CS256/Spring 2008 — Lecture #14

Zohar Manna

Satisfiability over a finite-state program

<u>P-validity problem (of φ)</u>

Given a finite-state program P
and formula φ,

is φ P-valid?

i.e. do all P-computations satisfy φ?

<u>P-satisfiability problem (of φ)</u>

Given a finite-state program P
and formula φ

is φ P-satisfiable?

i.e., does there exist a P-computation which satisfies φ?

To determine whether φ is P-valid,
<u>it suffices</u> to apply an algorithm for
deciding if there is a P-computation
that satisfies $\neg\varphi$.

The Idea

To check P-satisfiability of φ,
we combine the tableau T_φ and the
transition graph G_P into one product graph,
called the behavior graph $\mathcal{B}_{(P,\varphi)}$,
and search for paths

$$(s_0, A_0),\ (s_1, A_1),\ (s_2, A_2),\ \ldots$$

that satisfy the two requirements:

- $\sigma \vDash \varphi$:

  there exists a fulfilling path

  $\pi:\ A_0, A_1, \ldots$

  in the tableau T_φ such that $\varphi \in A_0$.

- σ is a P-computation:

  there exists a fair path

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

  in the transition graph G_P.

State transition graph G_P: Construction

- Place as nodes in G_P all initial states s ($s \Vdash \Theta$)

- Repeat

  for some $s \in G_P,\ \tau \in \mathcal{T}$,
  add all its τ-successors s' to G_P
  if not already there,
  and add edges between s and s'.

  Until no new states or edges can be added.

If this procedure terminates, the system is finite-state.

Example: Program mux-pet1 (Fig. 3.4)

(Peterson's Algorithm for mutual exclusion)

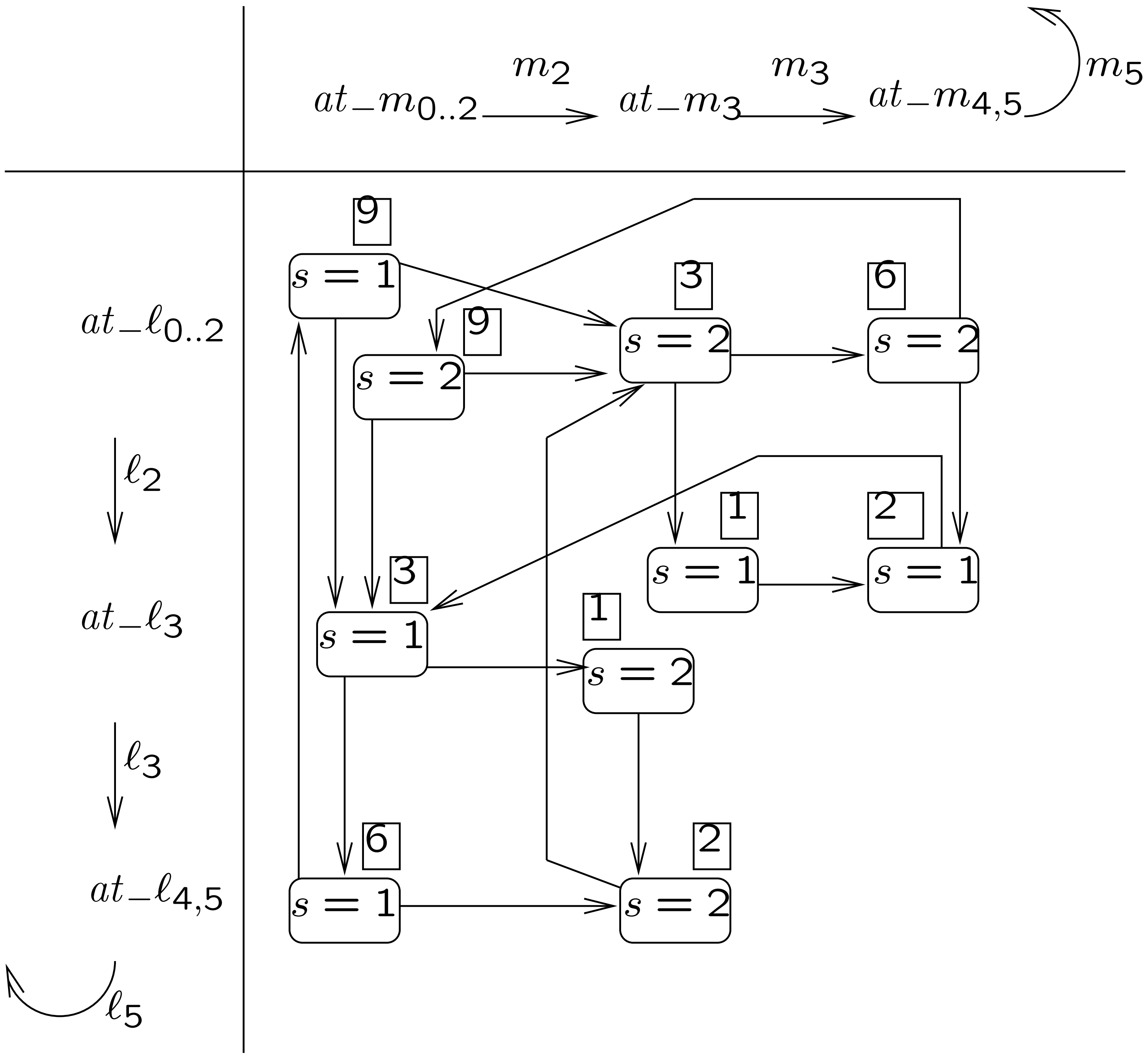
$$
\begin{array}{llll}
\textbf{local} & y_1, y_2 : & \textbf{boolean} & \textbf{where } y_1 = \text{F}, y_2 = \text{F} \\
& s \quad : & \textbf{integer} & \textbf{where } s = 1
\end{array}
$$

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

$$||$$

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

Abstract state-transition graph for MUX-PET1



We use $y_1 \Leftrightarrow at\_\ell_{3..5}$

$y_2 \Leftrightarrow at\_m_{3..5}$

Some states have been lumped together:
a super-state labeled by $\boxed{i}$ represents i states

MUX-PET1 has 42 reachable states.

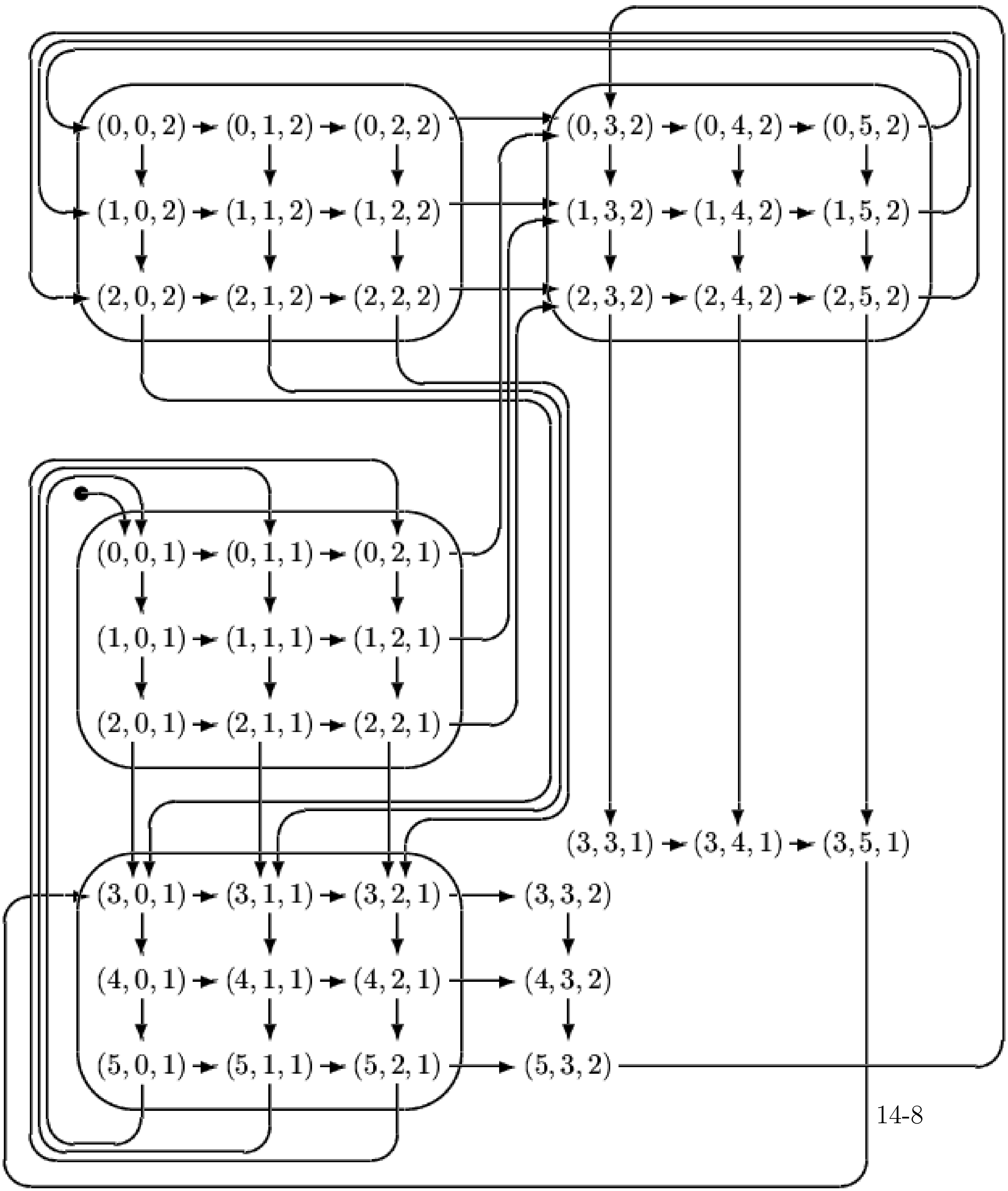
Based on this graph it is straightforward to check the properties

$$\psi_1: \quad \Box\, \neg(at\_\ell_4 \wedge at\_m_4)$$

$$\psi_2: \quad \Box(at\_\ell_3 \wedge \neg at\_m_3 \rightarrow s = 1)$$

$$\psi_3: \quad \Box(at\_m_3 \wedge \neg at\_\ell_3 \rightarrow s = 2)$$

MUX-PET1 Full state-transition graph (l_i, m_j, s)



Definitions

- For atom A, $state(A)$ is the conjunction of all state formulas in A
  (by R_{sat}, $state(A)$ must be satisfiable)

- For $A \in T_\varphi$,
  $\underline{\delta(A)\ }$ denotes the set of successors of A in T_φ

- atom A is consistent with state s
  if $s \Vdash state(A)$,

  i.e. s satisfies all state formulas in A.

- ϑ: $A_0, A_1, \ldots$ path in T_φ
  σ: $s_0, s_1, \ldots$ computation of P

  ϑ is a trail of T_φ over σ if
  A_j is consistent with s_j, for all $j \geq 0$

Behavior Graph

For finite-state program P and formula φ,
we construct the (P, φ)-behavior graph

$$\mathcal{B}_{(P,\varphi)} \quad \approx \quad G_P \times T_\varphi^- \text{ (pruned)}$$

such that

- nodes are labeled by (s, A)

  where s is a state from G_P and
  A is an atom from T_φ consistent with s.

- edges
  There is an edge

  $$s, A \xrightarrow{\tau} s', A'$$

  if and only if $s' \in \tau(s)$ and $A' \in \delta(A)$

  $$s \xrightarrow{\tau} s' \qquad A \longrightarrow A'$$

  in G_P $\qquad$ in T_φ

- initial φ-node $\quad (s, A)$

  if s is an initial state $(s \Vdash \Theta)$
  and A is an initial φ-atom $(\varphi \in A)$

  It is marked s, A

Algorithm behavior-graph

(constructing $\mathcal{B}_{(P,\varphi)}$)

- Place in $\mathcal{B}$ all initial φ-nodes (s, A)

  (s initial state of P,
  A initial φ-atom in T_φ^-
  A consistent with s)

- Repeat until no new nodes or new edges can be added:

  Let (s, A) be a node in $\mathcal{B}$
  $\tau \in \mathcal{T}$ a transition
  (s', A') a pair s.t.
  s' is a τ-successor of s
  $A' \in \delta(A)$ in pruned T_φ^-
  A' consistent with s'

  - Add (s', A') to $\mathcal{B}$, if not already there
  - Draw a τ-edge from (s, A) to (s', A'), if not already there

Example: Given FTS LOOP

$$\Theta: \ x = 0$$

$$\mathcal{T} = \{\tau, \tau_I\}$$

with τ_I (idling)

τ where ρ_τ: $x' = (x+1) mod 4$

$$\mathcal{J}: \quad \{\tau\}$$

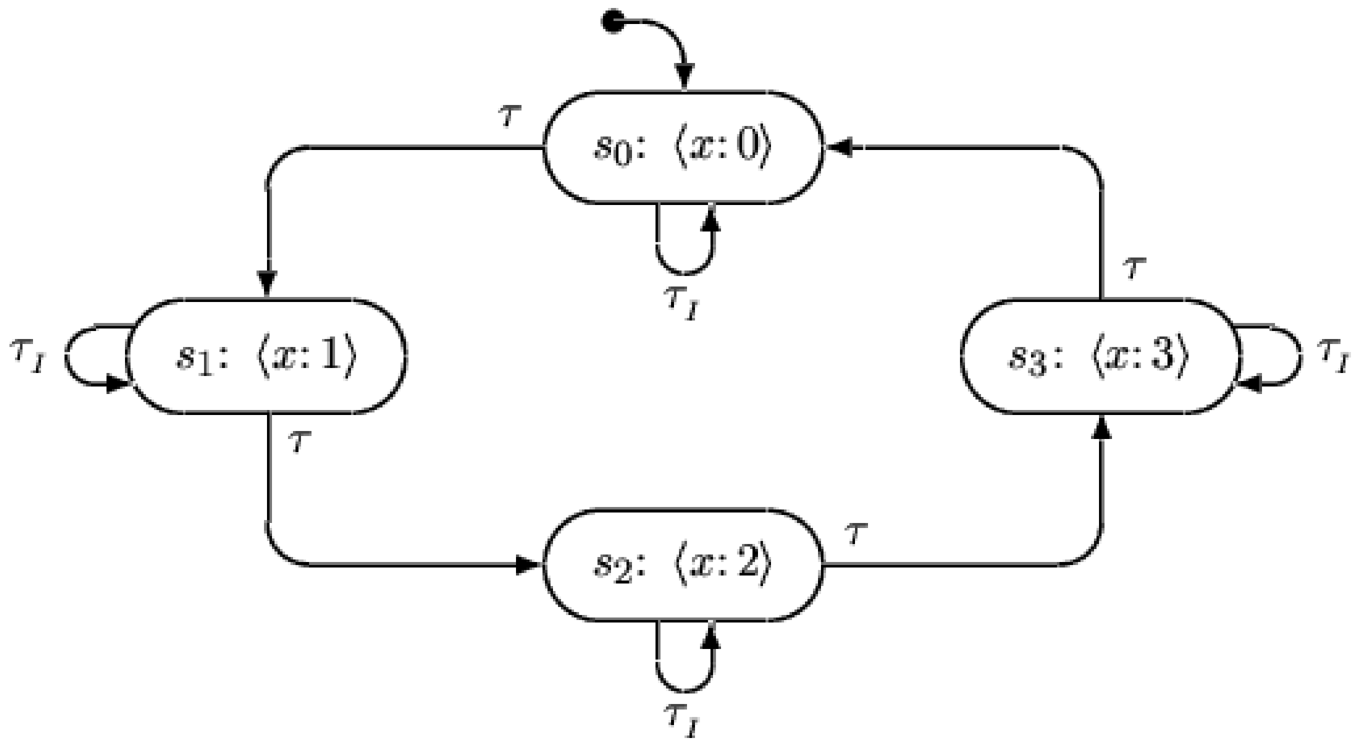
Check P-satisfiability of $\boxed{\psi_3: \ \diamond \square (x \neq 3)}$

state-transition graph G_{LOOP} (Fig 5.9)

pruned $T^-_{\psi_3}$ (Fig 5.8)

Behavior graph $\mathcal{B}_{(\text{LOOP},\psi_3)}$ (Fig 5.10)

Fig. 5.9. State-transition graph G_{LOOP}



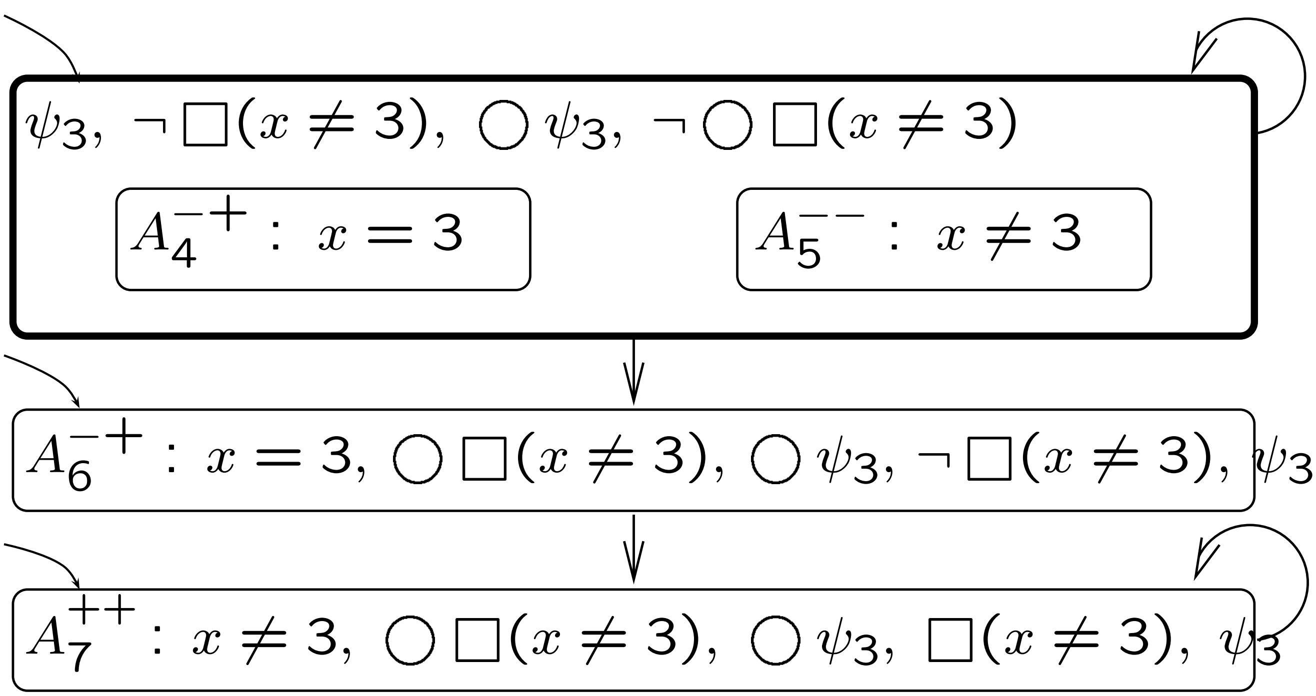
Pruned tableau $T^{-}_{\psi_3}$ (Fig. 5.8)

Eliminating

- MSCS's not reachable from an initial ψ_3-atom and
- non-fulfilling terminal MSCS's

Promising formulas:

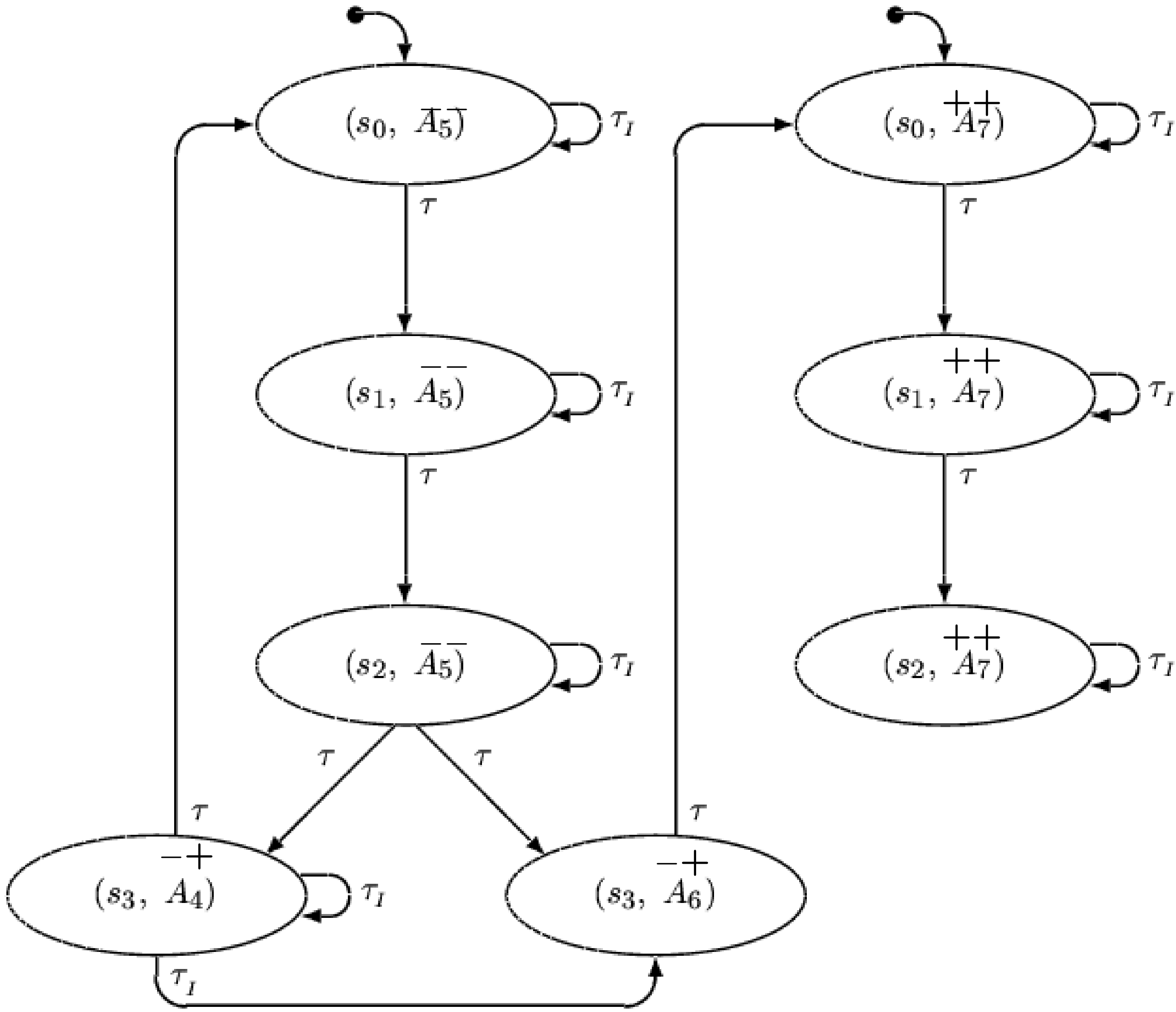
$$\diamond \square (x \neq 3) \quad \text{promising} \quad \square(x \neq 3)$$
$$\neg \square (x \neq 3) \quad \text{promising} \quad (x = 3)$$



Two non-transient MSCS's:

$\{A_4^{-+}, A_5^{--}\}$	not fulfilling
$\{A_7^{++}\}$	fulfilling

Behavior graph $\mathcal{B}_{(\text{LOOP},\psi_3)}$ (Fig 5.10)



Example: Given FTS ONE:

Θ: $x = 0$

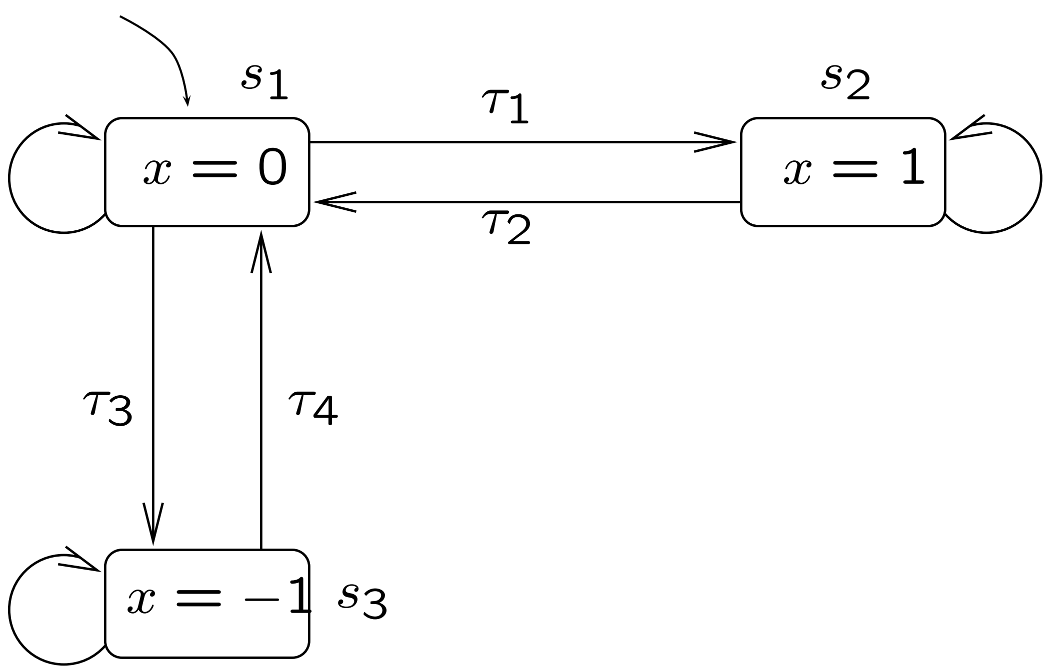
$\mathcal{T}$: $\{\tau_1, \tau_2, \tau_3, \tau_4, \tau_I\}$

with
$$\begin{aligned} \rho_{\tau_1} &: \quad x = 0 \wedge x' = 1 \\ \rho_{\tau_2} &: \quad x = 1 \wedge x' = 0 \\ \rho_{\tau_3} &: \quad x = 0 \wedge x' = -1 \\ \rho_{\tau_4} &: \quad x = -1 \wedge x' = 0 \end{aligned}$$

$\mathcal{J}$: $\emptyset$

$\mathcal{C}$: $\{\tau_1, \tau_3\}$

Transition graph G_{ONE}



We want to know whether

$$\boxed{\varphi : \; \square \diamond (x = 1)}$$

is valid over ONE.

Check P-satisfiability of

$$\boxed{\neg\varphi : \; \underbrace{\diamond \square (x \neq 1)}_{\psi}}$$

Φ^{+}_{ψ} : $\{\psi,\ \bigcirc \psi,\ \square(x \neq 1),\ \bigcirc \square(x \neq 1),\ x = 1\}$

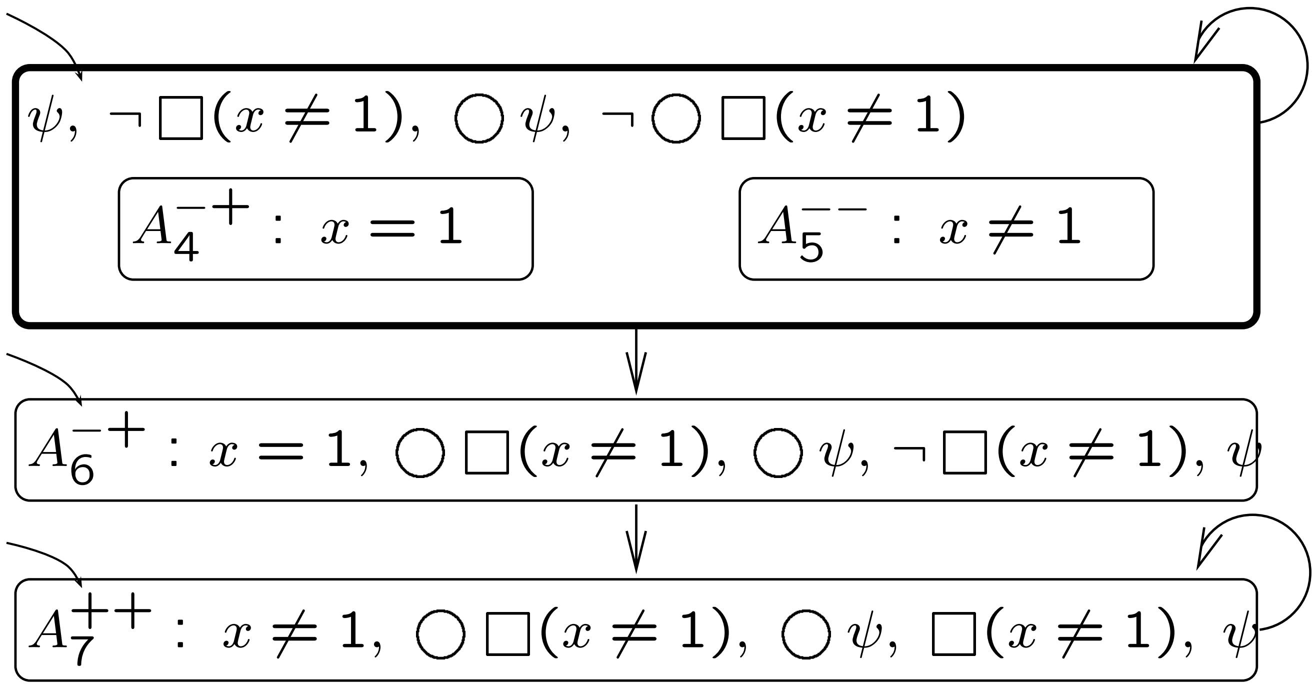
basic formulas: $\{\bigcirc \psi,\ \bigcirc \square(x \neq 1),\ x = 1\}$

Promising formulas:

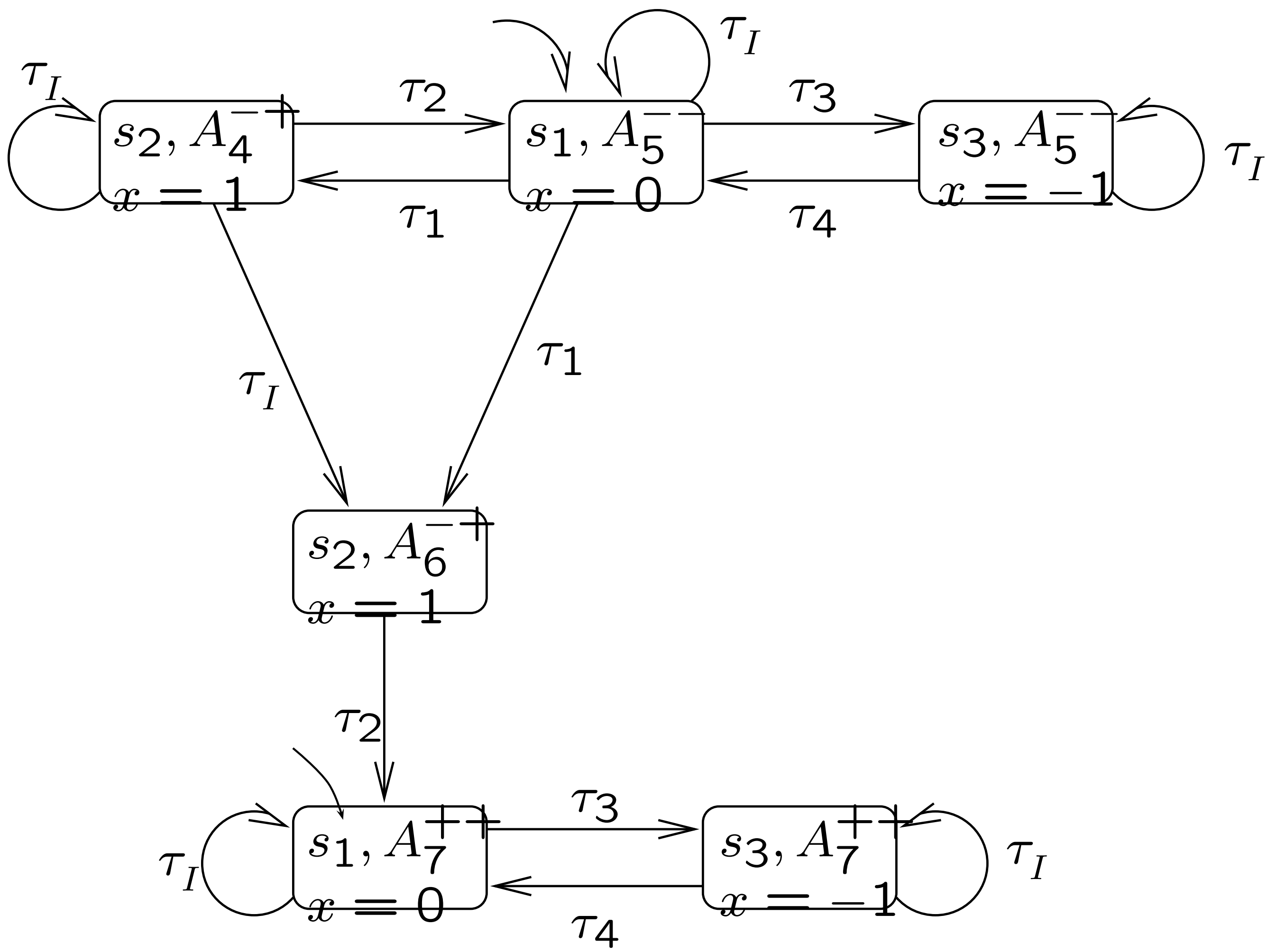
$$\psi_1 : \psi = \diamond \square (x \neq 1) \quad \text{promising} \quad r_1 : \; \square(x \neq 1)$$

$$\psi_2 : \; \neg \square (x \neq 1) \quad \text{promising} \quad r_2 : \; x = 1$$

Pruned tableau T_{ψ}^{-}



Behavior graph $\mathcal{B}_{(\text{ONE}, \Diamond \Box (x \neq 1))}$



Two non-transient MSCS's:

$\{(s_2, A_4^{-+}),\ (s_1, A_5^{--}),\ (s_3, A_5^{--})\}$: not fulfilling,

$\{(s_1, A_7^{++}), (s_3, A_7^{++})\}$: fulfilling

Paths of $\mathcal{B}_{(P,\varphi)}$

Claim 5.9 (paths of $\mathcal{B}_{(P,\varphi)}$)

The infinite sequence

$$\pi\text{:}\ \underbrace{(s_0, A_0)}_{\varphi\text{-initial}},\ (s_1, A_1),\ \ldots$$

is a path in $\mathcal{B}_{(P,\varphi)}$

iff

σ_π: $s_0, s_1, \ldots$ is a run of P
(i.e. computation of P less fairness)

ϑ_π: $A_0, A_1, \ldots$ is a trail of T_φ over σ_π
(i.e. A_j consistent with s_j, for all $j \geq 0$)

Example: In $\mathcal{B}_{(\text{LOOP},\psi_3)}$ (Fig. 5.10)

$$\pi\text{:}\ \Big((s_0, A_5),\ (s_1, A_5),\ (s_2, A_5),\ (s_3, A_4)\Big)^\omega$$

induces

σ_π: $(s_0, s_1, s_2, s_3)^\omega$ run of LOOP

ϑ_π: $(A_5, A_5, A_5, A_4)^\omega$ trail of T_{ψ_3} over σ_π

`Proposition 5.10` (P-satisfiability by path)

P has a computation satisfying φ

iff

there is an infinite φ-initialized path π
in $\mathcal{B}_{(P,\varphi)}$ s.t.

σ_π is a $\underline{P\text{-computation}}$ (fair run of P)

ϑ is a $\underline{\text{fulfilling trail}}$ over σ_π

Searching for "good" paths in $\mathcal{B}_{(P,\varphi)}$

— not practical.

Definitions

For behavior graph $\mathcal{B}_{(P,\varphi)}$

- node (s', A') is a <u>τ-successor</u> of (s, A) if $\mathcal{B}_{(P,\varphi)}$ contains τ-edge connecting (s, A) to (s', A')

- transition τ is <u>enabled</u> on node (s, A) if τ is enabled on state s

Definitions (Con't)

For scs $S \subseteq \mathcal{B}_{(P,\varphi)}$:

- Transition τ is taken in S if there exists two nodes $(s, A), (s', A') \in S$ s.t.

  (s', A') is a τ-successor of (s, A)

- S is $\left\{\begin{matrix}\underline{\text{just}} \\ \underline{\text{compassionate}}\end{matrix}\right\}$ if every $\left\{\begin{matrix}\text{just} \\ \text{compassionate}\end{matrix}\right\}$ transition $\tau \left\{\begin{matrix}\in \mathcal{J} \\ \in \mathcal{C}\end{matrix}\right\}$ is either taken in S or is disabled on $\left\{\begin{matrix}\text{some node} \\ \text{all nodes}\end{matrix}\right\}$ in S

- S is fair if it is both just and compassionate

- S is fulfilling if every promising formula $\psi \in \Phi_\psi$ is fulfilled by some atom A, s.t.

  $(s, A) \in S$ for some state s

- S is adequate if it is fair and fulfilling

Adequate SCS's

Proposition 5.11 (adequate SCS and satisfiability)

Given a finite-state program P and temporal formula φ.
φ is P-satisfiable

iff

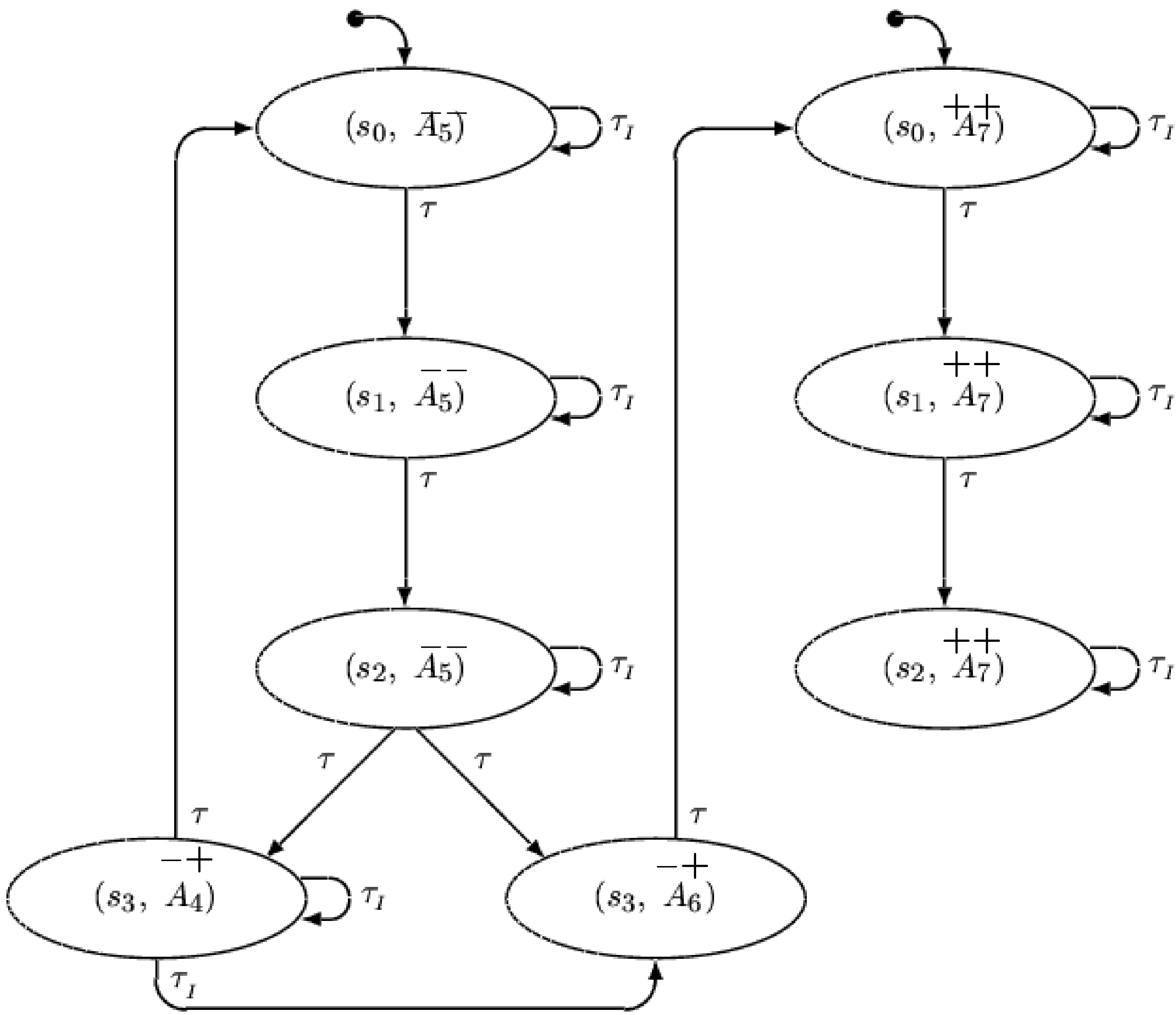
$\mathcal{B}_{(P,\varphi)}$ has an adequate SCS

Example: Consider LOOP and

$$\boxed{\psi_3\text{: } \diamondsuit \square (x \neq 3)}$$

Is ψ_3 LOOP-satisfiable?
Check the SCS's in $\mathcal{B}_{(\mathrm{LOOP},\psi_3)}$ (Fig. 5.10)

Behavior graph $\mathcal{B}_{(\text{LOOP},\psi_3)}$ (Fig 5.10)



Example (Con't)

- $\{ (s_0, A_5^{--}), (s_1, A_5^{--}), (s_2, A_5^{--}), (s_3, A_4^{-+}) \}$
  is fair but not fulfilling

- $\{ (s_0, A_7^{++})\},\ \{(s_1, A_7^{++})\},\ \{(s_2, A_7^{++})\}$

  each is fulfilling but not fair
  Not just with respect to transition τ

- $\{(s_3, A_6^{-+})\}$

  is neither fair (unjust toward τ) $\boxed{\text{nor}}$
  fulfilling (being transient)

No adequate subgraphs in $\mathcal{B}_{(\text{LOOP}, \psi_3)}$

Therefore, by `proposition 5.11`, LOOP has no computation that satisfies ψ_3: $\Diamond \Box (x \neq 3)$

Example: Consider LOOP and

$$\boxed{\varphi_3:\ \Box\Diamond(x=3)}$$

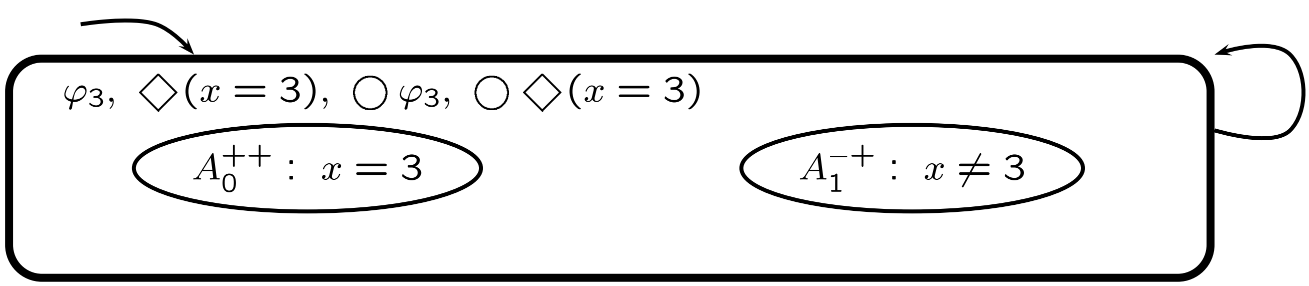
Is φ_3 LOOP-satisfiable?

Promising formulas :

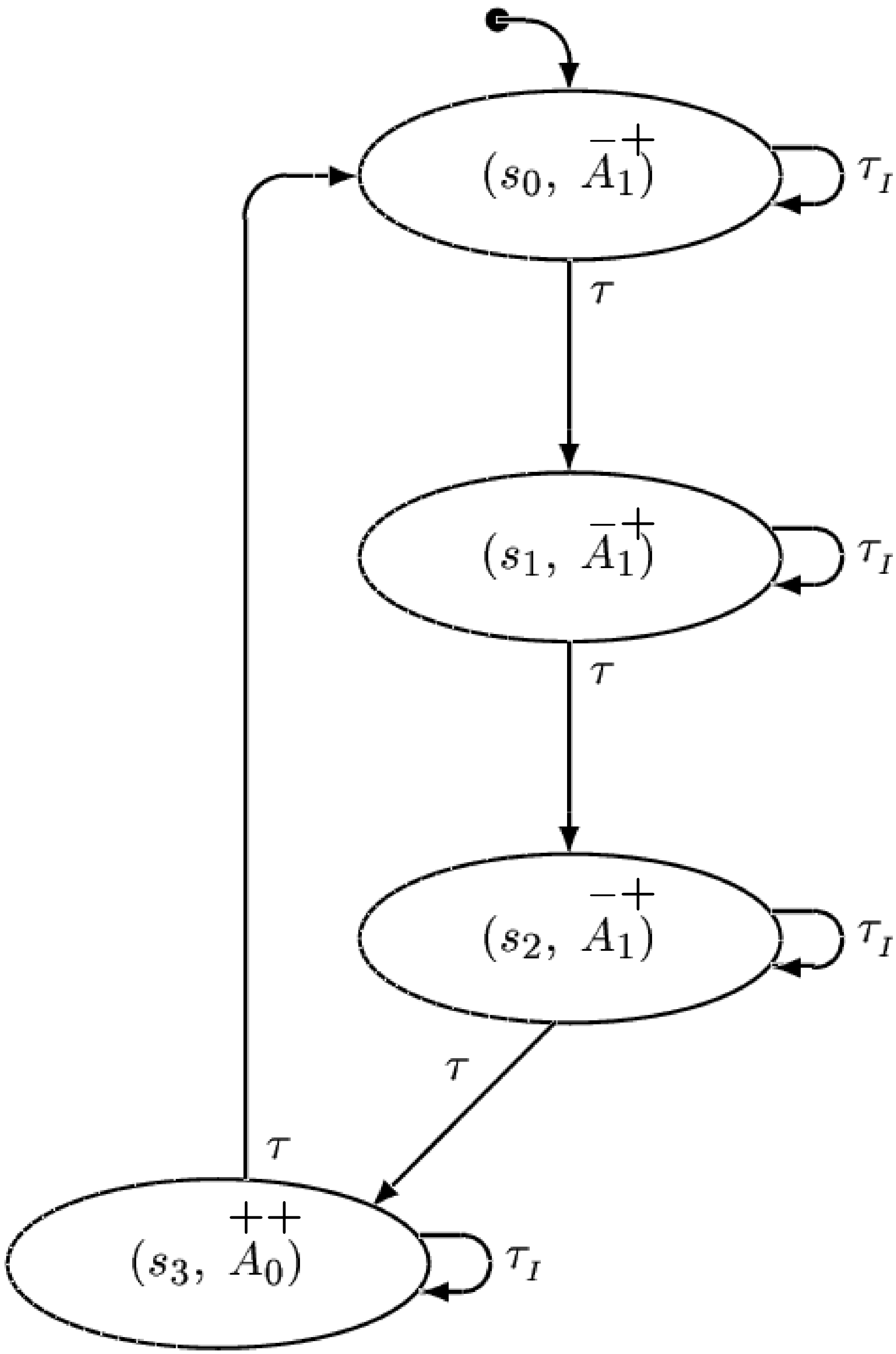
$$\Diamond(x=3) \quad \text{promising} \quad (x=3)$$

$$\neg\Box\Diamond(x=3) \quad \text{promising} \quad \neg\Diamond(x=3)$$

Pruned tableau T_{φ_3} (Fig. 5.6)



Behavior graph $\mathcal{B}_{(\mathbf{LOOP},\varphi_3)}$ (Fig. 5.11)



$$S = \{\,(s_0, A_1^{-+}),\ (s_1, A_1^{-+}),\ (s_2, A_1^{-+}),\ (s_3, A_0^{++})\}$$

is an adequate subgraph:

fair (τ taken in S)
fulfilling

Therefore, by `proposition 5.11`, program LOOP has a computation satisfying φ_3: $\Box \Diamond (x = 3)$

The periodic computation σ: $(x{:}\,0, x{:}\,1, x{:}\,2, x{:}\,3)^{\omega}$ satisfies φ_3

From Atom Tableau T_φ to ω-Automaton $\mathcal{A}_\varphi$

For temporal formula φ, construct the ω-automaton

$$\mathcal{A}_\varphi : \ \langle \underbrace{N,\ N_0,\ E,}_{\substack{\text{Same as} \\ T_\varphi}}\ \mu,\ \mathcal{F} \rangle$$

where

- Node labeling μ:
  For node $n \in N$ labeled by atom A in T_φ,

  $$\mu(n) \ = \ state(A).$$

- Acceptance condition $\mathcal{F}$:

  Muller:

  $$\mathcal{F} \ = \ \{\text{SCS } S \mid S \text{ is fulfilling }\}$$

  Street:

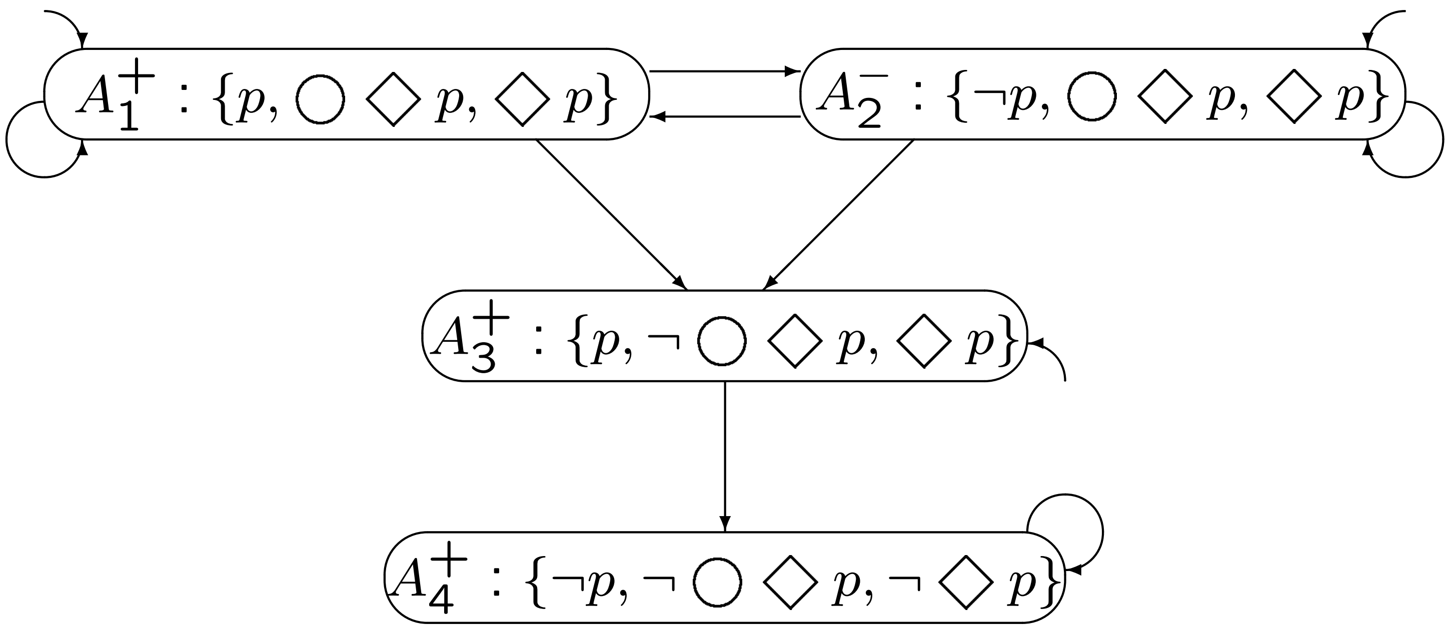
  $$\mathcal{F} \ = \ \{(P_\psi, R_\psi) \mid \psi \in \Phi_\varphi \text{ promises } r\},$$

  where

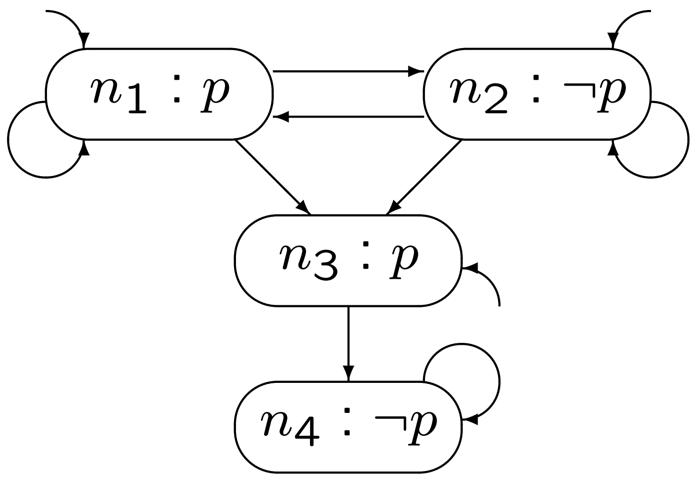
  $$\begin{aligned} P_\psi \ &= \ \{ \ A \mid \neg\psi \in A \ \} \\ R_\psi \ &= \ \{ \ A \mid r \in A \ \} \end{aligned}$$

<u>`Example`</u>: $\varphi: \ \Diamond p$

Tableau T_φ:



Example: $\mathcal{A}_{\Diamond p}$ from $T_{\Diamond p}$



$$
\begin{aligned}
\mathcal{F}_M &= \{\{n_1\}, \{n_1, n_2\}, \{n_4\}\} \\
\mathcal{F}_S &= \{(P_{\Diamond p}, R_{\Diamond p})\} \\
&= \{(\{n_4\}, \{n_1, n_3\})\} \\
&\approx \{(\{n_4\}, \{n_1\})\}
\end{aligned}
$$

since no path can visit n_3 infinitely often

Abstraction

Abstraction = a method to verify infinite-state systems.

Idea:

$$
\begin{array}{rcl}
 & \text{abstraction} & \\
 & \downarrow & \\
\text{Program } P & \longrightarrow & \text{Abstract program } P^A \\
\text{(infinite state)} & & \text{(finite state)} \\
 & & \\
\text{Property } \varphi & \longrightarrow & \text{Abstract property } \varphi^A \\
 & & \\
P \models \varphi? & \longrightarrow & P^A \models \varphi^A \\
 & & \downarrow \\
 & & \text{model checking}
\end{array}
$$

We want to ensure that
if $P^A \models \varphi^A$ then $P \models \varphi$.

Abstraction (Cont'd)

How do we obtain such an abstraction function?

- 1) Abstract the domain to a finite-state one (*data abstraction*):
  For variables $\vec{x}$ ranging over domain D, find an abstract domain D^A and an abstraction function $\alpha : D \rightarrow D^A$.

- 2) From the data abstraction it is possible to compute an abstraction for the program and for the property such that
  if $P^A \models \varphi^A$ then $P \models \varphi$.

Example: Abstracting Bakery

Program MUX-BAK (infinite-state program)

$$
P_1 :: \left[\begin{array}{l}
\textbf{loop forever do} \\
\left[\begin{array}{l}
\ell_0 : \textbf{noncritical} \\
\ell_1 : y_1 := y_2 + 1 \\
\ell_2 : \textbf{await } y_2 = 0 \lor y_1 \leq y_2 \\
\ell_3 : \textbf{critical} \\
\ell_4 : y_1 := 0
\end{array}\right]
\end{array}\right]
$$

$\|$

$$
P_2 :: \left[\begin{array}{l}
\textbf{loop forever do} \\
\left[\begin{array}{l}
m_0 : \textbf{noncritical} \\
m_1 : y_2 := y_1 + 1 \\
m_2 : \textbf{await } y_1 = 0 \lor y_2 < y_1 \\
m_3 : \textbf{critical} \\
m_4 : y_2 := 0
\end{array}\right]
\end{array}\right]
$$

Abstract domain: the boolean algebra over

$B = \{b_1, b_2, b_3 : \textbf{boolean}\}$,

with $b_1 :\ y_1 = 0$

$b_2 :\ y_2 = 0$

$b_3 :\ y_1 \leq y_2$

Example: Abstracting Bakery (Cont'd)

Program MUX-BAK-ABSTR (finite-state program)

$$
\begin{array}{l}
P_1 :: \left[\begin{array}{l}
\textbf{loop forever do} \\
\left[\begin{array}{l}
\ell_0 : \textbf{noncritical} \\
\ell_1 : (b_1, b_3) := (\mathit{false}, \mathit{false}) \\
\ell_2 : \textbf{await } b_2 \vee b_3 \\
\ell_3 : \textbf{critical} \\
\ell_4 : (b_1, b_3) := (\mathit{true}, \mathit{true})
\end{array}\right]
\end{array}\right] \\
\| \\
P_2 :: \left[\begin{array}{l}
\textbf{loop forever do} \\
\left[\begin{array}{l}
m_0 : \textbf{noncritical} \\
m_1 : (b_2, b_3) := (\mathit{false}, \mathit{true}) \\
m_2 : \textbf{await } b_1 \vee \neg b_3 \\
m_3 : \textbf{critical} \\
m_4 : (b_2, b_3) := (\mathit{true}, b_1)
\end{array}\right]
\end{array}\right]
\end{array}
$$

This program can now be checked for mutual exclusion, bounded overtaking, response.

Show MUX-BAK-ABSTR $\models \Box \neg(at\_\ell_3 \wedge at\_m_3)$. Then it follows that MUX-BAK $\models \Box \neg(at\_\ell_3 \wedge at\_m_3)$.