Linear-Time Properties

Lecture #5 of Model Checking

Joost-Pieter Katoen

Lehrstuhl 2: Software Modeling and Verification

E-mail: `katoen@cs.rwth-aachen.de`

April 17, 2007

Overview Lecture #5

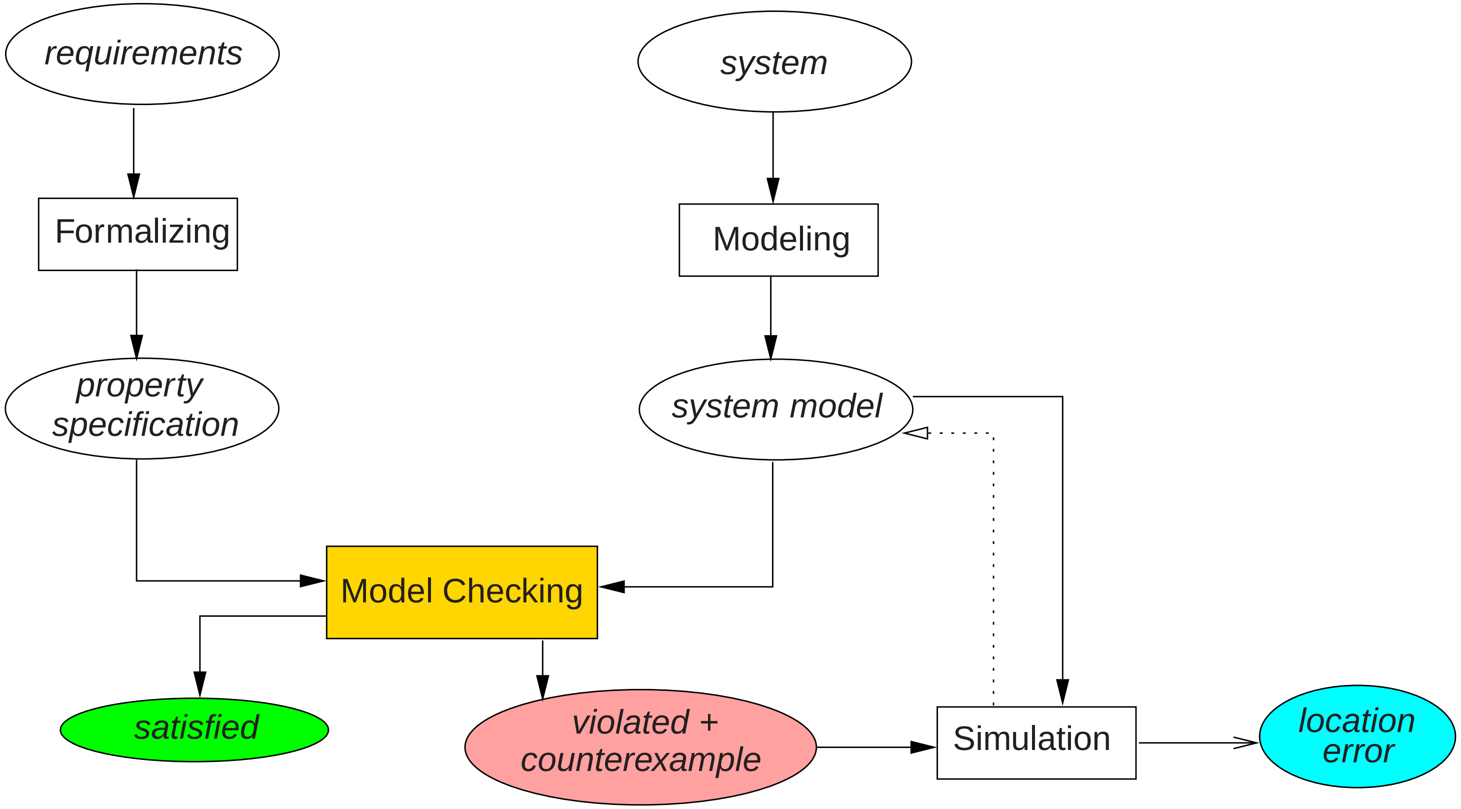
- Paths and traces
- Linear-time (LT) properties
- Trace equivalence and LT properties
- Safety properties and invariants

Summary of Chapter 2

- Transition systems are fundamental for modeling software and hardware
- Interleaving = execution of independent concurrent processes by nondeterminism
- For shared variable communication use composition on program graphs
- Handshaking on a set H of actions amounts to
  - executing actions $\notin H$ autonomously and those in H simultaneously
- Channel systems = program graphs + first-in first-out communication channels
  - handshaking for channels of capacity 0
  - asynchronous message passing when capacity exceeds 0
  - semantical model of Promela
- Size of transition systems grows exponentially
  - in the number of concurrent components and the number of variables

what about properties of transition systems?

Recall model checking



Recall executions

- A *finite execution fragment* ϱ of *TS* is an alternating sequence of states and actions ending with a state:

$$\varrho \;=\; s_0\,\alpha_1\,s_1\,\alpha_2\,\ldots\alpha_n\,s_n \text{ such that } s_i \xrightarrow{\alpha_{i+1}} s_{i+1} \text{ for all } 0 \leqslant i < n.$$

- An *infinite execution fragment* ρ of *TS* is an infinite, alternating sequence of states and actions:

$$\rho \;=\; s_0\,\alpha_1\,s_1\,\alpha_2\,s_2\,\alpha_3\ldots \text{ such that } s_i \xrightarrow{\alpha_{i+1}} s_{i+1} \text{ for all } 0 \leqslant i.$$

- An *execution* of *TS* is an initial, maximal execution fragment
  - a *maximal* execution fragment is either finite ending in a terminal state, or infinite
  - an execution fragment is *initial* if $s_0 \in I$

State graph

- The *state graph* of *TS*, notation $G(\mathit{TS})$, is the digraph (V, E)

  with vertices $V = S$ and edges $E = \{(s, s') \in S \times S \mid s' \in \mathit{Post}(s)\}$

  $\Rightarrow$ omit all state and transition labels in *TS* and ignore being initial

- $\mathit{Post}^*(s)$ is the set of states reachable $G(\mathit{TS})$ from s

$$\mathit{Post}^*(C) = \bigcup_{s \in C} \mathit{Post}^*(s) \quad \text{for } C \subseteq S$$

- The notations $\mathit{Pre}^*(s)$ and $\mathit{Pre}^*(C)$ have analogous meaning

- The set of reachable states: $\mathit{Reach}(\mathit{TS}) = \mathit{Post}^*(I)$

Path fragments

- A path fragment is an execution fragment without actions

- A *finite path fragment* $\widehat{\pi}$ of *TS* is a state sequence:

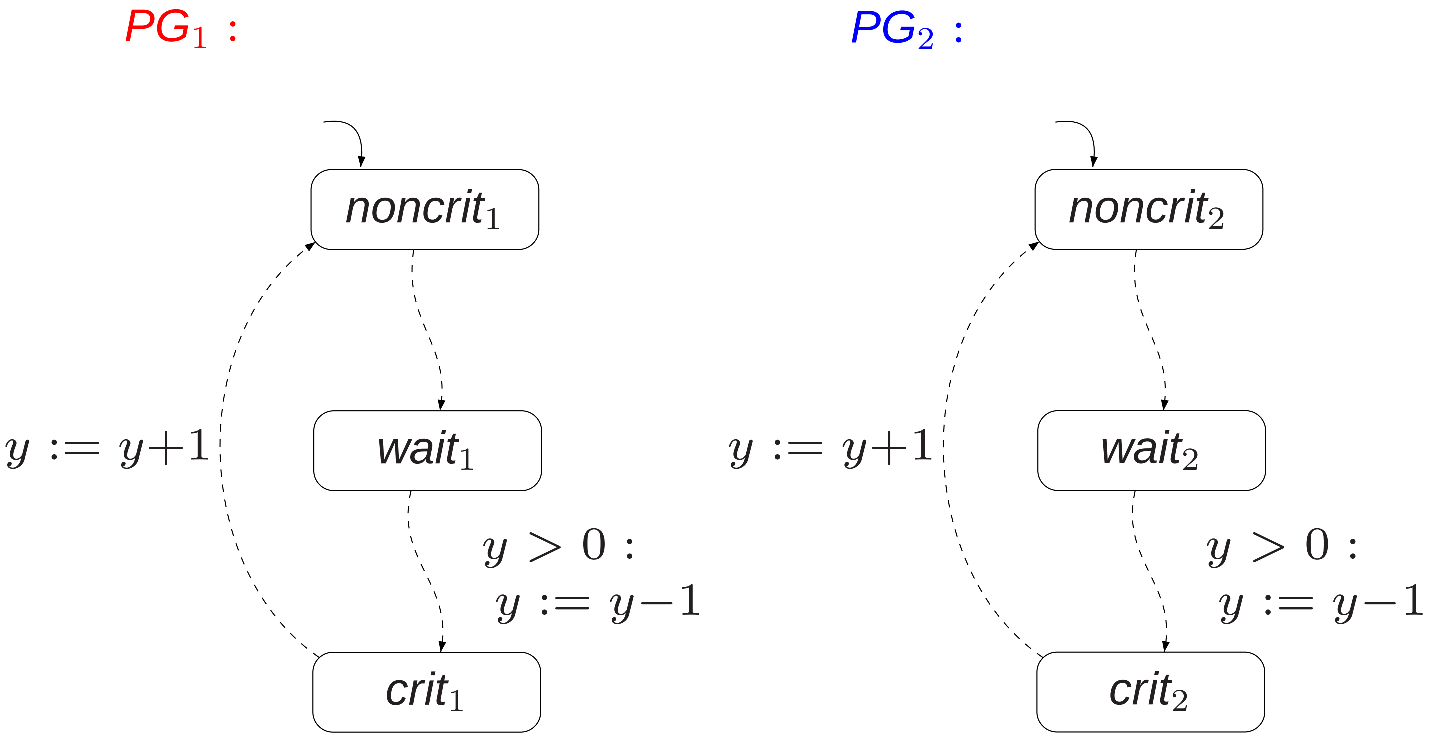
$$\widehat{\pi} \;=\; s_0\, s_1 \ldots s_n \quad \text{such that} \quad s_{i+1} \in \mathit{Post}(s_i) \text{ for all } 0 \leqslant i < n \text{ where } n \geqslant 0$$

- An *infinite path fragment* π of *TS* is an infinite state sequence:

$$\pi \;=\; s_0\, s_1\, s_2 \ldots \quad \text{such that } s_{i+1} \in \mathit{Post}(s_i) \text{ for all } i \geqslant 0$$

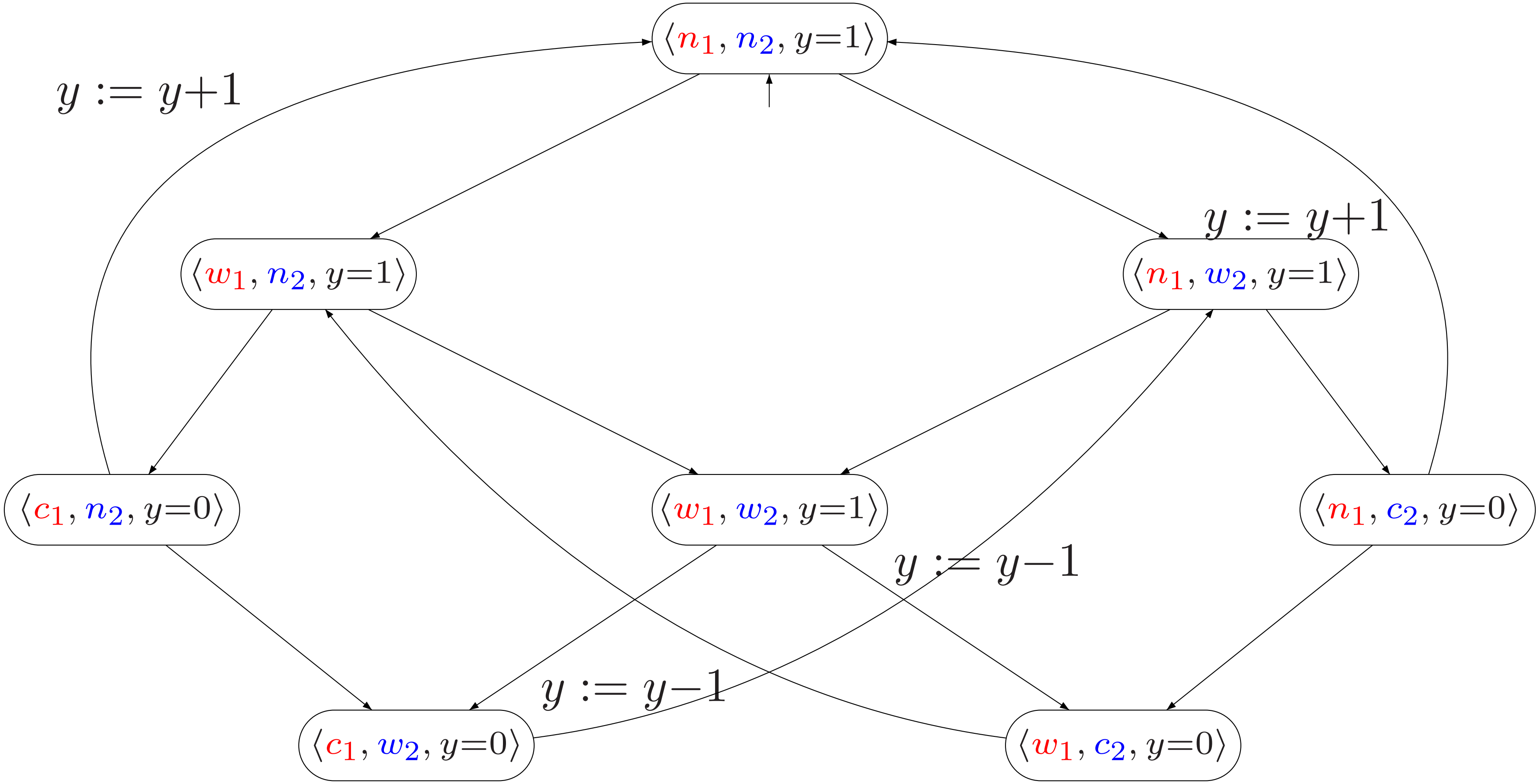
- A *path* of *TS* is an initial, maximal path fragment
  - a *maximal* path fragment is either finite ending in a terminal state, or infinite
  - a path fragment is *initial* if $s_0 \in I$
  - *Paths*(s) is the set of maximal path fragments π with *first*$(\pi) = s$

Semaphore-based mutual exclusion



$y{=}0$ means “lock is currently possessed”; $y{=}1$ means “lock is free”

Transition system $TS(PG_1 \,|||\, PG_2)$


$\langle n_1, n_2, y{=}1\rangle$
$y := y{+}1$
$\langle w_1, n_2, y{=}1\rangle$
$y := y{+}1$
$\langle n_1, w_2, y{=}1\rangle$
$\langle c_1, n_2, y{=}0\rangle$
$\langle w_1, w_2, y{=}1\rangle$
$\langle n_1, c_2, y{=}0\rangle$
$y := y{-}1$
$y := y{-}1$
$\langle c_1, w_2, y{=}0\rangle$
$\langle w_1, c_2, y{=}0\rangle$

Example paths

Traces

- Actions are mainly used to model the (possibility of) interaction
  - synchronous or asynchronous communication
- Here, focus on the states that are visited during executions
  - the states themselves are not “observable”, but just their atomic propositions
- Consider sequences of the form $L(s_0)\, L(s_1)\, L(s_2) \ldots$
  - just register the (set of) atomic propositions that are valid along the execution
  - instead of execution $s_0 \xrightarrow{\alpha_0} s_1 \xrightarrow{\alpha_1} s_2 \ldots$

  $\Rightarrow$ this is called a *trace*
- For a transition system without terminal states:
  - traces are infinite words over the alphabet 2^{AP}, i.e., they are in $\left(2^{AP}\right)^\omega$

Traces

- Let transition system $\mathit{TS} = (S, \mathit{Act}, \rightarrow, I, \mathit{AP}, L)$ without terminal states
  - all maximal paths (and excutions) are infinite
- The *trace* of path fragment $\pi = s_0\, s_1 \ldots$ is $\mathit{trace}(\pi) = L(s_0)\, L(s_1) \ldots$
  - the trace of $\widehat{\pi} = s_0\, s_1 \ldots s_n$ is $\mathit{trace}(\widehat{\pi}) = L(s_0)\, L(s_1) \ldots L(s_n)$
- The set of traces of a set Π of paths: $\mathit{trace}(\Pi) = \{\, \mathit{trace}(\pi) \mid \pi \in \Pi \,\}$
- $\mathit{Traces}(s) = \mathit{trace}(\mathit{Paths}(s))$ $\mathit{Traces}(\mathit{TS}) = \bigcup_{s \in I} \mathit{Traces}(s)$
- $\mathit{Traces}_{\mathit{fin}}(s) = \mathit{trace}(\mathit{Paths}_{\mathit{fin}}(s))$ $\mathit{Traces}_{\mathit{fin}}(\mathit{TS}) = \bigcup_{s \in I} \mathit{Traces}_{\mathit{fin}}(s)$

Example traces

Let $AP = \{ \mathit{crit}_1, \mathit{crit}_2 \}$

Example path:

$$\begin{aligned} \pi \quad = \quad & \langle n_1, n_2, y=1 \rangle \rightarrow \langle w_1, n_2, y=1 \rangle \rightarrow \langle c_1, n_2, y=0 \rangle \rightarrow \\ & \langle n_1, n_2, y=1 \rangle \rightarrow \langle n_1, w_2, y=1 \rangle \rightarrow \langle n_1, c_2, y=0 \rangle \rightarrow \ldots \end{aligned}$$

The trace of this path is the infinite word:

$$\mathit{trace}(\pi) \; = \; \varnothing \, \varnothing \, \{ \mathit{crit}_1 \} \, \varnothing \, \varnothing \, \{ \mathit{crit}_2 \} \, \varnothing \, \varnothing \, \{ \mathit{crit}_1 \} \, \varnothing \, \varnothing \, \{ \mathit{crit}_2 \} \ldots$$

The trace of the finite path fragment:

$$\begin{aligned} \widehat{\pi} \quad = \quad & \langle n_1, n_2, y=1 \rangle \rightarrow \langle w_1, n_2, y=1 \rangle \rightarrow \langle w_1, w_2, y=1 \rangle \rightarrow \\ & \langle w_1, c_2, y=0 \rangle \rightarrow \langle w_1, n_2, y=1 \rangle \rightarrow \langle c_1, n_2, y=0 \rangle \end{aligned}$$

is:

$$\mathit{trace}(\widehat{\pi}) \; = \; \varnothing \, \varnothing \, \varnothing \, \{ \mathit{crit}_2 \} \, \varnothing \, \{ \mathit{crit}_1 \}$$

Linear-time properties

- Linear-time properties specify the traces that a TS must exhibit
  - LT-property specifies the admissible behaviour of system under consideration

later, a logic will be introduced for specifying LT properties

- A *linear-time property* (LT property) over *AP* is a subset of $\left(2^{AP}\right)^{\omega}$
  - finite words are not needed, as it is assumed that there are *no terminal states*
- *TS* (over *AP*) *satisfies* LT property P (over *AP*):

$$\mathit{TS} \models P \quad \text{if and only if} \quad \mathit{Traces}(\mathit{TS}) \ \subseteq \ P$$

  - *TS* satisfies the LT property P if all its “observable” behaviors are admissible
  - state $s \in S$ satisfies P, notation $s \models P$, whenever $\mathit{Traces}(s) \ \subseteq \ P$

How to specify mutual exclusion?

“Always at most one process is in its critical section”

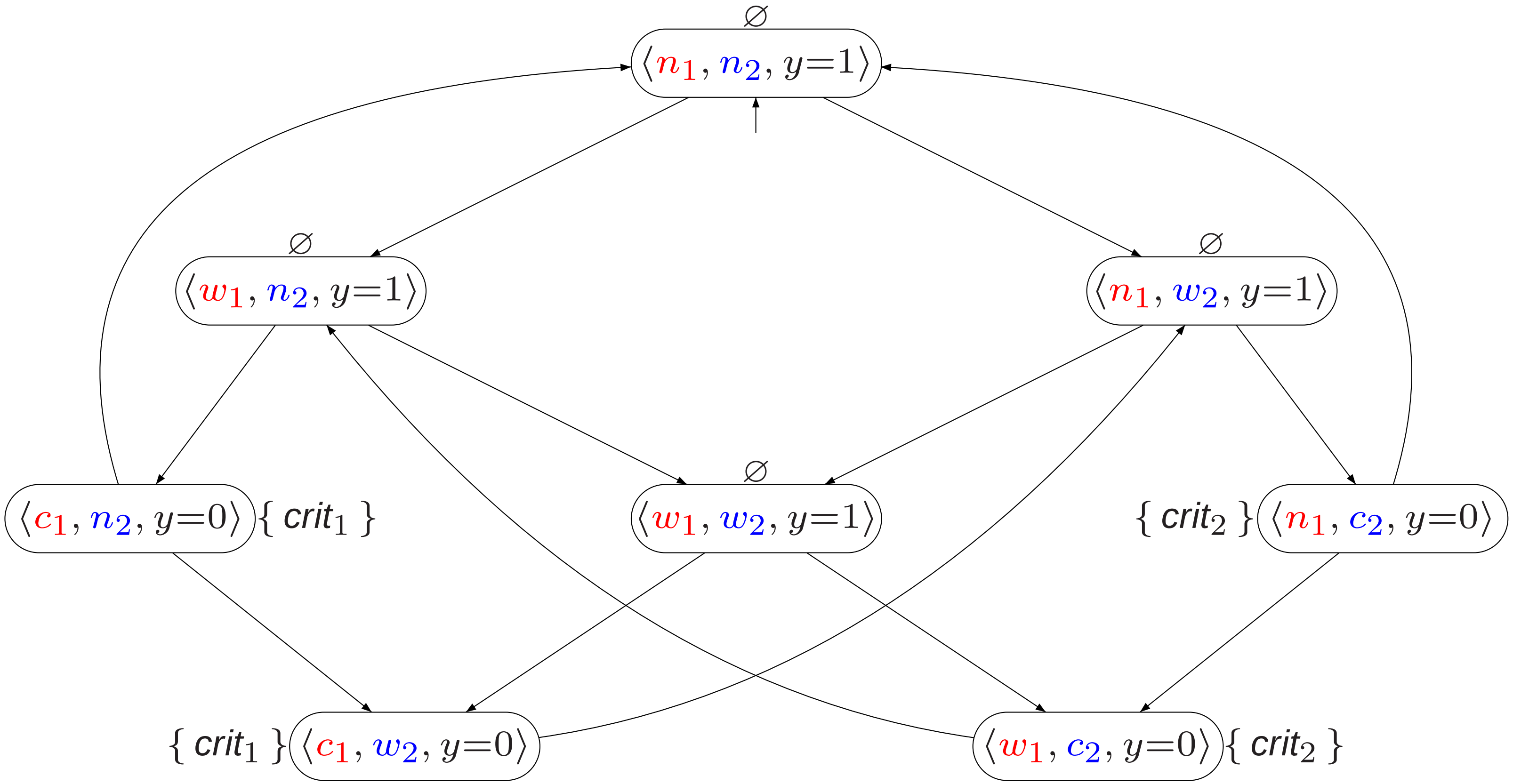
- Let $\mathit{AP} = \{\, \mathit{crit}_1, \mathit{crit}_2 \,\}$
  - other atomic propositions are not of any relevance for this property
- Formalization as LT property

$$P_{mutex} \;=\; \text{set of infinite words } A_0\, A_1\, A_2 \ldots \text{ with } \{\, \mathit{crit}_1, \mathit{crit}_2 \,\} \not\subseteq A_i \text{ for all } 0 \leqslant i$$

- Contained in P_{mutex} are e.g., the infinite words:
  - $(\{\, \mathit{crit}_1 \,\}\, \{\, \mathit{crit}_2 \,\})^{\omega}$ and $\{\, \mathit{crit}_1 \,\}\, \{\, \mathit{crit}_1 \,\}\, \{\, \mathit{crit}_1 \,\} \ldots$ and $\varnothing\, \varnothing\, \varnothing \ldots$
  - but not $\{\, \mathit{crit}_1 \,\}\, \varnothing\, \{\, \mathit{crit}_1, \mathit{crit}_2 \,\} \ldots$ or $\varnothing\, \{\, \mathit{crit}_1 \,\}, \varnothing\, \varnothing\, \{\, \mathit{crit}_1, \mathit{crit}_2 \,\} \varnothing \ldots$

Does the semaphore-based algorithm satisfy P_{mutex}?

Does the semaphore-based algorithm satisfy P_{mutex}?


$\varnothing$
$\langle n_1, n_2, y{=}1\rangle$
$\varnothing$
$\langle w_1, n_2, y{=}1\rangle$
$\varnothing$
$\langle n_1, w_2, y{=}1\rangle$
$\langle c_1, n_2, y{=}0\rangle$ $\{ \mathit{crit}_1 \}$
$\varnothing$
$\langle w_1, w_2, y{=}1\rangle$
$\{ \mathit{crit}_2 \}$ $\langle n_1, c_2, y{=}0\rangle$
$\{ \mathit{crit}_1 \}$ $\langle c_1, w_2, y{=}0\rangle$
$\langle w_1, c_2, y{=}0\rangle$ $\{ \mathit{crit}_2 \}$

Yes as there is no reachable state labeled with $\{ \mathit{crit}_1, \mathit{crit}_2 \}$

How to specify starvation freedom?

"A process that wants to enter the critical section is eventually able to do so‘"

- Let $AP = \{ \mathit{wait}_1, \mathit{crit}_1, \mathit{wait}_2, \mathit{crit}_2 \}$
- Formalization as LT-property

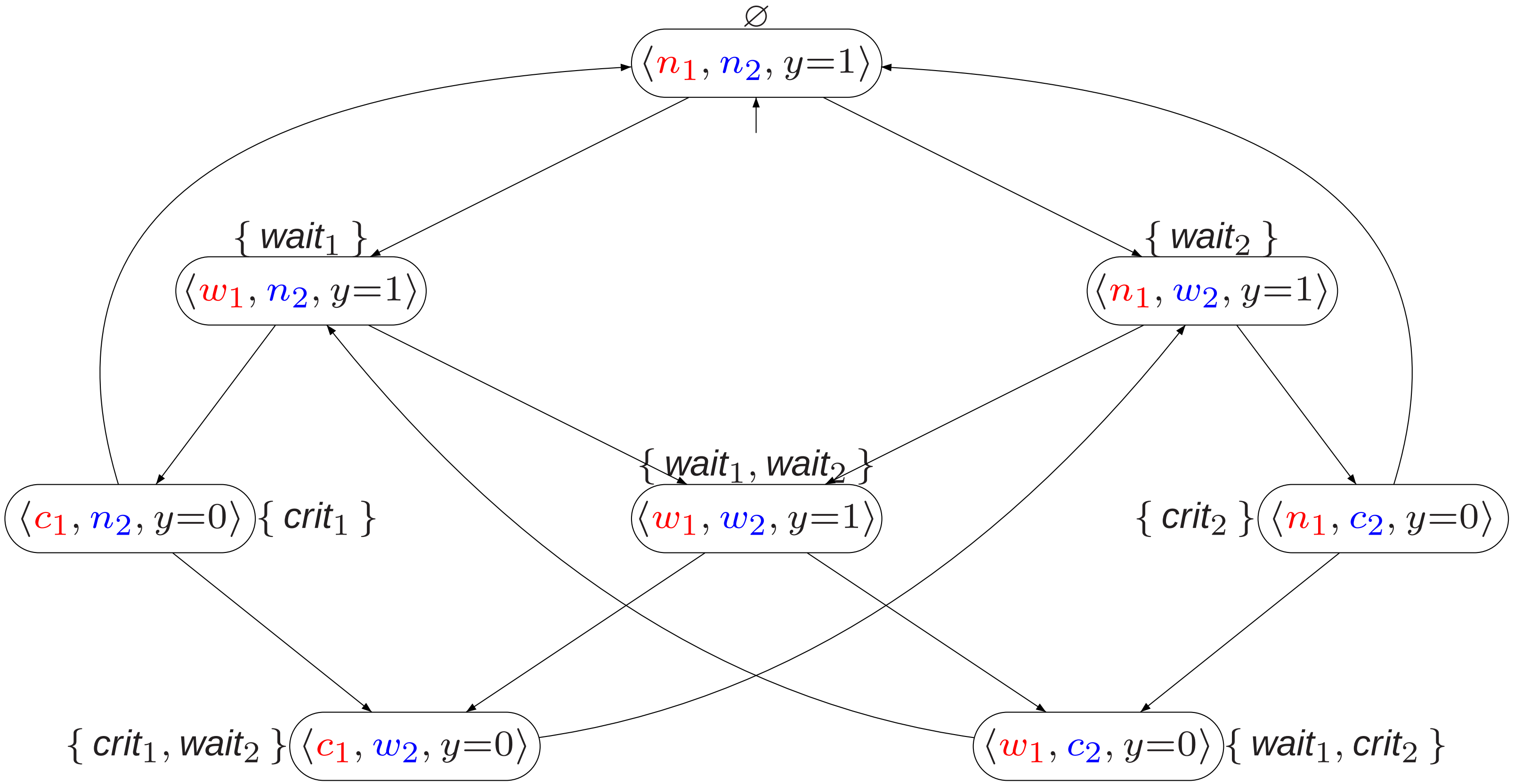
$P_{nostarve}$ = set of infinite words $A_0\, A_1\, A_2 \ldots$ such that:

$$\Big(\overset{\infty}{\exists}\, j.\ \mathit{wait}_i \in A_j \Big) \;\Rightarrow\; \Big(\overset{\infty}{\exists}\, j.\ \mathit{crit}_i \in A_j \Big) \qquad \text{for each } i \in \{ 1, 2 \}$$

there exist infinitely many: $\Big(\overset{\infty}{\exists}\, j.\, \mathit{wait}_i \in A_j \Big) \equiv (\forall k \geqslant 0.\, \exists j > k.\, \mathit{wait}_i \in A_j)$

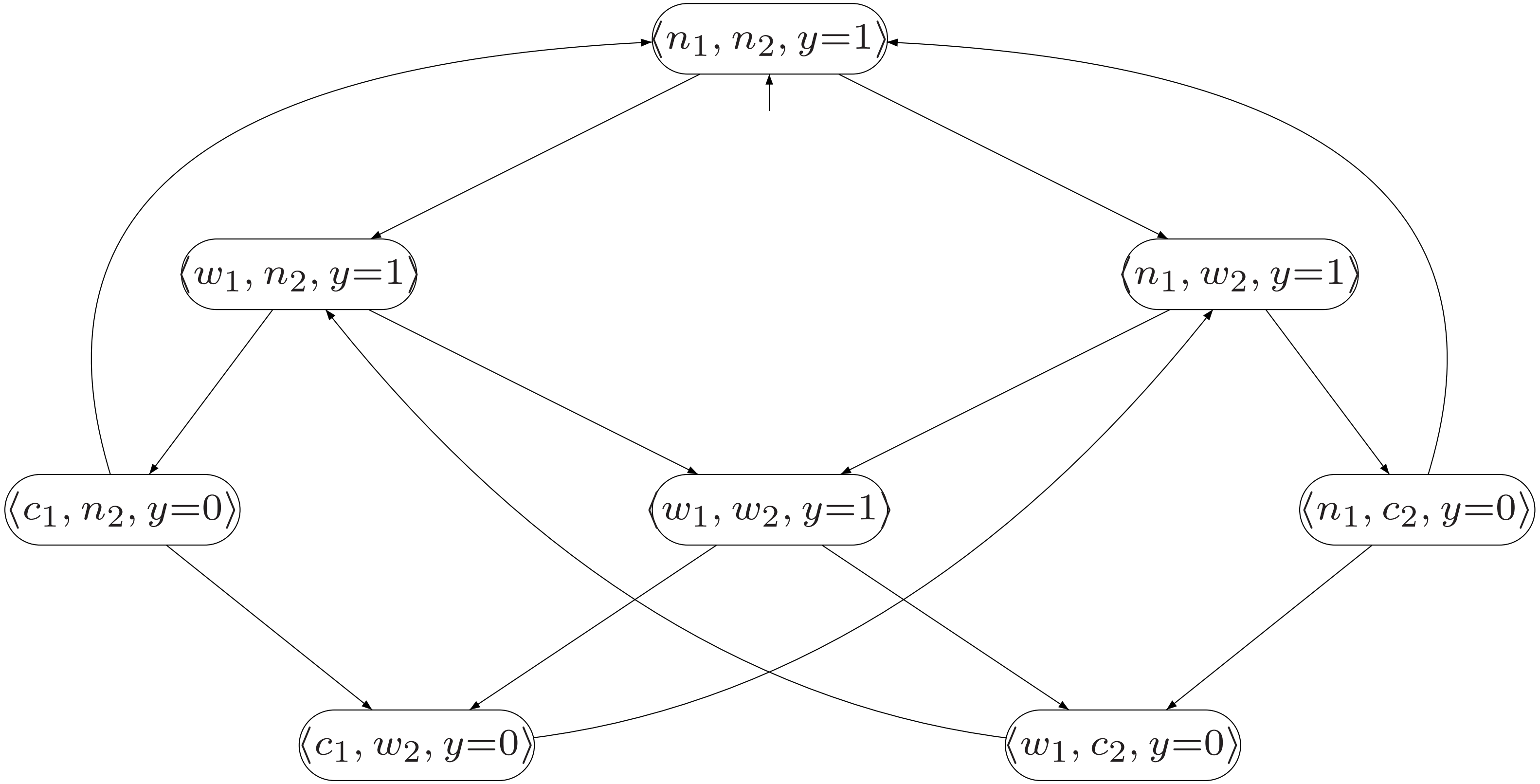
Does the semaphore-based algorithm satisfy $P_{nostarve}$?

Does the semaphore-based algorithm satisfy $P_{nostarve}$?


$\varnothing$
$\langle n_1, n_2, y{=}1 \rangle$
$\{ wait_1 \}$
$\langle w_1, n_2, y{=}1 \rangle$
$\{ wait_2 \}$
$\langle n_1, w_2, y{=}1 \rangle$
$\langle c_1, n_2, y{=}0 \rangle$
$\{ crit_1 \}$
$\{ wait_1, wait_2 \}$
$\langle w_1, w_2, y{=}1 \rangle$
$\{ crit_2 \}$
$\langle n_1, c_2, y{=}0 \rangle$
$\{ crit_1, wait_2 \}$
$\langle c_1, w_2, y{=}0 \rangle$
$\langle w_1, c_2, y{=}0 \rangle$
$\{ wait_1, crit_2 \}$

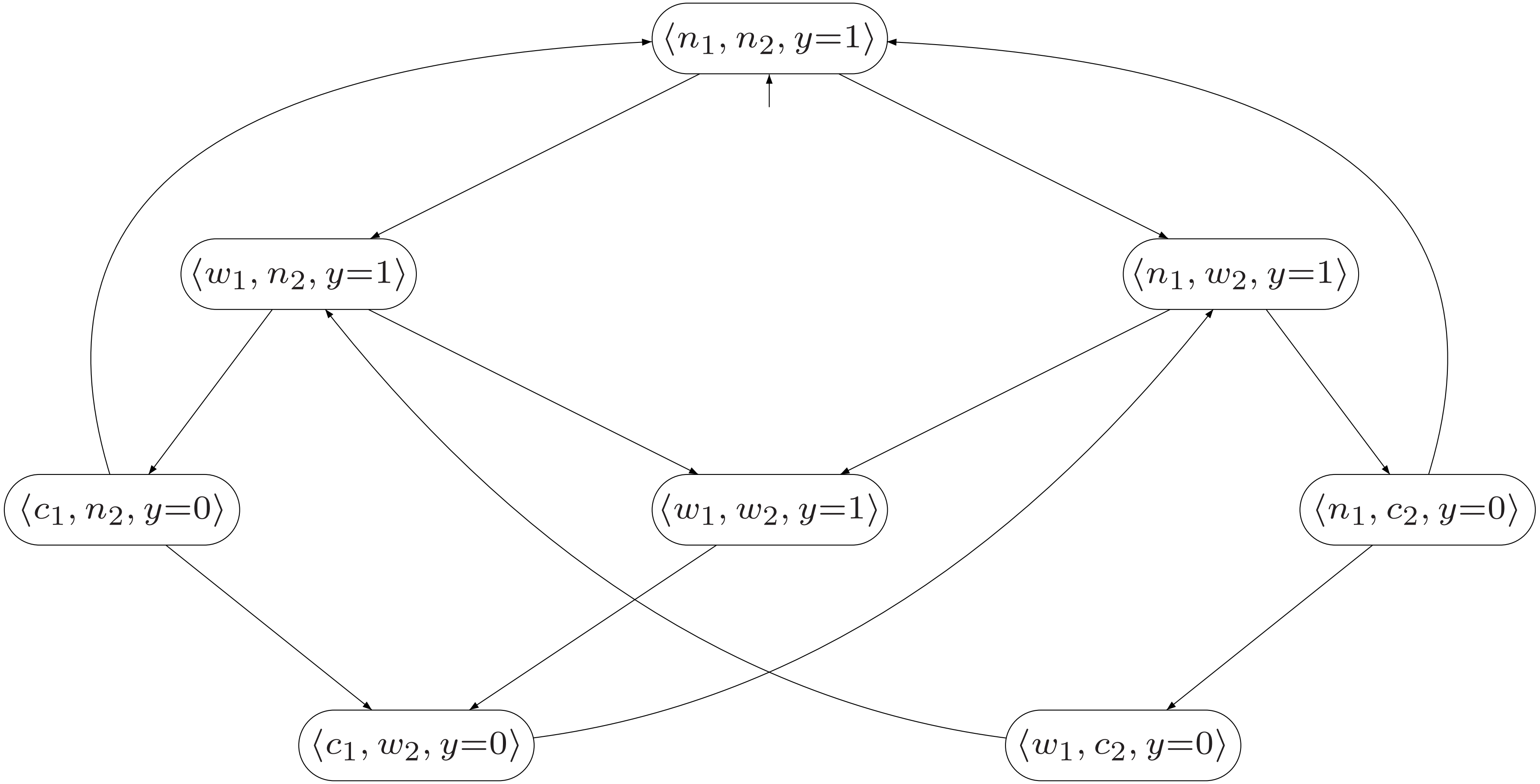
No. Trace $\varnothing \; (\{ \mathit{wait}_2 \} \, \{ \mathit{wait}_1, \mathit{wait}_2 \} \, \{ \mathit{crit}_1, \mathit{wait}_2 \})^\omega \in \mathit{Traces}(\mathit{TS})$, but $\notin P_{nostarve}$

Mutual exclusion algorithm revisited


$\langle n_1, n_2, y{=}1\rangle$
$\langle w_1, n_2, y{=}1\rangle$
$\langle n_1, w_2, y{=}1\rangle$
$\langle c_1, n_2, y{=}0\rangle$
$\langle w_1, w_2, y{=}1\rangle$
$\langle n_1, c_2, y{=}0\rangle$
$\langle c_1, w_2, y{=}0\rangle$
$\langle w_1, c_2, y{=}0\rangle$

this algorithm satisfies P_{mutex}

Refining mutual exclusion algorithm


$\langle n_1, n_2, y{=}1\rangle$
$\langle w_1, n_2, y{=}1\rangle$
$\langle n_1, w_2, y{=}1\rangle$
$\langle c_1, n_2, y{=}0\rangle$
$\langle w_1, w_2, y{=}1\rangle$
$\langle n_1, c_2, y{=}0\rangle$
$\langle c_1, w_2, y{=}0\rangle$
$\langle w_1, c_2, y{=}0\rangle$

this variant algorithm with an omitted edge also satisfies P_{mutex}

Trace equivalence and LT properties

For *TS* and *TS*′ be transition systems (over *AP*) without terminal states:

$$\mathit{Traces}(\mathit{TS}) \subseteq \mathit{Traces}(\mathit{TS}')$$

if and only if

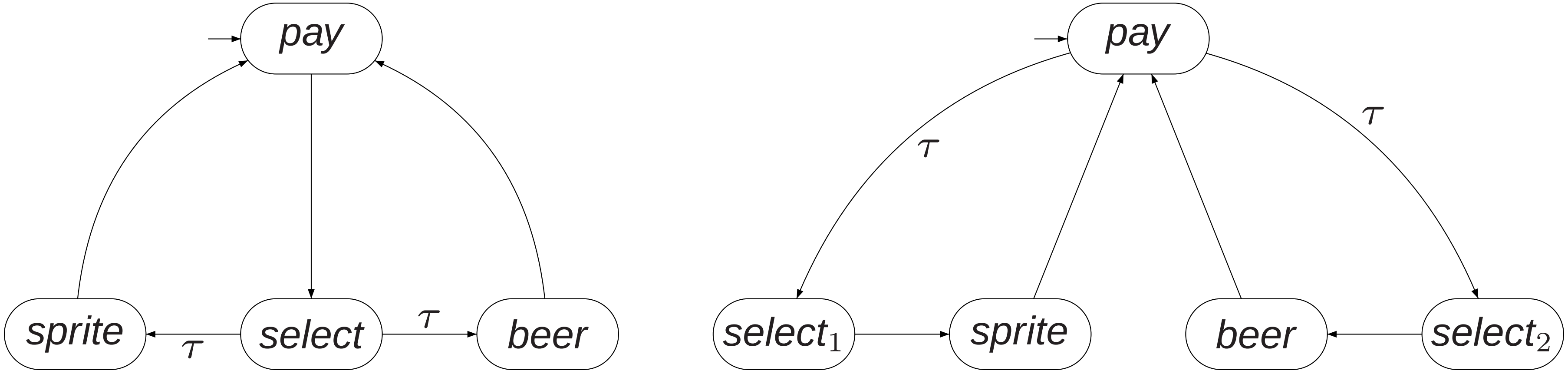
for any LT property P: $\mathit{TS}' \models P$ implies $\mathit{TS} \models P$

$$\mathit{Traces}(\mathit{TS}) = \mathit{Traces}(\mathit{TS}')$$

if and only if

TS and *TS*′ satisfy the same LT properties

Two beverage vending machines



$$AP = \{ pay, sprite, beer \}$$

there is no LT-property that can distinguish between these machines

Invariants

- Safety properties $\approx$ “nothing bad should happen” [Lamport 1977]
- Typical safety property: mutual exclusion property
  - the bad thing (having > 1 process in the critical section) never occurs
- Another typical safety property is deadlock freedom

$\Rightarrow$ These properties are in fact invariants

- An invariant is an LT property
  - that is given by a condition Φ for the states
  - and requires that Φ holds for all reachable states
  - e.g., for mutex property $\Phi \equiv \neg \mathit{crit}_1 \ \vee \ \neg \mathit{crit}_2$

Invariants

- An LT property P_{inv} over *AP* is an *invariant* if there is a propositional logic formula Φ over *AP* such that:

$$P_{inv} = \left\{ A_0 A_1 A_2 \ldots \in \left(2^{AP}\right)^{\omega} \mid \forall j \geqslant 0.\, A_j \models \Phi \right\}$$

  - Φ is called an *invariant condition* of P_{inv}

- Note that

$$\begin{array}{lll} TS \models P_{inv} & \text{iff} & trace(\pi) \in P_{inv} \text{ for all paths } \pi \text{ in } TS \\ & \text{iff} & L(s) \models \Phi \text{ for all states } s \text{ that belong to a path of } TS \\ & \text{iff} & L(s) \models \Phi \text{ for all states } s \in Reach(TS) \end{array}$$

- Φ has to be fulfilled by all initial states and
  - satisfaction of Φ is invariant under all transitions in the reachable fragment of *TS*

Checking an invariant

- Checking an invariant for the propositional formula Φ

  $=$ check the validity of Φ in every reachable state

  $\Rightarrow$ use a slight modification of standard graph traversal algorithms (DFS and BFS)

  - provided the given transition system *TS* is *finite*

- Perform a forward depth-first search

  - at least one state s is found with $s \not\models \Phi \Rightarrow$ the invariance of Φ is violated

- Alternative: backward search

  - starts with all states where Φ does not hold
  - calculates (by a DFS or BFS) the set $\bigcup_{s \in S, s \not\models \Phi} \mathit{Pre}^*(s)$

A naive invariant checking algorithm

Input: finite transition system *TS* and propositional formula Φ
Output: true if *TS* satisfies the invariant ”always Φ”, otherwise false

```
set of state $R := \varnothing$;                      (* the set of visited states *)
stack of state $U := \varepsilon$;                    (* the empty stack *)
bool $b :=$ true;                                     (* all states in $R$ satisfy $\Phi$ *)
for all $s \in I$ do
   if $s \notin R$ then
      visit($s$)                                      (* perform a dfs for each unvisited initial state *)
   fi
od
return $b$
```

A naive invariant checking algorithm

```
procedure visit (state $s$)
  push$(s, U)$;                                   (* push $s$ on the stack *)
  $R := R \cup \{ s \}$;                          (* mark $s$ as reachable *)
  repeat
    $s' :=$ top$(U)$;
    if Post$(s') \subseteq R$ then
      pop$(U)$;
      $b := b \wedge (s' \models \Phi)$;          (* check validity of $\Phi$ in $s'$ *)
    else
      let $s'' \in$ Post$(s') \setminus R$
      push$(s'', U)$;
      $R := R \cup \{ s'' \}$;                    (* state $s''$ is a new reachable state *)
    fi
  until $(U = \varepsilon)$
endproc
```

error indication is state refuting Φ

initial path fragment $s_0\, s_1\, s_2 \ldots s_n$ with $s_i \models \Phi$ ($i \neq n$) and $s_n \not\models \Phi$ is more useful

Invariant checking by DFS

Input: finite transition system *TS* and propositional formula Φ
Output: ”yes” if *TS* $\models$ ”always Φ”, otherwise ”no” plus a counterexample

```
set of states $R := \varnothing$;                          (* the set of reachable states *)
stack of states $U := \varepsilon$;                              (* the empty stack *)
bool $b :=$ true;                                  (* all states in $R$ satisfy $\Phi$ *)
while $(I \setminus R \neq \varnothing \ \wedge \ b)$ do
   let $s \in I \setminus R$;             (* choose an arbitrary initial state not in $R$ *)
   visit($s$);                (* perform a DFS for each unvisited initial state *)
od
if $b$ then
   return(”yes”)                                       (* TS $\models$ ”always $\Phi$” *)
else
   return(”no”, reverse($U$))          (* counterexample arises from the stack content *)
fi
```

Invariant checking by DFS

```
procedure visit (state $s$)
  push$(s, U)$;                                   (* push $s$ on the stack *)
  $R := R \cup \{ s \}$;                           (* mark $s$ as reachable *)
  repeat
    $s' :=$ top$(U)$;
    if Post$(s') \subseteq R$ then
      pop$(U)$;
      $b := b \;\wedge\; (s' \models \Phi)$;         (* check validity of $\Phi$ in $s'$ *)
    else
      let $s'' \in$ Post$(s') \setminus R$
      push$(s'', U)$;
      $R := R \cup \{ s'' \}$;                      (* state $s''$ is a new reachable state *)
    fi
  until $((U = \varepsilon) \;\vee\; \neg b)$
endproc
```

Time complexity

- Under the assumption that
  - $s' \in \mathit{Post}(s)$ can be encountered in time $\Theta(|\mathit{Post}(s)|)$

  $\Rightarrow$ this holds for a representation of $\mathit{Post}(s)$ by adjacency lists

- The time complexity for invariant checking is $\mathcal{O}(\, N * (1 + |\Phi|) + M \,)$
  - where N denotes the number of reachable states, and
  - $M = \sum_{s \in S} |\mathit{Post}(s)|$ the number of transitions in the reachable fragment of *TS*

- The adjacency lists are typically given *implicitly*
  - e.g., by a syntactic description of the concurrent processes as program graphs
  - $\mathit{Post}(s)$ is obtained by the rules for the transition relation

Safety properties

- Safety properties may impose requirements on finite path fragments
  - and cannot be verified by considering the reachable states only

- A safety property which is not an invariant:
  - consider a cash dispenser, also known as automated teller machine (ATM)
  - property “money can only be withdrawn once a correct PIN has been provided”

  $\Rightarrow$ not an invariant, since it is not a state property

- But a safety property:
  - any infinite run violating the property has a finite prefix that is “bad”
  - i.e., in which money is withdrawn without issuing a PIN before

Safety properties

- LT property P_{safe} over *AP* is a *safety property* if
  - for all $\sigma \in \left(2^{AP}\right)^{\omega} \setminus P_{safe}$ there exists a finite prefix $\widehat{\sigma}$ of σ such that:

$$P_{safe} \cap \underbrace{\left\{ \sigma' \in \left(2^{AP}\right)^{\omega} \mid \widehat{\sigma} \text{ is a prefix of } \sigma' \right\}}_{\text{all possible extensions of } \widehat{\sigma}} = \varnothing$$

  - any such finite word $\widehat{\sigma}$ is called a bad prefix for P_{safe}

- *Minimal* bad prefix for P_{safe}:
  - is a bad prefix $\widehat{\sigma}$ for P_{safe} for which no proper prefix of $\widehat{\sigma}$ is a bad prefix for P_{safe}

$\Rightarrow$ minimal bad prefixes are bad prefixes of minimal length

Example safety properties

Safety properties and finite traces

For transition system *TS* without terminal states

and safety property P_{safe}:

$\mathit{TS} \models P_{safe}$ if and only if $\mathit{Traces}_{\mathit{fin}}(\mathit{TS}) \cap \mathit{BadPref}(P_{safe}) = \varnothing$

where $\mathit{BadPref}(P_{safe})$ is the set of bad prefixes of P_{safe}