W} \quad at_{\ell 4} \]
Start with
\[ \phi_0 : at_{\ell 3} \land at_{m0 \cdot 2} \]
and calculate \( \text{post}(m_2, \phi_0) \):
\[ \exists (s^0, y_1^0, y_2^0, s^0) : (at_{\ell 3}^0 \land (at_{m0 \cdot 2}^0) \land \]
\[ (at_{m2}^0) \land (at_{m3}^0 \land ((at_{\ell 3}^0) \leftrightarrow at_{\ell 3}) \land s = 2 \land \ldots \]
\[ \rho_{m2}(V^0, V) \]

\( P \)-equivalent to
\[ \psi_1 : at_{\ell 3} \land at_{m3} \land s = 2, \]
using the invariant \( \varphi_1 : y_1 \leftrightarrow at_{\ell 3 \cdot 5} \).

Thus,
\[ \phi_1 : at_{\ell 3} \land at_{m0 \cdot 2} \lor (at_{\ell 3} \land at_{m3} \land s = 2) \]

Example: Backward Propagation

Start with
\[ \Gamma_0 : \neg at_{m4} \]
We calculated \( \text{pre}(m_3, \Gamma_0) \) above, which is \( P \)-equivalent to
\[ \Delta_1 : at_{m3} \rightarrow (y_1 \land s = 2) \]
Thus,
\[ \Gamma_1 : \neg at_{m4} \land (at_{m3} \rightarrow (y_1 \land s = 2)) \]
Consider transition \( \tau_{m2} \), and calculate \( \text{pre}(m_2, \Gamma_1) \):
\[ \forall V' : at_{m2} \land (at_{m3} \land y_1' = y_1 \land s' = 2 \land \ldots \]
\[ \rightarrow \neg at_{m4} \land (at_{m3} \rightarrow (y_1' \land s' = 2)) \]

\( P \)-equivalent to
\[ \Delta_2 : at_{m2} \rightarrow y_1. \]
Thus,

\[ \Gamma_2 : \neg \text{at } m_4 \land (\text{at } m_3 \rightarrow s = 2) \land (\text{at } m_{2,3} \rightarrow y_1). \]

Considering transitions \( \tau_{m_1}, \tau_{m_0}, \) and \( \tau_{m_5} \) leads to the following sequence:

\[ \Gamma_3 : \neg \text{at } m_4 \land (\text{at } m_3 \rightarrow s = 2) \land (\text{at } m_{1,3} \rightarrow y_1) \]
\[ \Gamma_4 : \neg \text{at } m_4 \land (\text{at } m_3 \rightarrow s = 2) \land (\text{at } m_{0,3} \rightarrow y_1) \]
\[ \Gamma_5 : \neg \text{at } m_4 \land (\text{at } m_3 \rightarrow s = 2) \land (\text{at } m_{0,3,5} \rightarrow y_1) \]

By the control invariant \( \text{at } m_{0,5} \), \( \Gamma_5 \) can be simplified to

\[ \Gamma_5 : \neg \text{at } m_4 \land (\text{at } m_3 \rightarrow s = 2) \land y_1. \]

Calculating \( \text{pre}(\ell_5, \Gamma_5) \),

\[ \forall V' : \text{at } \ell_5 \land y_1' = \ell \land \cdots \to \neg \text{at } m_4' \land (\text{at } m_{3} \to s' = 2) \land y_1', \]

which gives

\[ \Delta_6 : \text{at } \ell_5 \to f. \]

Propagating \( \Gamma_5 \land \Delta_6 \) via \( \tau_{\ell_4} \) gives

\[ \Delta_7 : \text{at } \ell_4 \to f. \]

Hence,

\[ \Gamma_7 : \neg \text{at } m_4 \land (\text{at } m_3 \rightarrow s = 2) \land \text{at } \ell_3 \]

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