

## CS256/Winter 2009 Lecture #8

Zohar Manna

### Finding Inductive Assertions

Top-Down Approach

#### Assertion propagation

we have previously proven  $\Box \chi$

and we want to prove  $\Box \varphi$

but

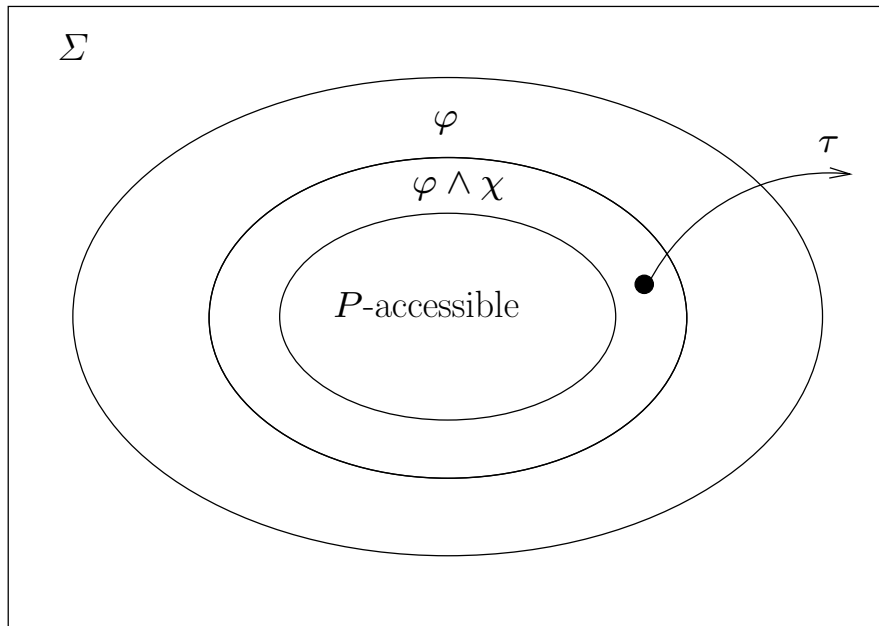
$$\{\chi \wedge \varphi\} \tau \{\varphi\}$$

is not state-valid for some  $\tau \in \mathcal{T}$ .

What is the problem?

(assuming that  $\varphi$  is indeed an invariant)

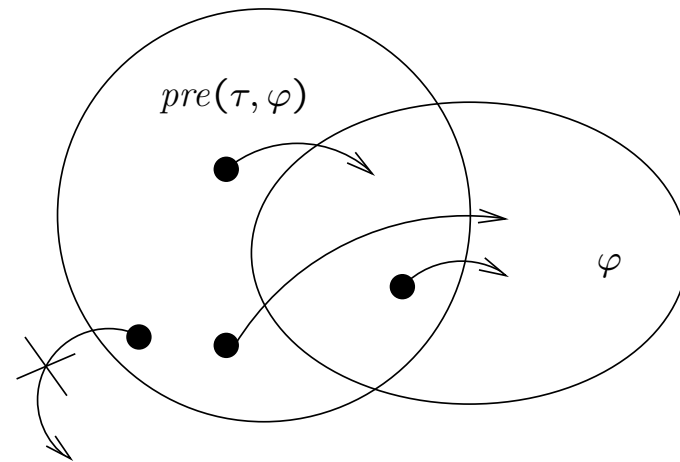
Top-Down Approach (Con'd)



**Solution:** Take the largest set of states that will result in a  $\varphi$ -state when  $\tau$  is taken. How?

**Precondition of  $\varphi$  w.r.t.  $\tau$**

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



a state  $s$  satisfies  $pre(\tau, \varphi)$   
 iff  
 all  $\tau$ -successors of  $s$  satisfy  $\varphi$ .

**Note:**

$s$  trivially satisfies  $pre(\tau, \varphi)$  if it does not have any  $\tau$ -successors (i.e.,  $\tau$  is not enabled in  $s$ ).

### Precondition of $\varphi$ w.r.t. $\tau$ (Con'd)

Example:

$V : \{x\}$  integer

$\rho_\tau : x > 0 \wedge x' = x - 1$

$\varphi : x \geq 2$

$pre(\tau, \varphi) :$

$$\forall x'. \underbrace{x > 0 \wedge x' = x - 1}_{\rho_\tau} \rightarrow \underbrace{x' \geq 2}_{\varphi'}$$

$$x > 0 \rightarrow x - 1 \geq 2$$

$$x \leq 0 \vee x \geq 3$$

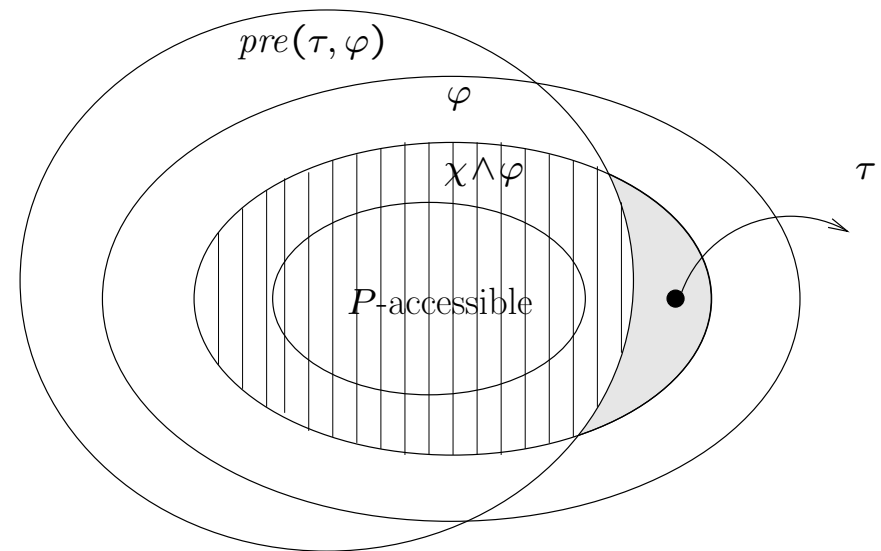
$$\frac{x \leq 0 \overset{j}{\vee} x \geq 3 \quad \tau \quad j+1}{x \geq 2}$$

### Properties of $pre(\tau, \varphi)$

By the definition of  $pre(\tau, \varphi)$ ,

$$\{\chi \wedge \varphi \wedge pre(\tau, \varphi)\} \tau \{\varphi\}$$

is guaranteed to be state-valid.



But we have to justify adding the conjunct  $pre(\tau, \varphi)$  to the antecedent.

This can be done in two ways:

1. Incremental: prove  $\square pre(\tau, \varphi)$
2. Strengthening: prove  $\square(\varphi \wedge pre(\tau, \varphi))$

### Properties of $pre(\tau, \varphi)$ (Con'd)

Claim: If  $\varphi$  is  $P$ -invariant then so is  $pre(\tau, \varphi)$  for every  $\tau \in \mathcal{T}$ .

Proof:

Suppose  $\varphi$  is  $P$ -invariant, but  $pre(\tau, \varphi)$  is not  $P$ -invariant.

Then there exists a  $P$ -accessible state  $s$  such that

$$s \not\models pre(\tau, \varphi).$$

But then, by the definition of  $pre(\tau, \varphi)$ , there exists a  $\tau$ -successor  $s'$  of  $s$  such that  $s' \not\models \varphi$ .

Since  $s$  is  $P$ -accessible,  $s'$  is also  $P$ -accessible, contradicting that  $\varphi$  is a  $P$ -invariant.

### Properties of $pre(\tau, \varphi)$ (Con'd)

Definition: A transition  $\tau$  is said to be self-disabling if for every state  $s$ ,  $\tau$  is disabled in all  $\tau$ -successors of  $s$ .

Claim: For every assertion  $\varphi$  and self-disabling transition  $\tau$

$$\{\varphi \wedge pre(\tau, \varphi)\} \tau \{\varphi \wedge pre(\tau, \varphi)\}$$

is state-valid.

Proof:

Assume  $s \models \varphi \wedge pre(\tau, \varphi)$ .

Then by definition of  $pre(\tau, \varphi)$ , for every  $s'$ ,  $\tau$ -successor of  $s$ ,

$$s' \models \varphi.$$

Since  $\tau$  is self-disabling,  $\tau$  is disabled in all  $\tau$ -successors  $s'$  of  $s$ , and so trivially

$$s' \models pre(\tau, \varphi)$$

Thus for all  $\tau$ -successors  $s'$  of  $s$ ,

$$s' \models \varphi \wedge pre(\tau, \varphi).$$

## Heuristic

If the verification condition

$$\{\chi \wedge \varphi\} \tau \{\varphi\}$$

is not state-valid:

Find  $pre(\tau, \varphi)$  and then

- Strengthening approach:

strengthen  $\varphi$  by adding the conjunct  $pre(\tau, \varphi)$

prove  $\square(\varphi \wedge pre(\tau, \varphi))$

or,

- Incremental approach:

prove  $\square pre(\tau, \varphi)$

and add  $pre(\tau, \varphi)$  to  $\chi$ .

### Note:

$pre(\tau, \varphi)$  is not guaranteed to be an inductive invariant, so the premises of INV have to be checked again.

Example:

**local  $x$ : integer where  $x = 1$**

$$\left[ \begin{array}{l} \ell_0 : \text{request } x \\ \ell_1 : \text{critical} \\ \ell_2 : \text{release } x \end{array} \right]$$

We want to prove

$$\boxed{\square \underbrace{(at\_l_1 \rightarrow x = 0)}_{\varphi}}$$

Problem:

$$\{at\_l_1 \rightarrow x = 0\} \tau_{\ell_0} \{at\_l_1 \rightarrow x = 0\}$$

is not state-valid.

If we use the above heuristic we get

$$pre(\tau_{\ell_0}, \varphi) =$$

$$\forall x', \pi'. \underbrace{(move(\ell_0, \ell_1) \wedge x > 0 \wedge x' = x - 1)}_{\rho_{\ell_0}} \rightarrow \underbrace{(at\_l_1 \rightarrow x' = 0)}_{\varphi'}$$

Example (Con'd):

$$pre(\tau_{\ell_0}, \varphi) = \forall x', \pi'. \underbrace{(move(\ell_0, \ell_1) \wedge x > 0 \wedge x' = x - 1)}_{\rho_{\ell_0}} \rightarrow \underbrace{(at'_{\ell_1} \rightarrow x' = 0)}_{\varphi'}$$

Since

$$move(\ell_0, \ell_1) \rightarrow at_{\ell_0} = \top, at'_{\ell_1} = \top \\ x' = x - 1 \wedge x' = 0 \rightarrow x = 1$$

it simplifies to

$$pre(\tau_{\ell_0}, \varphi): at_{\ell_0} \wedge x > 0 \rightarrow x = 1$$

Strengthened assertion

$$\varphi \wedge pre(\tau_{\ell_0}, \varphi): (at_{\ell_1} \rightarrow x = 0) \wedge (at_{\ell_0} \rightarrow x = 1)$$

what we “guessed” before

Show that  $\varphi \wedge pre(\tau_{\ell_0}, \varphi)$  is inductive  
 (“strengthening approach”)

### Substituted form of $pre(\tau, \varphi)$

Many transition relations have the form

$$\rho_{\tau}: C_{\tau} \wedge \bar{V}' = \bar{E}$$

where  $C_{\tau}$  is the enabled condition of  $\tau$ .

And so

$$pre(\tau, \varphi): \forall \bar{V}'. C_{\tau} \wedge \bar{V}' = \bar{E} \rightarrow \varphi'$$

can be simplified to

$$\forall \bar{V}'. C_{\tau} \rightarrow \varphi[\bar{E}/\bar{V}']$$

replacing all primed variables by its

corresponding expression,

thus the quantifier can be eliminated to obtain

$$pre(\tau, \varphi): C_{\tau} \rightarrow \varphi[\bar{E}/\bar{V}']$$

**Example: Program mux-pet1(Fig. 2.25)**

(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**

$P_1 ::$   $\left[ \begin{array}{l} \ell_1 : \text{noncritical} \\ \ell_2 : (y_1, s) := (T, 1) \\ \ell_3 : \text{await } (\neg y_2) \vee (s \neq 1) \\ \ell_4 : \text{critical} \\ \ell_5 : y_1 := F \end{array} \right]$

||

$m_0$  : **loop forever do**

$P_2 ::$   $\left[ \begin{array}{l} m_1 : \text{noncritical} \\ m_2 : (y_2, s) := (T, 2) \\ m_3 : \text{await } (\neg y_1) \vee (s \neq 2) \\ m_4 : \text{critical} \\ m_5 : y_2 := F \end{array} \right]$

**Example: Program mux-pet1 (Fig. 2.25) (Con'd)**

We want to prove mutual exclusion:

$$\boxed{\square \underbrace{\neg(at_{\ell_4} \wedge at_{m_4})}_{\psi}}$$

Bottom-up invariants:

$$\varphi_0: s = 1 \vee s = 2$$

$$\varphi_1: y_1 \leftrightarrow at_{\ell_3..5}$$

$$\varphi_2: y_2 \leftrightarrow at_{m_3..5}$$

Problem: the verification conditions

$$\{\varphi_0 \wedge \varphi_1 \wedge \varphi_2 \wedge \psi\} \ell_3 \{\psi\}$$

$$\{\varphi_0 \wedge \varphi_1 \wedge \varphi_2 \wedge \psi\} m_3 \{\psi\}$$

are not state-valid

**Example: Program mux-pet1 (Fig. 2.25) (Con'd)**

$$pre(\tau_{l_3}, \psi): \forall \pi': \underbrace{move(l_3, l_4) \wedge (\neg y_2 \vee s \neq 1)}_{\rho_{l_3}} \rightarrow \underbrace{\neg(at'_{l_4} \wedge at'_{m_4})}_{\psi'}$$

since

$$move(l_3, l_4) \text{ implies } at'_{l_4} = T, at'_{m_4} = at_{m_4}$$

$pre(\tau_{l_3}, \psi)$  simplifies to:

$$at_{l_3} \wedge (\neg y_2 \vee s \neq 1) \rightarrow \neg at_{m_4}$$

$$\boxed{\varphi_3: at_{l_3} \wedge at_{m_4} \rightarrow y_2 \wedge s = 1}$$

$pre(\tau_{m_3}, \psi): \forall \pi' \dots\dots$

simplifies to:

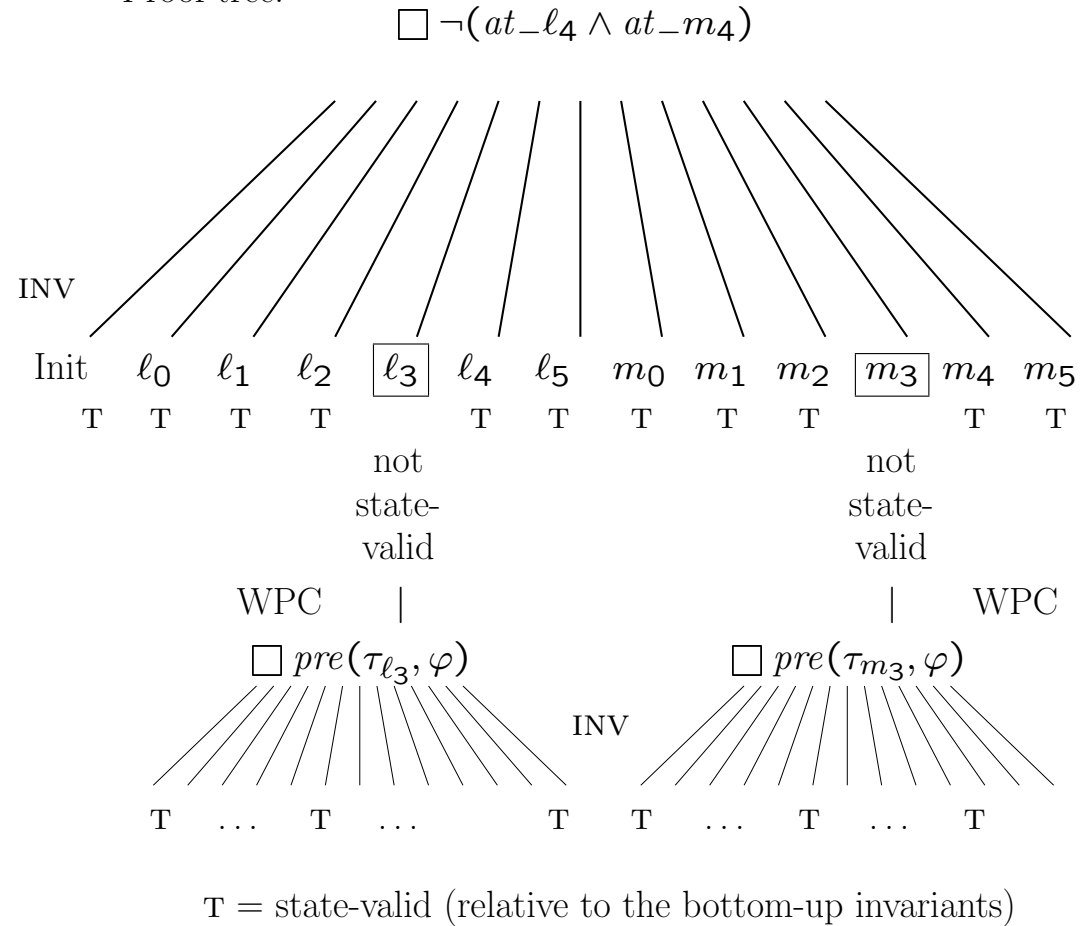
$$\boxed{\varphi_4: at_{l_4} \wedge at_{m_3} \rightarrow y_1 \wedge s = 2}$$

Show that  $\varphi_3: pre(\tau_{l_3}, \psi)$  and  $\varphi_4: pre(\tau_{m_3}, \psi)$  are inductive relative to  $\varphi_0 \wedge \varphi_1 \wedge \varphi_2$  (“incremental approach”)

Then show that  $\psi$  is inductive relative to  $\varphi_0 \wedge \dots \wedge \varphi_4$ .

**Example: Program mux-pet1 (Fig. 2.25) (Con'd)**

Proof tree:



**Example: *pre* may never terminate**

The transition is

$$\rho_\tau : x' = x + y \wedge y' = y$$

The property is

$$\varphi : x \geq 0$$

The VC is

$$\underbrace{x' = x + y \wedge y' = y}_{\rho_\tau} \wedge \underbrace{x \geq 0}_{\varphi} \rightarrow \underbrace{x' \geq 0}_{\varphi'}$$

which is not state valid.

**Step 1:** The precondition is

$$pre(\tau, x \geq 0) : \forall x', y' : x' = x + y \wedge y' = y \rightarrow x' \geq 0$$

that is  $y \geq -x$ .

Attempting to prove  $\square pre(\tau, \varphi)$  state valid, the VC

$$\underbrace{x' = x + y \wedge y' = y}_{\rho_\tau} \wedge \underbrace{y \geq -x}_{pre} \rightarrow \underbrace{y' \geq -x'}_{pre'}$$

is not state-valid.

**Step 2:** Compute  $pre(\tau, y \geq -x)$

$$\forall x', y' : \underbrace{x' = x + y \wedge y' = y}_{\rho_\tau} \rightarrow \underbrace{y' \geq -x'}_{pre'}$$

that is  $y \geq -\frac{x}{2}$ .

In general the precondition

$$pre\left(\tau, y \geq -\frac{x}{n}\right) : y \geq -\frac{x}{n+1}$$

Taking the limit as  $n$  approaches infinity, we obtain

$$y \geq 0$$

which is what we want.

## Finite-State Algorithmic Verification

finite-state program  $P$

each  $x \in V$  assumes only finitely many values in all  $P$ -computations

Therefore,

there are only finitely many distinct  $P$ -accessible states.

**Example:**

MUX-PET1 (Fig 2.25) is finite-state program:

$s = 1, 2$

$y_1 = T, F \quad y_2 = T, F$

$\pi$  can assume at most 36 different values

**Example: Program mux-pet1 (Fig. 2.25)**

(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**

$P_1 ::$   $\left[ \begin{array}{l} \ell_1 : \text{noncritical} \\ \ell_2 : (y_1, s) := (T, 1) \\ \ell_3 : \text{await } (\neg y_2) \vee (s \neq 1) \\ \ell_4 : \text{critical} \\ \ell_5 : y_1 := F \end{array} \right]$

||

$m_0$  : **loop forever do**

$P_2 ::$   $\left[ \begin{array}{l} m_1 : \text{noncritical} \\ m_2 : (y_2, s) := (T, 2) \\ m_3 : \text{await } (\neg y_1) \vee (s \neq 2) \\ m_4 : \text{critical} \\ m_5 : y_2 := F \end{array} \right]$

### Algorithm (transition-graph)

For a given finite-state program  $P$   
Incrementally construct the  
state-transition graph  $G_P$ , where each node  
represents a state.

- Initially

Place as nodes in  $G_P$  all initial states  
(satisfy  $\Theta$ )

- Repeat until no new nodes or

new edges can be added to  $G_P$

[ For some  $s \in G_P$ , let  $s_1, \dots, s_k$  be its succes-  
sors  
Add to  $G_P$  all new nodes in  $\{s_1, \dots, s_k\}$   
and draw edges connecting  $s$  to  $s_i$ ,  
 $i = 1, \dots, k$  ]

### **Algorithmic Verification of Invariance**

For assertion  $q$ ,

To check validity of  $\Box q$  over finite-state program  $P$ :

1. Construct the state-transition graph  $G_P$ .
2. Check if  $q$  holds in each state of the graph.

**Example:** Program MUX-SEM (Fig 2.26)

Generates finite state-transition graph (Fig 2.27)

Check assertion

$$\varphi: \neg(at\_l3 \wedge at\_m3)$$

in the graph.

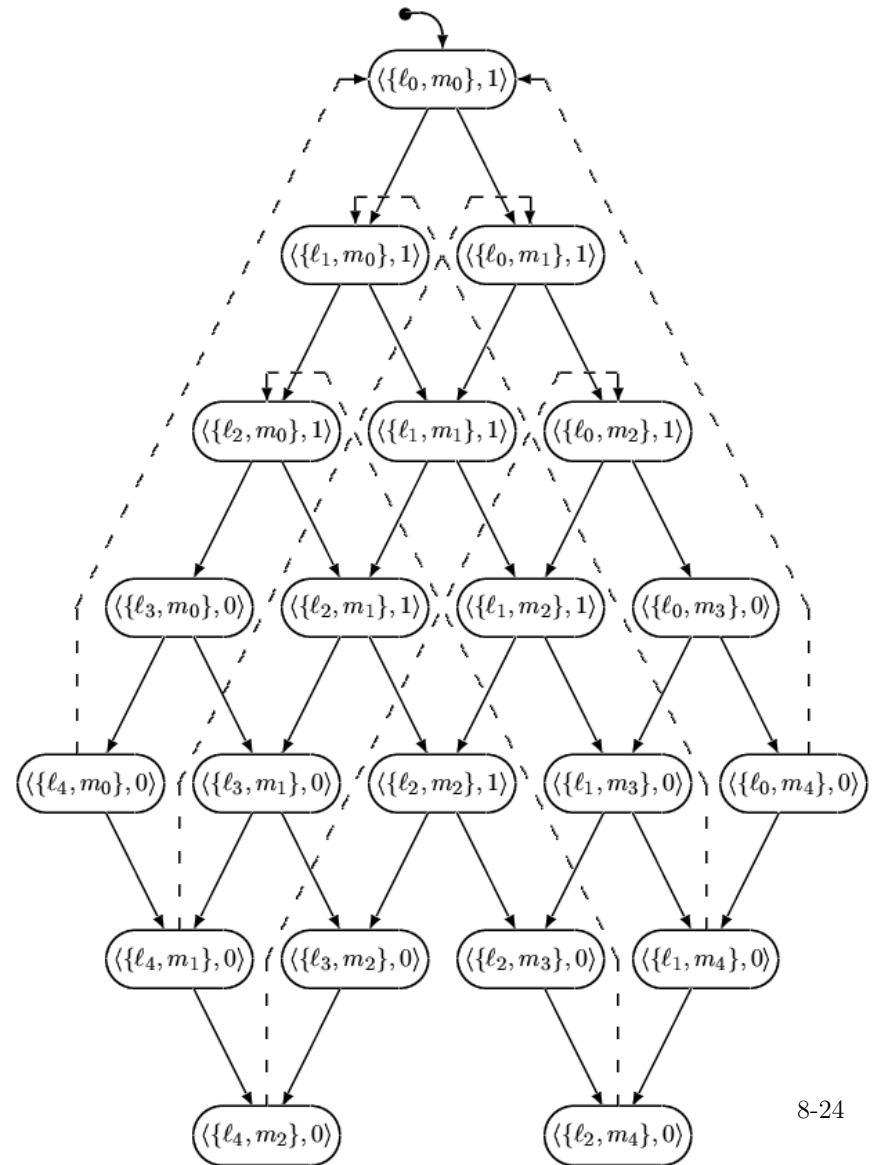
$\varphi$  holds over all accessible states.

Thus,  $\Box \varphi$  for MUX-SEM.

Program MUX-SEM state-transition graph (Fig. 2.27)

Program MUX-SEM (Fig. 2.26)  
 (mutual exclusion by semaphores)

local  $y$ : integer where  $y = 1$

$$P_1 :: \left[ \begin{array}{l} \ell_0: \text{loop forever do} \\ \left[ \begin{array}{l} \ell_1 : \text{noncritical} \\ \ell_2 : \text{request } y \\ \ell_3 : \text{critical} \\ \ell_4 : \text{release } y \end{array} \right] \end{array} \right] \parallel P_2 :: \left[ \begin{array}{l} m_0: \text{loop forever do} \\ \left[ \begin{array}{l} m_1: \text{noncritical} \\ m_2: \text{request } y \\ m_3: \text{critical} \\ m_4: \text{release } y \end{array} \right] \end{array} \right]$$


**Example:** Program MUX-PET1 (Fig 2.25)

State-transition graph  $G_P$  (Fig 2.28)

$(i, j, v)$  means  $\pi: \{l_i, m_j\}, s: v$

No  $y_1, y_2$  since

$y_1 = T$  iff  $3 \leq i \leq 5$

$y_2 = T$  iff  $3 \leq j \leq 5$

Property checked

$$\square \underbrace{\neg(at_{l_4} \wedge at_{m_4})}_{\psi}$$

**Example: Program mux-pet1(Fig. 2.25)**

(Peterson's Algorithm for mutual exclusion)

**local**  $y_1, y_2$ : **boolean** where  $y_1 = F, y_2 = F$   
 $s$  : **integer** where  $s = 1$

$l_0$  : **loop forever do**

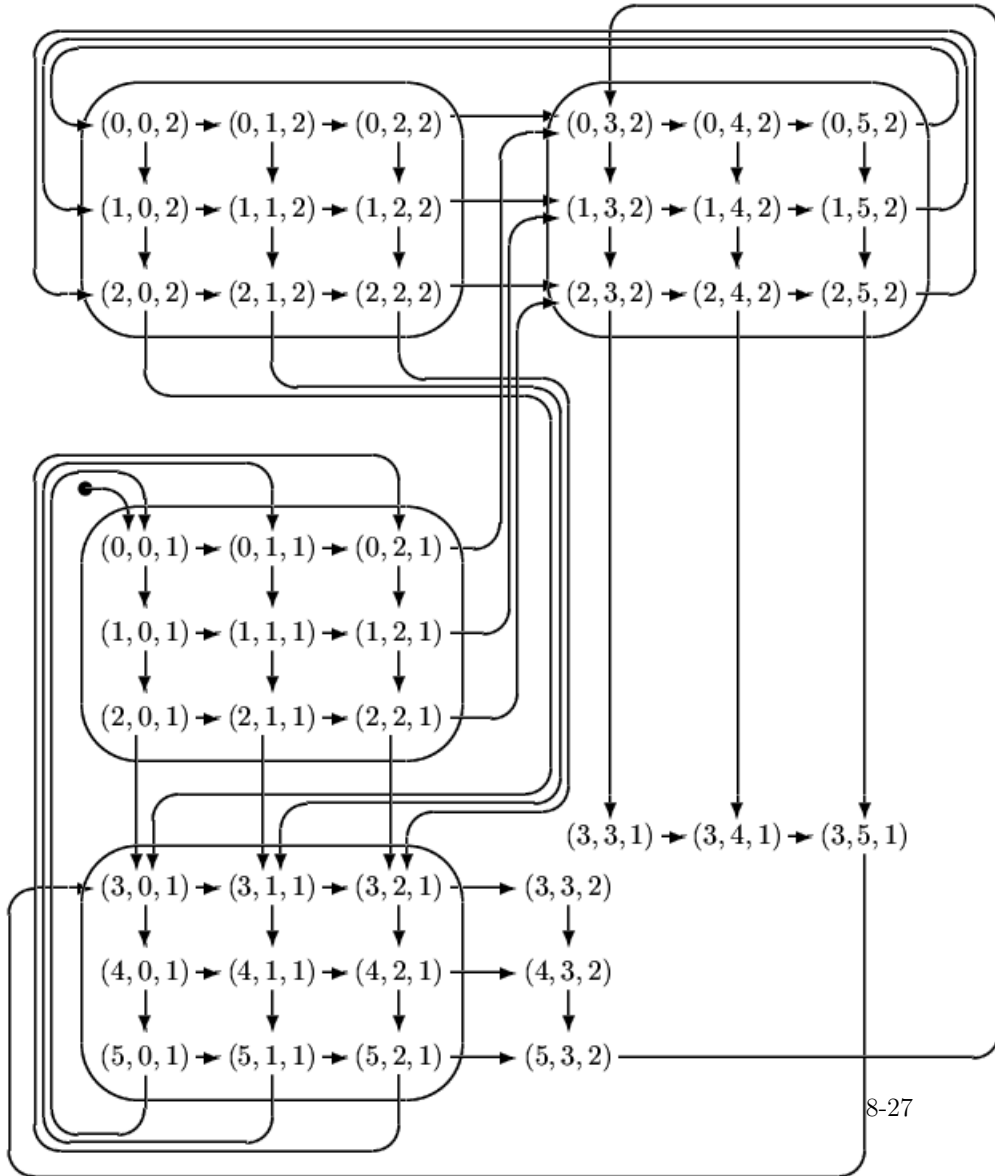
$P_1 ::$   $\left[ \begin{array}{l} l_1 : \text{noncritical} \\ l_2 : (y_1, s) := (T, 1) \\ l_3 : \text{await } (\neg y_2) \vee (s \neq 1) \\ l_4 : \text{critical} \\ l_5 : y_1 := F \end{array} \right]$

||

$m_0$  : **loop forever do**

$P_2 ::$   $\left[ \begin{array}{l} m_1 : \text{noncritical} \\ m_2 : (y_2, s) := (T, 2) \\ m_3 : \text{await } (\neg y_1) \vee (s \neq 2) \\ m_4 : \text{critical} \\ m_5 : y_2 := F \end{array} \right]$

MUX-PET1 State-transition graph (Fig 2.28)



8-27

### Completeness of rule INV

#### Rule INV (general invariance)

For assertions  $\varphi, q$ ,

- I1.  $\models \varphi \rightarrow q$
- I2.  $\models \Theta \rightarrow \varphi$
- I3.  $\models \{\varphi\} \mathcal{T} \{\varphi\}$

---


$$\models \Box q$$

#### Theorem (Relative completeness of rule INV)

For every assertion  $q$  such that

$$\Box q \text{ is } P\text{-valid}$$

there exists an assertion  $\varphi$  such that I1 – I3 are provable from state validities

8-28

We actually show

“completeness relative to  
first-order reasoning”

taking all state-valid assertions as axioms

### Outline of proof

Given FTS  $P$  with system variables (program + control variables)

$$\bar{y} = (y_1, \dots, y_m)$$

- Assume  $\Box q$  is  $P$ -valid, i.e.,  
( $\dagger$ )  $q$  holds over every  $P$ -accessible state
- Construct (to be shown) accessibility assertion  
 $acc_P(\bar{y})$   
such that for any state  $s$ ,  
(\*)  $s$  is  $P$ -accessible state iff  $s \models acc_P$
- Take  $\varphi = acc_P$

We have to show :

1.  $acc_P$  satisfies I1 – I3
2.  $acc_P$  can be “constructed”

### 1. $acc_P$ satisfies I1 – I3

- Premise I1:  $\underbrace{acc_P}_{\varphi} \rightarrow q$

$$s \models acc_P \stackrel{(*)}{\Rightarrow} s \text{ is } P\text{-accessible state}$$

$$\stackrel{(\dagger)}{\Rightarrow} s \models q$$

Thus

$$\underbrace{acc_P}_{\varphi} \rightarrow q$$

is state-valid

- Premise I2:  $\Theta \rightarrow \underbrace{acc_P}_{\varphi}$

$$s \models \Theta \Rightarrow s \text{ is } P\text{-accessible}$$

$$\stackrel{(*)}{\Rightarrow} s \models \underbrace{acc_P}_{\varphi}$$

Thus

$$\Theta \rightarrow \underbrace{acc_P}_{\varphi}$$

is state-valid

- Premise I3: for every  $\tau \in \mathcal{T}$ ,

$$\rho_\tau \wedge acc_P \rightarrow acc'_P$$

where  $acc'_P = acc_P(\bar{y}')$ .

Take  $s'$  to be a  $\bar{y}$ -variant of  $s$  ( $s$  agrees with  $s'$  on all variables other than  $\bar{y}$ ) and for each  $y_i$  take

$$s'[y_i] = s[y_i]$$

Then

$$\left. \begin{array}{l} s \models \rho_\tau \Rightarrow s' \text{ is a } \tau\text{-successor of } s \\ s \models acc_P \xrightarrow{(*)} s \text{ is } P\text{-accessible} \end{array} \right\} \Rightarrow$$

$$\Rightarrow s' \text{ is } P\text{-accessible}$$

$$\xrightarrow{(*)} s' \models acc_P$$

$$\Rightarrow s \models acc'_P$$

Example:

$$V: \{y\} \quad \Theta: y = 0$$

$$\mathcal{T}: \{\tau_I, \tau\}, \text{ where } \rho_\tau: y' = y + 2$$

$$\text{For this program: } acc_P(y): y \geq 0 \wedge even(y)$$

## 2. Construction of $acc_P$

Assume assertion language includes

dynamic array  $\underline{a}$  over  $D$

Array  $\underline{a}$  is viewed as function,

$$a: [1..n] \mapsto D$$

where  $n$  is the size of the array

The assumption is not essential

We can use Gödel numbering

$$(k_1, \dots, k_n) \mapsto n = p_1^{k_1} \dots p_n^{k_n}$$

where  $p_i$  is the  $i$ th prime number

Case: single-variable  $y$

Define

$$acc_P(y): (\exists n > 0) (\exists a \in [1..n] \mapsto D) . \\ \text{init} \wedge \text{last} \wedge \text{evolve}$$

where

$$\text{init}: \Theta(a[1])$$

$$\text{last}: a[n] = y$$

$$\text{evolve}: \forall i . 1 \leq i < n . \bigvee_{\tau \in \mathcal{T}} \rho_{\tau}(a[i], a[i+1])$$

i.e., there exists an array  $a$ , such that

- $a[1]$  is an initial state
- $a[n]$  has value  $y$  (last element)
- every two consecutive elements are related by some transition relation

array  $a$  represents a prefix

$$s_1, \dots, s_n$$

of a computation where  $a[i]$  stands for  
the value of  $y$  at state  $s_i$

Claim:

For any value  $d \in D$ ,

$$acc_P(d) = \text{T}$$

iff

$d$  is a possible value of  $y$  in a  $P$ -accessible state

$acc_P(d)$  asserts the existence of a computation prefix that leads to a state where  $y = d$ .

**Example:** Transition system EVEN

$V: \{y\}$  ranges over  $\mathbb{Z}$  (the integers)

$\Theta: y = 0$

$\rho_\tau: y' = y + 2$

$acc_P(y):$

$(\exists n > 0)(\exists a \in [1..n] \mapsto \mathbb{Z}).$

$$\left( a[1] = 0 \wedge a[n] = y \wedge \right. \\ \left. \forall i. 1 \leq i < n. a[i+1] = a[i] + 2 \right)$$

simplifies to

$(\exists n > 0)(\exists a \in [1..n] \mapsto \mathbb{Z}).$

$$\left( a[n] = y \wedge \right. \\ \left. \forall i. 1 \leq i \leq n. a[i] = 2 \cdot (i - 1) \right)$$

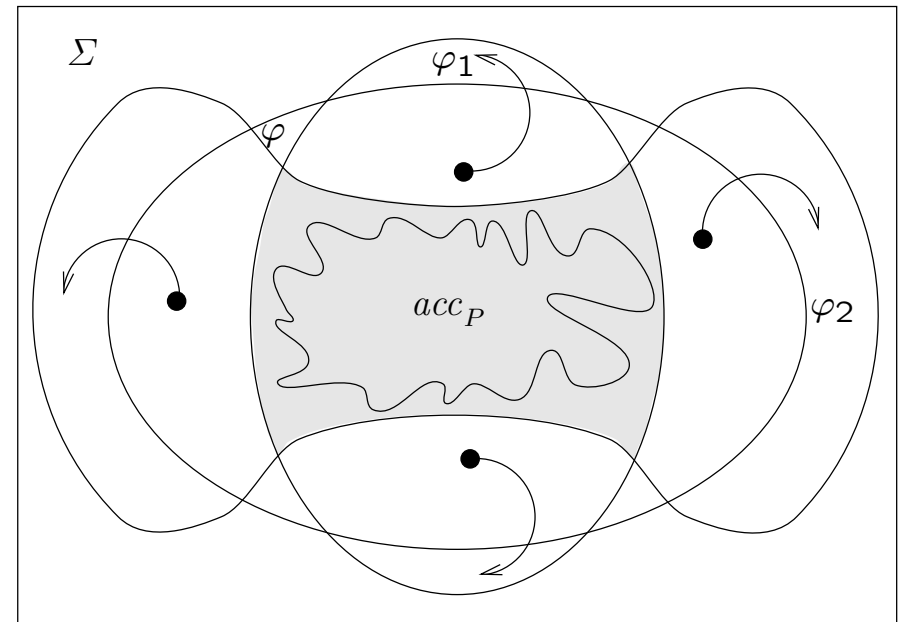
simplifies to

$$y \geq 0 \wedge \text{even}(y)$$

Precisely characterizes the values that  $y$  may assume in  $P$ -accessible states of EVEN

## Discussion

Although the assertion  $acc_P$  is inductive and strengthens any  $P$ -invariant, it is not very useful in practice.



The shaded area is preserved by all transitions. Its description is much simpler than that of  $acc_P$ .

Multivariable  $\bar{y} = (y_1, \dots, y_m)$  case

Use 2-dimensional array  $a$

$$\begin{array}{cccc}
 \underline{y_1} & & & \underline{y_m} \\
 a[1, 1] & . & . & . & a[1, m] \\
 a[2, 1] & . & . & . & a[2, m] \\
 & \vdots & & & \vdots \\
 & \vdots & & & \vdots \\
 & \vdots & & & \vdots
 \end{array}$$

**Example:** Transition system FACT

$y, z$  ranging over  $\mathbb{N}$  (the nonnegative integers)

$$\Theta: y = 1 \wedge z = 1$$

$$\rho_\tau: y' = y + 1 \wedge z' = (y + 1) \cdot z$$

**Construction of  $acc_P$ :**

$$(\exists n > 0)(\exists a \in [1..n] \times [1, 2] \mapsto \mathbb{N}).$$

$$\left( \begin{array}{l}
 a[1, 1] = 1 \wedge a[1, 2] = 1 \wedge \\
 a[n, 1] = y \wedge a[n, 2] = z \\
 \wedge \\
 \forall i: 1 \leq i < n: a[i + 1, 1] = a[i, 1] + 1 \wedge \\
 a[i + 1, 2] = (a[i, 1] + 1) \cdot a[i, 2]
 \end{array} \right)$$

$(\exists n > 0)(\exists a \in [1..n] \times [1, 2] \mapsto \mathbb{N}) .$

$$\left( \begin{array}{l} a[1, 1] = 1 \wedge a[1, 2] = 1 \wedge \\ a[n, 1] = y \wedge a[n, 2] = z \\ \wedge \\ \forall i: 1 \leq i < n: a[i + 1, 1] = a[i, 1] + 1 \wedge \\ a[i + 1, 2] = (a[i, 1] + 1) \cdot a[i, 2] \end{array} \right)$$

simplifies to

$(\exists n > 0)(\exists a \in [1..n] \times [1, 2] \mapsto \mathbb{N}) .$

$$\left( \begin{array}{l} a[n, 1] = y \wedge a[n, 2] = z \\ \wedge \\ \forall i: 1 \leq i \leq n: a[i, 1] = i \wedge a[i, 2] = i! \end{array} \right)$$

simplifies to

$$\boxed{y \geq 1 \wedge z = y!}$$

Precisely characterizes the  $P$ -accessible states  
for the transition system FACT