State Sequences

- A **state** is a type-consistent interpretation of the system variables $V$

- A **state sequence** is an infinite sequence of states, represented as a mapping from time ($\mathbb{N}$) to states:

  \[
  \text{STATE_SEQ: \ TYPE} = [\text{TIME} \rightarrow \text{STATE}]
  \]

(Recall: $\sigma : s_0, s_1, s_2, \ldots$)

- **Assertions** are properties defined on individual states, without reference to their position in the state sequence.

  \[
  \text{ASSERTION: \ TYPE} = [\text{STATE} \rightarrow \text{bool}]
  \]

Disjunction, conjunction, negation and implication over assertions are defined in the natural manner.
Temporal Operators

Let $S$ be a state sequence.

We denote the notion of a temporal property $p$ holding at position $j \geq 0$ of $S$ by $p(S, j)$.

If $p$ is an assertion then $p(S, j) = p(S(j))$

For the LTL operators:

- Henceforth, $\square$, $G$ \quad $G(p)(S, j) \iff p(S, k)$ for all $k \geq j$
- Eventually, $\Diamond$, $F$ \quad $F(p)(S, j) \iff p(S, k)$ for some $k \geq j$
- Next, $\bigcirc$, $X$ \quad $X(p)(S, j) \iff p(S, j + 1)$
Example

\[ V = \{ c : \{ \text{RED}, \text{BLUE} \} \} \]

Two distinct states: \( \langle c : \text{RED} \rangle, \langle c : \text{BLUE} \rangle \)

\( \text{is\_blue} : \text{ASSERTION} = (\lambda (s : \text{STATE}) : s'c = \text{BLUE}) \)

Consider state sequence \( SS : [\text{TIME} \mapsto \text{STATE}] \) defined by the first 2 columns in the table:

<table>
<thead>
<tr>
<th>time</th>
<th>state</th>
<th>is_blue</th>
<th>X(is_blue)</th>
<th>F(is_blue)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>\langle c : \text{BLUE} \rangle</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>\langle c : \text{RED}  \rangle</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>\langle c : \text{RED}  \rangle</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>\langle c : \text{BLUE} \rangle</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>\langle c : \text{BLUE} \rangle</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>\langle c : \text{BLUE} \rangle</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

\[ \ldots \]

- Assertion \( \text{is\_blue} \) depends only on the state. \( \text{is\_blue}(SS, i) = \text{is\_blue}(SS(i)) \)

- Temporal properties depend on the whole state sequence. \( SS(0) = SS(3) \) but \( X(\text{is\_blue})(SS, 0) \) is 1, \( X(\text{is\_blue})(SS, 3) \) is 0

- \( X(\text{is\_blue})(SS(0)) \) is incorrectly typed as \( SS(0) \) is the state \( \langle c : \text{BLUE} \rangle \) (as is \( SS(3)! \))
tempoperators[STATE: NONEMPTY_TYPE]: THEORY
BEGIN
  TIME: TYPE = nat
  STATE_SEQ: TYPE = [TIME -> STATE]
  TP: TYPE = [STATE_SEQ, TIME -> boolean]
  s: VAR STATE
  t, u, j, k : VAR TIME
  a, b: VAR TP
  seq: VAR STATE_SEQ
  OR: [TP, TP -> TP] =
    (LAMBDA a, b: (LAMBDA seq, t: a(seq, t) OR b(seq, t)))
  AND: [TP, TP -> TP] =
    (LAMBDA a, b: (LAMBDA seq, t: a(seq, t) AND b(seq, t)))
    . . .
  G: [TP -> TP] = (LAMBDA a: (LAMBDA seq, j:
    FORALL t: t >= j IMPLIES a(seq, t)))
  F: [TP -> TP] = (LAMBDA a: (LAMBDA seq, j:
    EXISTS t: t >= j AND a(seq, t)))
  X: [TP -> TP] = (LAMBDA a: (LAMBDA seq, t: a(seq, t + 1)))
  U: [TP, TP -> TP] =
    (LAMBDA a, b: (LAMBDA seq, j:
      EXISTS t:
        t >= j AND b(seq, t) AND
        (FORALL k: j <= k AND k < t IMPLIES a(seq, k))))
END tempoperators
FDS - Fair Discrete System

Recall, \( FDS \ D = \langle V, \Theta, \rho, J, C \rangle \) consists of:

- \( V \) : a set of typed state variables. A \( V \)-state is an interpretation of \( V \). \( \Sigma_V \) is the set of all \( V \)-states.

- \( \Theta \) : The initial condition. An assertion characterizing the initial states.

- \( \rho \) : The transition relation. A predicate \( \rho(V, V') \) referring to the both unprimed (current) and primed (next) versions of state variables.

- \( J \) : The set of justice requirements. Each computation must have infinitely many \( J_i \)-states, for every \( J_i \in J \).

- \( C \) : The compassion requirements, each of the form \( \langle p, q \rangle \). Infinitely many \( p \)-states imply infinitely many \( q \)-states.
PFS - Parameterized Fair System

PFS: TYPE =
  [# initial: ASSERTION,
    rho: BI_ASSERTION,
    justice: JUSTICE_TYPE,
    compassion: COMPASSION_TYPE #]

where

BI_ASSERTION: TYPE = [STATE, STATE → boolean]

JUSTICE_TYPE: TYPE = [TRANSITION_DOMAIN → ASSERTION]

COMPASSION_PAIR: TYPE =
  [# p: ASSERTION, q: ASSERTION #]

COMPASSION_TYPE: TYPE =
  [TRANSITION_DOMAIN → COMPASSION_PAIR]

Note:

- **STATE** and **TRANSITION_DOMAIN** parameters are given in defining the PFS
- There is no state-variables \((V)\) component
Runs and Computations

A STATE_SEQ \texttt{seq} of \texttt{pfs} is an \textit{initialized run} if it satisfies

- \textbf{Initaility}: \( \texttt{seq}(0) \) is initial i.e. \( \texttt{pfs}'\text{initial}(\texttt{seq}(0)) \)

- \textbf{Consecution}: For every \( t = 0, 1, 2 \), state \( \texttt{seq}(j + 1) \) is a successor of \( \texttt{seq}(t) \).
  i.e. \( \texttt{pfs}'\text{rho}(\texttt{seq}(t), \texttt{seq}(t + 1)) \)

A \textit{computation} is an \textit{initialized run} which also satisfies the \textbf{fairness} requirements of:

- \textbf{Justice}: For every \( t \in \text{TRANSITION\_DOMAIN} \) there are infinitely many states in \( \texttt{seq} \) at which \( \texttt{pfs}'\text{justice}(t) \) holds.

- \textbf{Compassion}: For every \( t \in \text{TRANSITION\_DOMAIN} \), if there are infinititely many \( \texttt{(pfs}'\text{compassion}(t)'p) \)-states in \( \texttt{seq} \) then there are infinititely \( \texttt{(pfs}'\text{compassion}(t)'q) \)-states in \( \texttt{seq} \).
Validity

A temporal property \( p \) is termed

- **valid** if it hold in the first state of every state sequence \( \text{seq} \).
  
  \[
  \text{is\_valid}(p)
  \]

- **P-valid** if it hold in the first state of every computation \( \text{seq} \) of program \( P \).
  
  Assuming that \( \text{PFS} \) defines program \( P \),
  
  \[
  \text{is\_P\_valid}(p, \text{pfs})
  \]

- **P-reachable valid** if it hold in the first state of every initialized run \( \text{seq} \) of program \( P \).
  
  \[
  \text{is\_P\_reachable\_valid}(p, \text{pfs})
  \]
Validity ctd

- **state sequence ⊇ initialized runs ⊇ computations**, and so

  validity $\rightarrow P$-reachable validity $\rightarrow P$-validity

- Generally, interested in $P$-validity

- Rules like $\text{BINV}$ actually prove the stronger $P$-reachable validity property

- Can always convert $P$-reachable validity to $P$-validity.
  
  Sometimes having the stronger property is useful.
**Example:** MUX-SEM

<table>
<thead>
<tr>
<th>in</th>
<th>( N ) : integer where ( N &gt; 1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>local</td>
<td>( y ) : {0, 1} where ( y = 1 )</td>
</tr>
</tbody>
</table>

\[
\begin{array}{c}
\frac{N}{p=1} P[p] ::
\end{array}
\]

\[
\begin{array}{c}
\ell_0 : \text{loop forever do} \\
\ell_1 : \text{noncritical} \\
\ell_2 : \text{request } y \\
\ell_3 : \text{critical} \\
\ell_4 : \text{release } y \\
\end{array}
\]

Figure 2: Parameterized MUX-SEM
A PFS for MUX-SEM

muxsem[N: posnat]: THEORY
BEGIN

IMPORTING more_nat_types

LOCATION: TYPE = upto[4]

PROC_ID: TYPE = upto_nz[N]

TRANS_DOMAIN: TYPE =
    [# loc: LOCATION, pid: PROC_ID #]

STATE: TYPE =
    [# y: upto[1], loc: [PROC_ID \rightarrow LOCATION] #]

IMPORTING PFS[STATE, TRANS_DOMAIN]

p: VAR PROC_ID
\textbf{rho: BI-ASSERTION =}

\((\lambda \text{ current, next: STATE}):
\text{\textbf{next} = current} \lor
\exists p:\)
\begin{align*}
\text{loc(current)(p)} &= 0 \land \\
\text{y(next)} &= \text{y(current)} \land \\
\text{loc(next) = loc(current) WITH [(p) := 1]} \lor \\
\text{loc(current)(p)} &= 1 \land \\
\text{y(next)} &= \text{y(current)} \land \\
\text{loc(next) = loc(current) WITH [(p) := 2]} \lor \\
\text{loc(current)(p)} &= 2 \land \\
\text{y(current)} &= 1 \land \\
\text{y(next)} &= 0 \land \\
\text{loc(next) = loc(current) WITH [(p) := 3]} \lor \\
\text{loc(current)(p)} &= 3 \land \\
\text{y(next)} &= \text{y(current)} \land \\
\text{loc(next) = loc(current) WITH [(p) := 4]} \lor \\
\text{loc(current)(p)} &= 4 \land \\
\text{y(next)} &= 1 \land \\
\text{loc(next) = loc(current) WITH [(p) := 0]})
\end{align*}
st: VAR STATE
t: VAR TRANS_DOMAIN

justice: JUSTICE_TYPE =
(\lambda t: (\lambda st:
  IF loc(t) = 0 \lor loc(t) = 3 \lor loc(t) = 4
  THEN loc(st)(pid(t)) \neq loc(t)
  ELSE TRUE
ENDIF))

compassion: COMPASSION_TYPE =
(\lambda t:
  IF loc(t) = 2
  THEN
    (# p := (\lambda st: loc(st)(pid(t))=2 \land y(st)=1),
    q := (\lambda st: loc(st)(pid(t)) = 3) #)
  ELSE (# p := (\lambda st: TRUE), q := (\lambda st: TRUE) #)
ENDIF)

pfs: PFS =
(# initial := \{st | y(st)=1 \land (\forall p: loc(st)(p)=0)\},
 rho := rho,
 justice := justice,
 compassion := compassion #)

END muxSem
Transition domains

- Very often, as was the case in MUX-SEM, the transition domain is comprised of a location and processor identifier field.

- TRANS DOMAIN theory, which defines such a transition domain.

- Importing TRANS DOMAIN[progSize, N] creates and imports the following definitions:

  LOCATION: TYPE = upto[progSize - 1]

  PROC_ID: TYPE = upto_nz[N]

  TRANS DOMAIN: TYPE =
  [# loc: LOCATION, pid: PROC_ID #]

- In MUX-SEM, we could have defined

  IMPORTING TRANS DOMAIN[5, N]
**Example:** BAKERY

\[
\begin{align*}
\text{in} & \quad N : \text{integer where } N > 1 \\
\text{local} & \quad y : \text{array } [1..N] \text{ of natural where } y = 0 \\
\text{loop forever do} & \quad \begin{cases} \\
\ell_0 : & \text{NonCritical} \\
\ell_1 : & y[p] := \text{choose } m \text{ such that } \\
& \forall q : (m > y[q]) \\
\ell_2 : & \text{await } \forall q : (y[q] = 0 \lor y[p] < y[q]) \\
\ell_3 : & \text{Critical} \\
\ell_4 : & y[p] := 0 \\
\end{cases}
\end{align*}
\]

Figure 3: Parameterized mutual exclusion algorithm BAKERY
PFS for BAKERY

bakery_definition[\(N: \text{posnat}\)]: \text{THEORY}
BEGIN

IMPORTING \text{TRANS\_DOMAIN}[5, N]

\text{STATE: TYPE =}
[# y: [PROC\_ID \to \text{nat}],
loc: [PROC\_ID \to \text{LOCATION}] #]

IMPORTING PFS[\text{STATE, TRANS\_DOMAIN}]

p, q: \text{VAR PROC\_ID}
\[ \text{the: BI\_ASSERTION} = \\
\quad (\lambda (\text{current, next: STATE}): \\
\qquad \text{next} = \text{current} \lor \\
\qquad (\exists \; p: \\
\qquad \quad \text{loc}(\text{current})(p) = 0 \land \\
\qquad \quad y(\text{next}) = y(\text{current}) \land \\
\qquad \quad \text{loc}(\text{next}) = \text{loc}(\text{current}) \text{ WITH } [(p) := 1] \\
\qquad \lor \\
\qquad \quad \text{loc}(\text{current})(p) = 1 \land \\
\qquad \quad (\exists \; (m: \text{nat}): (\forall \; q: y(\text{current})(q) < m) \land \\
\qquad \quad y(\text{next}) = y(\text{current}) \text{ WITH } [(p) := m]) \\
\qquad \land \text{loc}(\text{next}) = \text{loc}(\text{current}) \text{ WITH } [(p) := 2] \\
\qquad \lor \\
\qquad \quad \text{loc}(\text{current})(p) = 2 \land \\
\qquad \quad (\forall \; q: q \neq p \rightarrow y(\text{current})(q) = 0 \lor \\
\qquad \quad \quad y(\text{current})(p) \leq y(\text{current})(q)) \\
\qquad \land y(\text{next}) = y(\text{current}) \\
\qquad \land \text{loc}(\text{next}) = \text{loc}(\text{current}) \text{ WITH } [(p) := 3] \\
\qquad \lor \\
\qquad \quad \text{loc}(\text{current})(p) = 3 \\
\qquad \land y(\text{next}) = y(\text{current}) \\
\qquad \land \text{loc}(\text{next}) = \text{loc}(\text{current}) \text{ WITH } [(p) := 4] \\
\qquad \lor \\
\qquad \quad \text{loc}(\text{current})(p) = 4 \\
\qquad \land y(\text{next}) = y(\text{current}) \text{ WITH } [(p) := 0] \\
\qquad \land \text{loc}(\text{next}) = \text{loc}(\text{current}) \text{ WITH } [(p) := 0])] \\
\]

\[ \text{st: VAR STATE} \]
\[ t: \text{VAR TRANS\_DOMAIN} \]

\[ \text{justice: } [\text{TRANS\_DOMAIN } \rightarrow \text{ASSERTION}] = \]
\[ (\lambda t: (\lambda st:\]
\[ \text{IF } \text{loc}(t) = 1\]
\[ \text{THEN } \text{loc}(st)(\text{pid}(t)) \neq 1\]
\[ \forall \neg (\exists (m: \text{nat}): \forall p: y(st)(p) < m)\]
\[ \text{ELSIF } \text{loc}(t) = 2\]
\[ \text{THEN } \text{loc}(st)(\text{pid}(t)) \neq 2 \land\]
\[ \neg (\forall q: q \neq \text{pid}(t) \rightarrow\]
\[ y(st)(q) = 0 \lor y(st)(\text{pid}(t)) \leq y(st)(q))\]
\[ \text{ELSIF } \text{loc}(t) = 3 \lor \text{loc}(t) = 4\]
\[ \text{THEN } \text{loc}(st)(\text{pid}(t)) \neq \text{loc}(t)\]
\[ \text{ELSE TRUE}\]
\[ \text{ENDIF})\]

\[ \text{pfs: } \text{PFS} = \]
\[ (# \text{initial} := \{st| \forall p: y(st)(p)=0 \land \text{loc}(st)(p)=0\},\]
\[ \text{rho} := \text{rho},\]
\[ \text{justice} := \text{justice},\]
\[ \text{compassion} := \text{empty_compassion} #)\]

\text{END bakery_definition}
Proving properties of BAKERY

\[ y_{Zero}: \text{ASSERTION} = \]
\[ \lambda \text{st}: \]
\[ \text{LET } y = y(\text{st}), \text{loc} = \text{loc}(\text{st}) \text{ IN} \]
\[ \forall (i: \text{PROC_ID}): \]
\[ (y(i) = 0 \text{ IFF } \text{loc}(i) = 0 \lor \text{loc}(i) = 1)) \]

\[ y_{Zero}: \text{LEMMA } \text{is\_P\_reachable\_valid}(G(y_{Zero}), \text{fds}) \]
yZero :

|--------
{1} \text{is\_P\_reachable\_valid}(G(\text{assertion\_to\_TP}[\text{STATE}[N]](yZero)), \text{pfs})

Rule? (BINV "yZero")
The inductive step of the BINV rule

{-1,(rho)}

\[
\text{loc(current!1)(p!1) = 0 AND y(next!1) = y(current!1) AND} \\
\text{loc(next!1) = loc(current!1) WITH [(p!1) := 1]}
\]

OR \text{loc(current!1)(p!1) = 1 AND}

(EXISTS (m: nat): (FORALL (q: PROC_ID):

\[
\text{y(current!1)(q) < m AND y(next!1) = y(current!1) WITH [(p!1) := m]} \\
\text{AND loc(next!1) = loc(current!1) WITH [(p!1) := 2]}
\]

OR \text{loc(current!1)(p!1) = 2 AND}

(FORALL q: q /= p!1 IMPLIES

\[
\text{y(current!1)(q) = 0 OR y(current!1)(p!1) <= y(current!1)(q)} \text{ AND} \\
\text{y(next!1) = y(current!1) AND} \\
\text{loc(next!1) = loc(current!1) WITH [(p!1) := 3]}
\]

OR \text{loc(current!1)(p!1) = 3 AND y(next!1) = y(current!1) AND}

\text{loc(next!1) = loc(current!1) WITH [(p!1) := 4]}

OR \text{loc(current!1)(p!1) = 4 AND}

\[
\text{y(next!1) = y(current!1) WITH [(p!1) := 0] AND} \\
\text{loc(next!1) = loc(current!1) WITH [(p!1) := 0]}
\]

{-2,(yZero invariant)}) FORALL (i: PROC_ID):

\[
\text{(y(current!1)(i) = 0 IFF (loc(current!1)(i) = 0 OR loc(current!1)(i) = 0) IMPLIES} \\
\text{(next!1)(i) = 0) AND loc(next!1) = loc(current!1) WITH [(p!1) := 0]}
\]

{-1,(rtp)} \text{(y(next!1)(i!1) = 0 IFF (loc(next!1)(i!1) = 0 OR loc(next!1)(i!1) = 0)}

Rule? (split-rho)
this yields 5 subgoals:

yZero.1 :
{-1,(rho)}
   loc(current!1)(p!1) = 0
{-2,(rho)}
   y(next!1) = y(current!1)
{-3,(rho)}
   loc(next!1) = loc(current!1) WITH [(p!1) := 1]
{-4,(yZero invariant)}
   FORALL (i: PROC_ID):
      IF y(current!1)(i) = 0
         THEN (loc(current!1)(i) = 0 OR loc(current!1)(i) = 1)
         ELSE NOT (loc(current!1)(i) = 0 OR loc(current!1)(i) = 1)
      ENDIF
|-------
{1,(rtp)}
   IF y(current!1)(i!1) = 0
      THEN (loc(current!1) WITH [(p!1) := 1](i!1) = 0 OR
          loc(current!1) WITH [(p!1) := 1](i!1) = 1)
      ELSE NOT (loc(current!1) WITH [(p!1) := 1](i!1) = 0 OR
          loc(current!1) WITH [(p!1) := 1](i!1) = 1)
   ENDIF

Rule? (inst - "i!1")
[-1, (rho)]
    \text{loc}(\text{current!1})(p!1) = 0

[-2, (rho)]
    \text{y}(\text{next!1}) = \text{y}(\text{current!1})

[-3, (rho)]
    \text{loc}(\text{next!1}) = \text{loc}(\text{current!1}) \text{ WITH } [(p!1) := 1]

{-4, (yZero invariant)}
    \text{IF } \text{y}(\text{current!1})(i!1) = 0
    \text{ THEN } (\text{loc}(\text{current!1})(i!1) = 0 \text{ OR } \text{loc}(\text{current!1})(i!1) = 1)
    \text{ ELSE NOT } (\text{loc}(\text{current!1})(i!1) = 0 \text{ OR } \text{loc}(\text{current!1})(i!1) = 1)
    \text{ ENDIF}

|--

[1, (rtp)]
    \text{IF } \text{y}(\text{current!1})(i!1) = 0
    \text{ THEN } (\text{loc}(\text{current!1}) \text{ WITH } [(p!1) := 1](i!1) = 0 \text{ OR } \text{loc}(\text{current!1}) \text{ WITH } [(p!1) := 1](i!1) = 1)
    \text{ ELSE NOT } (\text{loc}(\text{current!1}) \text{ WITH } [(p!1) := 1](i!1) = 0 \text{ OR } \text{loc}(\text{current!1}) \text{ WITH } [(p!1) := 1](i!1) = 1)
    \text{ ENDIF}

Rule? (split-all)
Split-all if-then-else consequents,

This completes the proof of yZero.1.
yZero.2
{-1, (rho)}
   \text{loc(\text{current}!1)(p!1) = 1}
{-2, (rho)}
   \text{FORALL (q: \text{PROC}ID[5, N]): y(\text{current}!1)(q) < m!1}
{-3, (rho)}
   \text{y(\text{next}!1) = y(\text{current}!1) WITH [(p!1) := m!1]}
{-4, (rho)}
   \text{loc(\text{next}!1) = loc(\text{current}!1) WITH [(p!1) := 2]}
{-5, (yZero invariant)}
   \text{FORALL (i: \text{PROC}ID):}
      \text{IF y(\text{current}!1)(i) = 0}
         \text{THEN (loc(\text{current}!1)(i) = 0 OR loc(\text{current}!1)(i) = 1)}
         \text{ELSE NOT (loc(\text{current}!1)(i) = 0 OR loc(\text{current}!1)(i) = 1) ENDIF}
   |-------
{1, (rtp)}
   \text{IF y(\text{current}!1) WITH [(p!1) := m!1](i!1) = 0}
      \text{THEN (loc(\text{current}!1) WITH [(p!1) := 2](i!1) = 0 OR}
         \text{loc(\text{current}!1) WITH [(p!1) := 2](i!1) = 1) ELSE NOT (loc(\text{current}!1) WITH [(p!1) := 2](i!1) = 0 OR}
            \text{loc(\text{current}!1) WITH [(p!1) := 2](i!1) = 1) ENDIF}

Rule? (split-all-inst ("i!1"))

This completes the proof of yZero.2.
The inductive step of the BINV rule

\{-1, (\rho)\}
\begin{align*}
\text{loc(current!1)(p!1)} &= 0 \text{ AND } y(\text{next!1}) = y(\text{current!1}) \text{ AND} \\
\text{loc(next!1)} &= \text{loc(current!1)} \text{ WITH } [(p!1) := 1] \\
\text{OR loc(current!1)(p!1)} &= 1 \text{ AND} \\
& \quad (\exists m : \text{nat}: (\forall q : \text{PROC_ID}):
\begin{align*}
&\quad y(\text{current!1})(q) < m \text{ AND } y(\text{next!1}) = y(\text{current!1}) \text{ WITH } [(p!1) := m] \\
&\quad \text{AND loc(next!1)} = \text{loc(current!1)} \text{ WITH } [(p!1) := 2] \\
\end{align*}
\text{OR loc(current!1)(p!1)} &= 2 \text{ AND} \\
& \quad (\forall q : q /= p!1 \implies \\
&\quad y(\text{current!1})(q) = 0 \text{ OR } y(\text{current!1})(p!1) \leq y(\text{current!1})(q)) \text{ AND} \\
&\quad y(\text{next!1}) = y(\text{current!1}) \text{ AND} \\
&\quad \text{loc(next!1)} = \text{loc(current!1)} \text{ WITH } [(p!1) := 3] \\
\text{OR loc(current!1)(p!1)} &= 3 \text{ AND } y(\text{next!1}) = y(\text{current!1}) \text{ AND} \\
& \quad \text{loc(next!1)} = \text{loc(current!1)} \text{ WITH } [(p!1) := 4] \\
\text{OR loc(current!1)(p!1)} &= 4 \text{ AND} \\
& \quad y(\text{next!1}) = y(\text{current!1}) \text{ WITH } [(p!1) := 0] \text{ AND} \\
&\quad \text{loc(next!1)} = \text{loc(current!1)} \text{ WITH } [(p!1) := 0] \\
\end{align*}
\end{align*}

\{-2, (yZero invariant)\} \forall i : \text{PROC_ID}:
\begin{align*}
\forall i : (y(\text{current!1})(i) = 0 \iff (\text{loc(\text{current!1})(i)} = 0 \text{ OR } \text{loc(\text{current!1})(i)} = 1) \\
\end{align*}

\{-1, (\rho)\} \begin{align*}
\forall i : (y(\text{next!1})(i!1) = 0 \iff (\text{loc(\text{next!1})(i!1)} = 0 \text{ OR } \text{loc(\text{next!1})(i!1)} = 1) \\
\end{align*}

\text{Rule? (split-rho-all ("i!1"))}

Q.E.D.
The Right Way of Using PVS

The previous pages showed some of the options and capabilities provided by PVS. However, the user who is mainly interested in verifying programs, should avoid getting into detailed interactive proofs.

To maximize the utility of the TLPVS package, it is recommended to structure all multiply quantified formulas as

\[(\text{FORALL } x: (\text{FORALL } y: \ldots)\]

rather than

\[(\text{FORALL } x, y: \ldots)\]

Then, whenever encountering a verification condition which includes a transition relation, use the standard strategy

\[(\text{split-rho-all} (\text{‘‘x_1’’ ‘‘x_2’’, \ldots}))\]

where \text{x_1, x_2, \ldots} are all the free variables currently appearing in the sequent.
For Example

yZero :

|-------
{1}  is_P_reachable_valid(G(assertion_to_TP[STATE[N]])(yZero)), pfs

Rule? (binv "yZero")

lemma BINV,
this simplifies to:
yZero :
;;;The inductive step of the BINV rule

{-1,(rho)}

  loc(current!1)(p!1) = 0 AND
  y(next!1) = y(current!1) AND
  loc(next!1) = loc(current!1) WITH [(p!1) := 1]
  OR loc(current!1)(p!1) = 1 AND
    (EXISTS (m: nat):
      (FORALL (q: PROC_ID[5, N]): y(current!1)(q) < m) AND
       y(next!1) = y(current!1) WITH [(p!1) := m])
      AND loc(next!1) = loc(current!1) WITH [(p!1) := 2]
  OR loc(current!1)(p!1) = 2 AND
    (FORALL (q: PROC_ID[5, N]):
      q /= p!1 IMPLIES
      y(current!1)(q) = 0 OR y(current!1)(p!1) <= y(current!1)(q))
    AND
\[ y(\text{next!1}) = y(\text{current!1}) \text{ AND} \]
\[ \text{loc(\text{next!1}) = loc(\text{current!1}) WITH [(p!1) := 3]} \]
\[ \text{OR loc(\text{current!1})(p!1) = 3 \text{ AND}} \]
\[ y(\text{next!1}) = y(\text{current!1}) \text{ AND} \]
\[ \text{loc(\text{next!1}) = loc(\text{current!1}) WITH [(p!1) := 4]} \]
\[ \text{OR loc(\text{current!1})(p!1) = 4 \text{ AND}} \]
\[ y(\text{next!1}) = y(\text{current!1}) \text{ WITH [(p!1) := 0]} \text{ AND} \]
\[ \text{loc(\text{next!1}) = loc(\text{current!1}) WITH [(p!1) := 0]} \]
\{-2,(y\text{Zero invariant})\}

\{1,(rtp)\}
\[ y(\text{next!1})(i!1) = 0 \text{ IFF} \]
\[ (\text{loc(\text{current!1})(i) = 0 OR loc(\text{current!1})(i) = 1}) \]
\|--
\{1,(rtp)\}
\[ y(\text{next!1})(i!1) = 0 \text{ IFF} \]
\[ (\text{loc(\text{next!1})(i!1) = 0 OR loc(\text{next!1})(i!1) = 1}) \]

Rule? (split-rho-all ("i!1" "p!1"))

... Tries to complete a proof using split-rho and then split-all-inst. If this fails does simple simplification and returns goal to user, Q.E.D.