Overview

Introduction

Modelling parallel systems

Linear Time Properties

Regular Properties

Linear Temporal Logic

Computation-Tree Logic

Equivalences and Abstraction

Overview

- Introduction
- Modelling parallel systems
- **Linear Time Properties**
  - state-based and linear time view ⟵
  - definition of linear time properties
  - invariants and safety
  - liveness and fairness
- Regular Properties
- Linear Temporal Logic
- Computation-Tree Logic
- Equivalences and Abstraction

State-based view of TS

transition system $\mathcal{T} = (S, Act, \longrightarrow, S_0, AP, L)$

State-based view of TS

transition system $\mathcal{T} = (S, Act, \longrightarrow, S_0, AP, L)$

Act for modeling interactions/communication

AP, L for specifying properties

State-based view of TS

transition system $\mathcal{T} = (S, Act, \longrightarrow, S_0, AP, L)$

Act	for modeling interactions/communication and specifying fairness assumptions
AP, L	for specifying properties

State-based view of TS

transition system $\mathcal{T} = (S, Act, \longrightarrow, S_0, AP, L)$

$\Big\downarrow$ abstraction from actions

state graph $G_{\mathcal{T}}$

- set of nodes = state space S
- edges = transitions without action label

Act for modeling interactions/communication and specifying fairness assumptions

AP, L for specifying properties

State-based view of TS

transition system $\mathcal{T} = (S, Act, \longrightarrow, S_0, AP, L)$

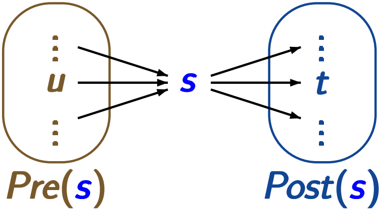
↓ abstraction from actions

state graph $G_{\mathcal{T}}$

- set of nodes = state space S
- edges = transitions without action label

use standard notations for graphs, e.g.,

$$Post(s) = \{t \in S : s \rightarrow t\}$$
$$Pre(s) = \{u \in S : u \rightarrow s\}$$



Execution fragments

execution fragment: sequence of consecutive transitions

$s_0 \xrightarrow{\alpha_0} s_1 \xrightarrow{\alpha_1} \ldots$ infinite or

$s_0 \xrightarrow{\alpha_0} s_1 \xrightarrow{\alpha_1} \ldots \xrightarrow{\alpha_{n-1}} s_n$ finite

Execution and path fragments

execution fragment: sequence of consecutive transitions

$s_0 \xrightarrow{\alpha_0} s_1 \xrightarrow{\alpha_1} \ldots$ infinite or

$s_0 \xrightarrow{\alpha_0} s_1 \xrightarrow{\alpha_1} \ldots \xrightarrow{\alpha_{n-1}} s_n$ finite

path fragment: sequence of states arising from the projection of an execution fragment to the states

Execution and path fragments

execution fragment: sequence of consecutive transitions

$s_0 \xrightarrow{\alpha_0} s_1 \xrightarrow{\alpha_1} \ldots$ infinite or

$s_0 \xrightarrow{\alpha_0} s_1 \xrightarrow{\alpha_1} \ldots \xrightarrow{\alpha_{n-1}} s_n$ finite

path fragment: sequence of states arising from the projection of an execution fragment to the states

$\pi = s_0\, s_1\, s_2 \ldots$ infinite or $\pi = s_0\, s_1 \ldots s_n$ finite

such that $s_{i+1} \in Post(s_i)$ for all $i < |\pi|$

Execution and path fragments

execution fragment: sequence of consecutive transitions

$s_0 \xrightarrow{\alpha_0} s_1 \xrightarrow{\alpha_1} \ldots$ infinite or

$s_0 \xrightarrow{\alpha_0} s_1 \xrightarrow{\alpha_1} \ldots \xrightarrow{\alpha_{n-1}} s_n$ finite

path fragment: sequence of states arising from the projection of an execution fragment to the states

$\pi = s_0\, s_1\, s_2 \ldots$ infinite or $\pi = s_0\, s_1 \ldots s_n$ finite

such that $s_{i+1} \in Post(s_i)$ for all $i < |\pi|$

initial: if $s_0 \in S_0$ = set of initial states

Execution and path fragments

execution fragment: sequence of consecutive transitions

$s_0 \xrightarrow{\alpha_0} s_1 \xrightarrow{\alpha_1} \ldots$ infinite or

$s_0 \xrightarrow{\alpha_0} s_1 \xrightarrow{\alpha_1} \ldots \xrightarrow{\alpha_{n-1}} s_n$ finite

path fragment: sequence of states arising from the projection of an execution fragment to the states

$\pi = s_0\, s_1\, s_2 \ldots$ infinite or $\pi = s_0\, s_1 \ldots s_n$ finite

such that $s_{i+1} \in Post(s_i)$ for all $i < |\pi|$

initial: if $s_0 \in S_0$ = set of initial states

maximal: if infinite or ending in a terminal state

Notations for paths

path fragment: sequence of states

$\pi = s_0\, s_1\, s_2 \ldots$ infinite or $\pi = s_0\, s_1 \ldots s_n$ finite

s.t. $s_{i+1} \in Post(s_i)$ for all $i < |\pi|$

initial: if $s_0 \in S_0$ = set of initial states

maximal: if infinite or ending in terminal state

path of TS $\mathcal{T}$ $\widehat{=}$ initial, maximal path fragment

Notations for paths

path fragment: sequence of states

$\pi = s_0\, s_1\, s_2 \ldots$ infinite or $\pi = s_0\, s_1 \ldots s_n$ finite

s.t. $s_{i+1} \in Post(s_i)$ for all $i < |\pi|$

initial: if $s_0 \in S_0$ = set of initial states

maximal: if infinite or ending in terminal state

path of TS $\mathcal{T}$ $\widehat{=}$ initial, maximal path fragment

path of state s $\widehat{=}$ maximal path fragment starting in state s

Notations for paths

path fragment: sequence of states

$\pi = s_0\, s_1\, s_2 \ldots$ infinite or $\pi = s_0\, s_1 \ldots s_n$ finite

s.t. $s_{i+1} \in Post(s_i)$ for all $i < |\pi|$

initial: if $s_0 \in S_0$ = set of initial states

maximal: if infinite or ending in terminal state

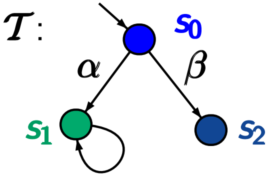
path of TS $\mathcal{T}$ $\widehat{=}$ initial, maximal path fragment

path of state s $\widehat{=}$ maximal path fragment starting in state s

$Paths(\mathcal{T})$ = set of all initial, maximal path fragments

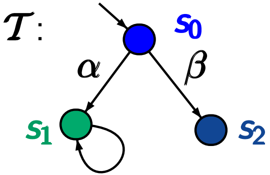
$Paths(s)$ = set of all maximal path fragments starting in state s

Paths of a TS



How many paths are there in $\mathcal{T}$?

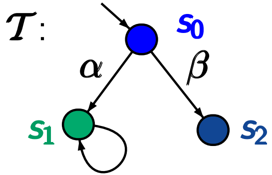
Paths of a TS



How many paths are there in $\mathcal{T}$?

answer: **2**, namely $s_0\, s_1\, s_1\, s_1 \ldots$ and $s_0\, s_2$

Paths of a TS and its states



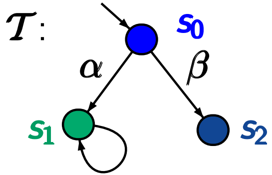
How many paths are there in $\mathcal{T}$?

answer: 2, namely $s_0\, s_1\, s_1\, s_1 \ldots$ and $s_0\, s_2$

$$Paths(s_1) = \text{set of all maximal paths fragments starting in } s_1$$

$$= \{s_1^\omega\} \text{ where } s_1^\omega = s_1\, s_1\, s_1\, s_1\, \ldots$$

Paths of a TS and its states



How many paths are there in $\mathcal{T}$?

answer: 2, namely $s_0\, s_1\, s_1\, s_1 \ldots$ and $s_0\, s_2$

$Paths(s_1)$ = set of all maximal paths fragments starting in s_1

$= \{s_1^\omega\}$ where $s_1^\omega = s_1\, s_1\, s_1\, s_1\, \ldots$

$Paths_{fin}(s_1)$ = set of all finite path fragments starting in s_1

$= \{s_1^n : n \in \mathbb{N}, n \geq 1\}$

Overview

Introduction

Modelling parallel systems

Linear Time Properties

- state-based and linear time view ⟵
- definition of linear time properties
- invariants and safety
- liveness and fairness

Regular Properties

Linear Temporal Logic

Computation-Tree Logic

Equivalences and Abstraction

Overview

- Introduction
- Modelling parallel systems
- **Linear Time Properties**
  - state-based and linear time view ⟵
  - definition of linear time properties
  - invariants and safety
  - liveness and fairness
- Regular Properties
- Linear Temporal Logic
- Computation-Tree Logic
- Equivalences and Abstraction

Linear-time vs branching-time

Linear-time vs branching-time

transition system

$$\mathcal{T} = (S, Act, \rightarrow, S_0, AP, L)$$

Linear-time vs branching-time

LTB2.4-1

transition system

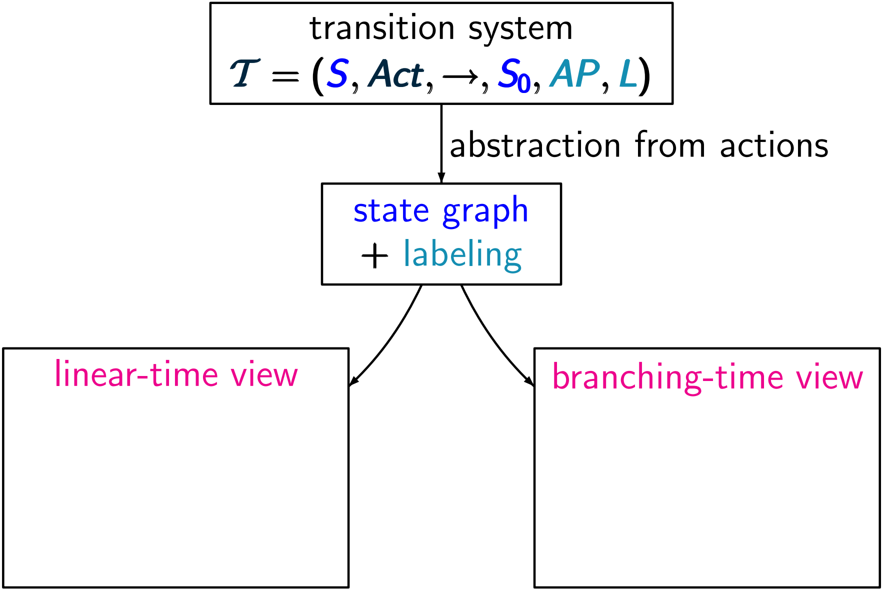
$\mathcal{T} = (S, Act, \rightarrow, S_0, AP, L)$

↓ abstraction from actions

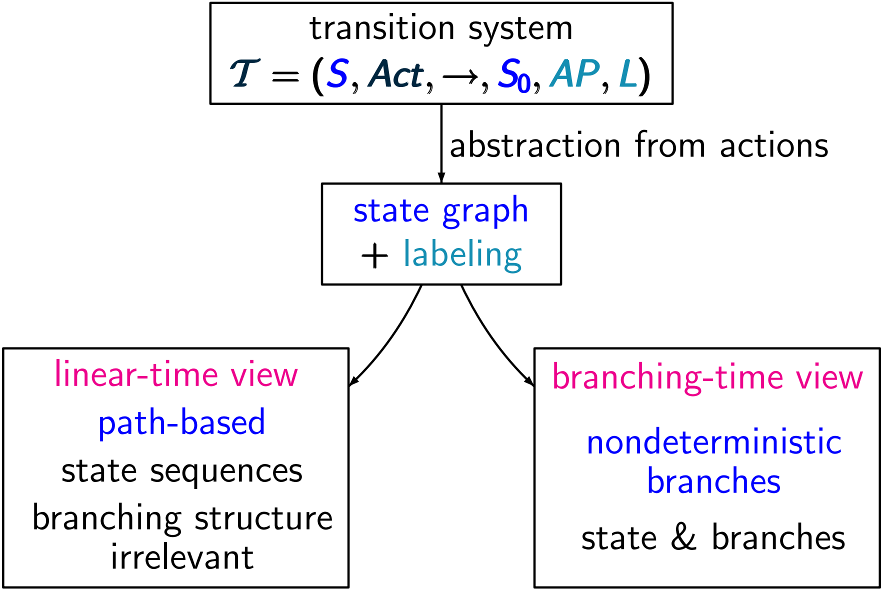
state graph

+ labeling

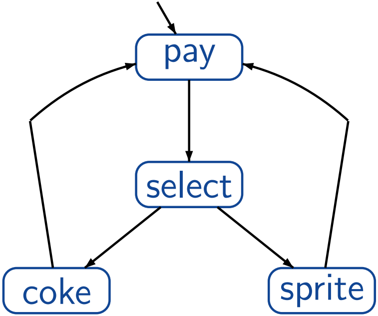
Linear-time vs branching-time



Linear-time vs branching-time

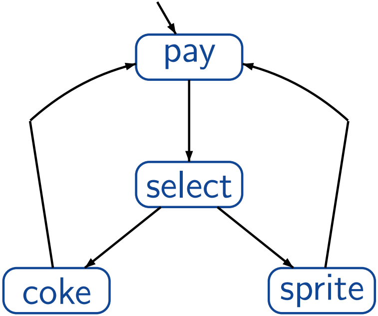


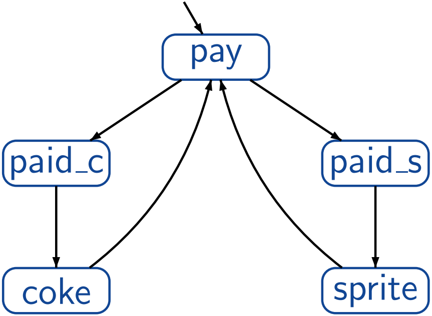
Example: vending machine



vending machine with
1 coin deposit
select drink after
having paid

Example: vending machine

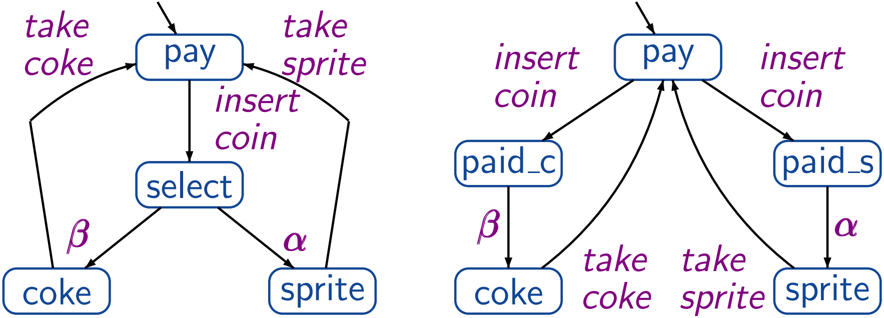




vending machine with
1 coin deposit
select drink after
having paid

vending machine with
2 coin deposits
select drink by inserting
the coin

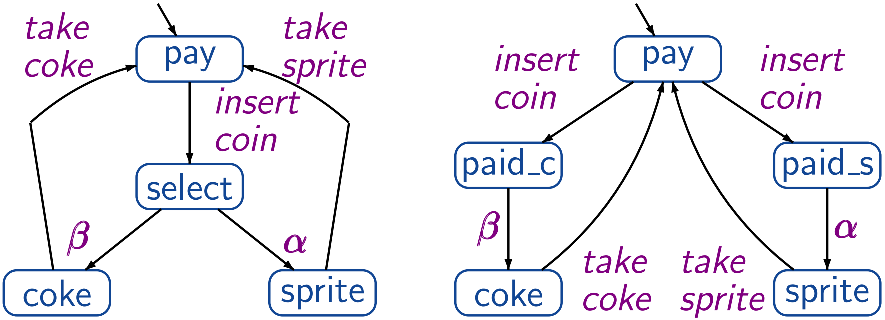
Example: vending machine



vending machine with
1 coin deposit
select drink after having paid

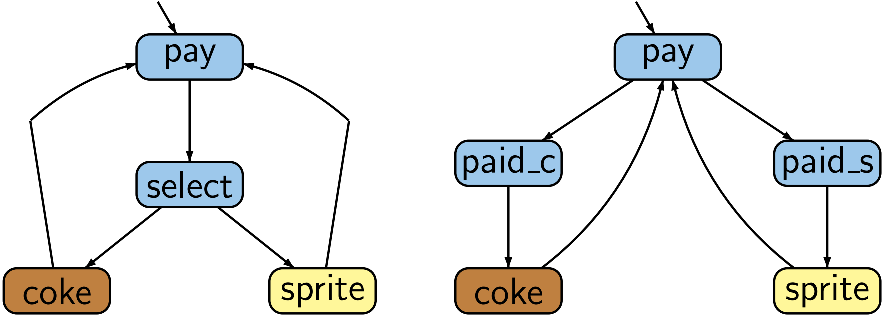
vending machine with
2 coin deposits
select drink by inserting the coin

Example: vending machine



state based view: abstracts from actions and projects onto atomic propositions, e.g. $AP = \{coke, sprite\}$

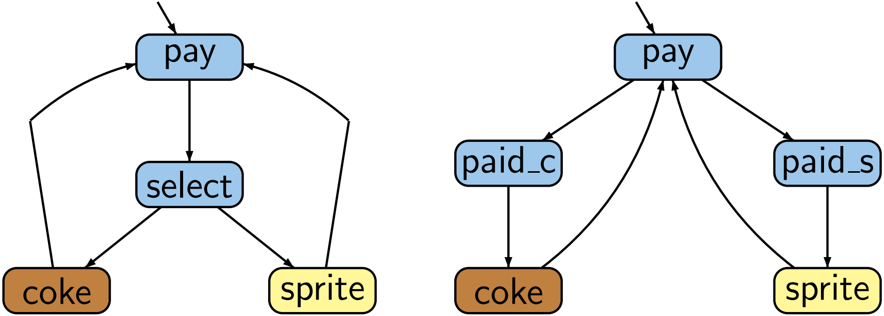
Example: vending machine



state based view: abstracts from actions and projects onto atomic propositions, e.g. $AP = \{coke, sprite\}$

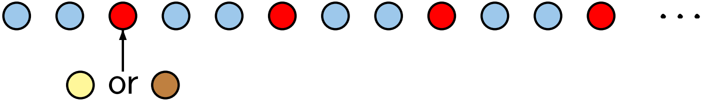
e.g., $L(\text{coke}) = \{coke\}$, $L(\text{pay}) = \varnothing$

Example: vending machine

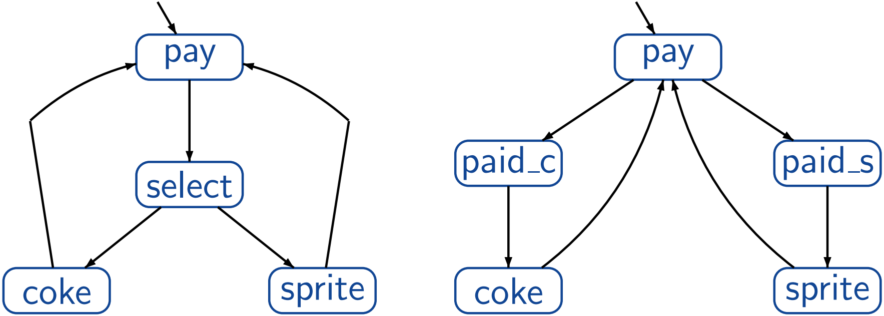


state based view: abstracts from actions and projects onto atomic propositions, e.g. $AP = \{coke, sprite\}$

linear time: all observable behaviors are of the form

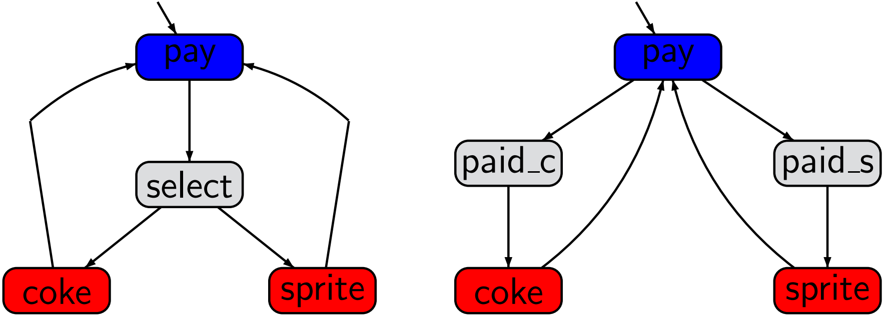


Example: vending machine



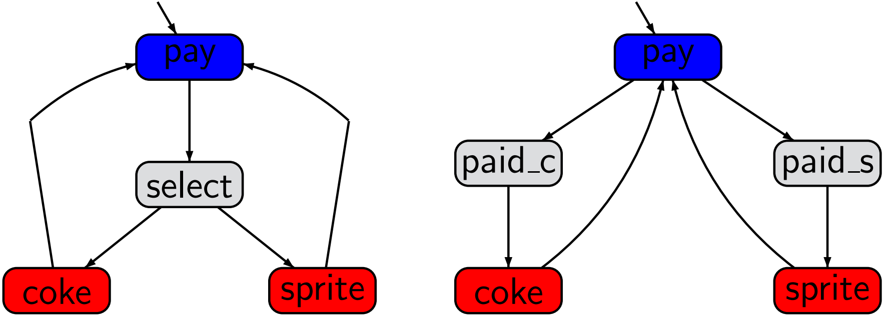
state based view: abstracts from actions and projects on atomc propositions, e.g., $AP = \{pay, drink\}$

Example: vending machine



state based view: abstracts from actions and projects on atomc propositions, e.g., $AP = \{pay, drink\}$

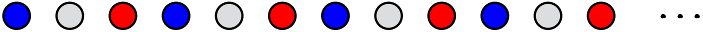
Example: vending machine



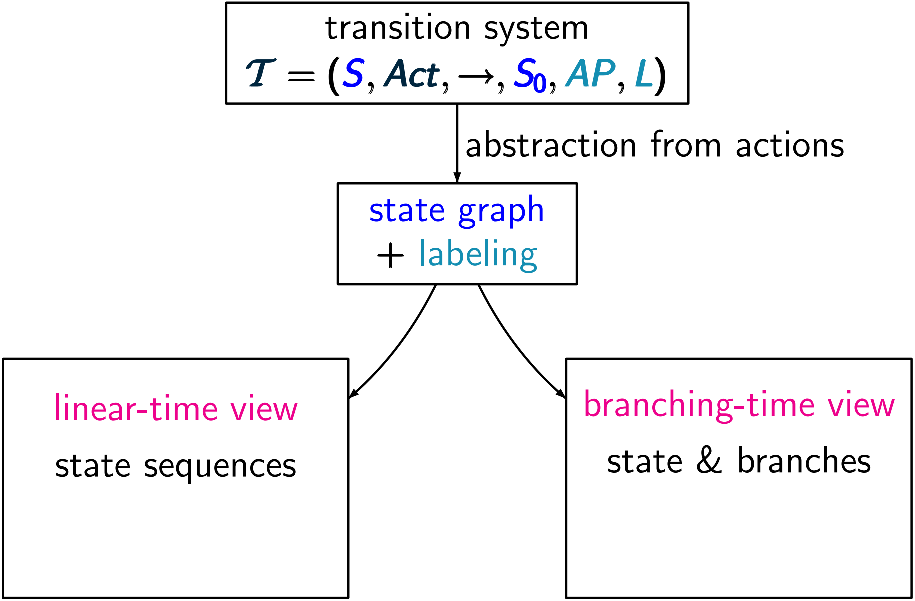
state based view: abstracts from actions and projects on atomc propositions, e.g., $AP = \{pay, drink\}$

linear & branching time:

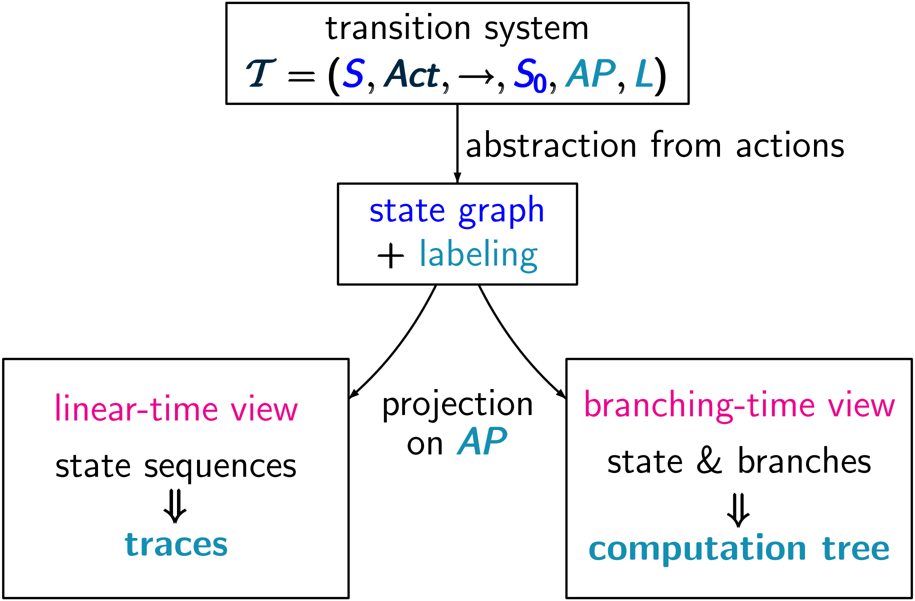
all observable behaviors have the form



Linear-time vs branching-time



Linear-time vs branching-time



Traces

LTB2.4-4

Traces

for TS with labeling function $L : S \to 2^{AP}$

execution: states + actions

$s_0 \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} s_2 \xrightarrow{\alpha_3} \ldots$ infinite or finite

↓

paths: sequences of states

$s_0\, s_1\, s_2 \ldots$ infinite or $s_0\, s_1 \ldots s_n$ finite

Traces

for TS with labeling function $L : S \to 2^{AP}$

execution: states + actions

$s_0 \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} s_2 \xrightarrow{\alpha_3} \ldots$ infinite or finite

↓

paths: sequences of states

$s_0\, s_1\, s_2 \ldots$ infinite or $s_0\, s_1 \ldots s_n$ finite

↓

traces: sequences of sets of atomic propositions

$L(s_0)\, L(s_1)\, L(s_2)\, \ldots$

Traces

for TS with labeling function $L : S \to 2^{AP}$

execution: states + actions

$s_0 \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} s_2 \xrightarrow{\alpha_3} \ldots$ infinite or finite

↓

paths: sequences of states

$s_0\, s_1\, s_2 \ldots$ infinite or $s_0\, s_1 \ldots s_n$ finite

↓

traces: sequences of sets of atomic propositions

$L(s_0)\, L(s_1)\, L(s_2)\, \ldots \quad \in (2^{AP})^\omega \cup (2^{AP})^+$

Traces

for TS with labeling function $L : S \to 2^{AP}$

execution: states + actions

$s_0 \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} s_2 \xrightarrow{\alpha_3} \ldots$ infinite or finite

↓

paths: sequences of states

$s_0\, s_1\, s_2 \ldots$ infinite or $s_0\, s_1 \ldots s_n$ finite

↓

traces: sequences of sets of atomic propositions

$L(s_0)\, L(s_1)\, L(s_2) \ldots \quad \in (2^{AP})^\omega \cup (2^{AP})^+$

for simplicity: we often assume that the given TS has **no terminal states**

Traces

for TS with labeling function $L : S \to 2^{AP}$

execution: states + actions

$s_0 \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} s_2 \xrightarrow{\alpha_3} \ldots$ infinite or ~~finite~~

↓

paths: sequences of states

$s_0\, s_1\, s_2 \ldots$ infinite or ~~$s_0\, s_1 \ldots s_n$ finite~~

↓

traces: sequences of sets of atomic propositions

$L(s_0)\, L(s_1)\, L(s_2)\, \ldots \in (2^{AP})^\omega \cup$ ~~$(2^{AP})^+$~~

for simplicity: we often assume that the given TS has **no terminal states**

Treatment of terminal states

perform standard graph algorithms to compute the reachable fragment of the given TS

$$\textit{Reach}(\mathcal{T}) = \left\{ \begin{array}{c} \text{set of states that are reachable} \\ \text{from some initial state} \end{array} \right.$$

Treatment of terminal states

perform standard graph algorithms to compute the reachable fragment of the given TS

$$Reach(\mathcal{T}) = \left\{ \begin{array}{c} \text{set of states that are reachable} \\ \text{from some initial state} \end{array} \right.$$

for each reachable terminal state s:

- if s stands for an intended halting configuration then add a transition from s to a trap state:

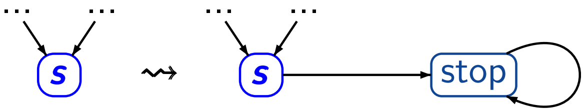
Treatment of terminal states

perform standard graph algorithms to compute the reachable fragment of the given TS

$$\mathit{Reach}(\mathcal{T}) = \left\{ \begin{array}{c} \text{set of states that are reachable} \\ \text{from some initial state} \end{array} \right.$$

for each reachable terminal state s:

- if s stands for an intended halting configuration then add a transition from s to a trap state:



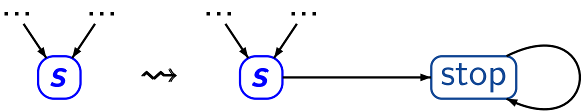
Treatment of terminal states

perform standard graph algorithms to compute the reachable fragment of the given TS

$$\mathit{Reach}(\mathcal{T}) = \left\{ \begin{array}{c} \text{set of states that are reachable} \\ \text{from some initial state} \end{array} \right.$$

for each reachable terminal state s:

- if s stands for an intended halting configuration then add a transition from s to a trap state:



- if s stands for system fault, e.g., deadlock then correct the design before checking further properties

Traces of a transition system

Let $\mathcal{T}$ be a TS

$$Traces(\mathcal{T}) \stackrel{\text{def}}{=} \{trace(\pi) : \pi \in Paths(\mathcal{T})\}$$

$$Traces_{fin}(\mathcal{T}) \stackrel{\text{def}}{=} \{trace(\widehat{\pi}) : \widehat{\pi} \in Paths_{fin}(\mathcal{T})\}$$

Traces of a transition system

Let $\mathcal{T}$ be a TS

$$Traces(\mathcal{T}) \overset{\text{def}}{=} \{trace(\pi) : \pi \in Paths(\mathcal{T})\}$$

↑ initial, maximal path fragment

$$Traces_{fin}(\mathcal{T}) \overset{\text{def}}{=} \{trace(\widehat{\pi}) : \widehat{\pi} \in Paths_{fin}(\mathcal{T})\}$$

↑ initial, finite path fragment

Traces of a transition system

Let $\mathcal{T}$ be a TS ⟵ *without* terminal states

$$Traces(\mathcal{T}) \stackrel{\text{def}}{=} \{trace(\pi) : \pi \in Paths(\mathcal{T})\}$$

↑ initial, infinite path fragment

$$Traces_{fin}(\mathcal{T}) \stackrel{\text{def}}{=} \{trace(\widehat{\pi}) : \widehat{\pi} \in Paths_{fin}(\mathcal{T})\}$$

↑ initial, finite path fragment

Traces of a transition system

Let $\mathcal{T}$ be a TS ⟵ *without* terminal states

$$Traces(\mathcal{T}) \stackrel{\text{def}}{=} \{trace(\pi) : \pi \in Paths(\mathcal{T})\} \subseteq (2^{AP})^\omega$$

↑ initial, infinite path fragment

$$Traces_{fin}(\mathcal{T}) \stackrel{\text{def}}{=} \{trace(\widehat{\pi}) : \widehat{\pi} \in Paths_{fin}(\mathcal{T})\} \subseteq (2^{AP})^*$$

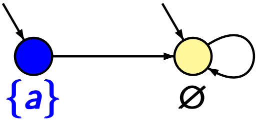
↑ initial, finite path fragment

Example: traces

Let $\mathcal{T}$ be a TS without terminal states.

$$Traces(\mathcal{T}) \stackrel{\text{def}}{=} \{trace(\pi) : \pi \in Paths(\mathcal{T})\} \subseteq (2^{AP})^{\omega}$$

$$Traces_{fin}(\mathcal{T}) \stackrel{\text{def}}{=} \{trace(\widehat{\pi}) : \widehat{\pi} \in Paths_{fin}(\mathcal{T})\} \subseteq (2^{AP})^{*}$$



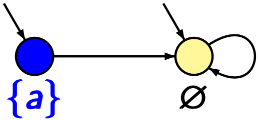
TS $\mathcal{T}$ with a single atomic proposition a

Example: traces

Let $\mathcal{T}$ be a TS without terminal states.

$$Traces(\mathcal{T}) \stackrel{\text{def}}{=} \{trace(\pi) : \pi \in Paths(\mathcal{T})\} \subseteq (2^{AP})^{\omega}$$

$$Traces_{fin}(\mathcal{T}) \stackrel{\text{def}}{=} \{trace(\widehat{\pi}) : \widehat{\pi} \in Paths_{fin}(\mathcal{T})\} \subseteq (2^{AP})^{*}$$

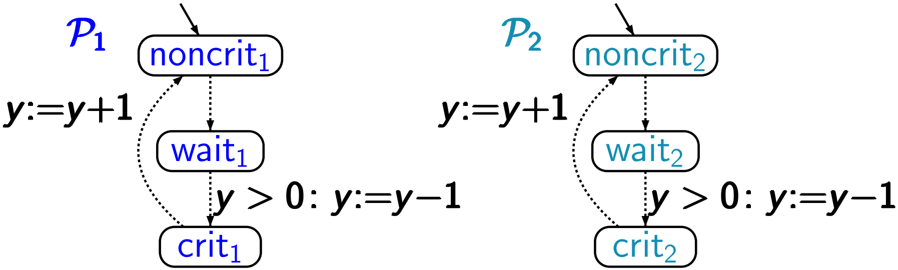


TS $\mathcal{T}$ with a single atomic proposition a

$$Traces(\mathcal{T}) = \{\{a\}\varnothing^{\omega}, \varnothing^{\omega}\}$$

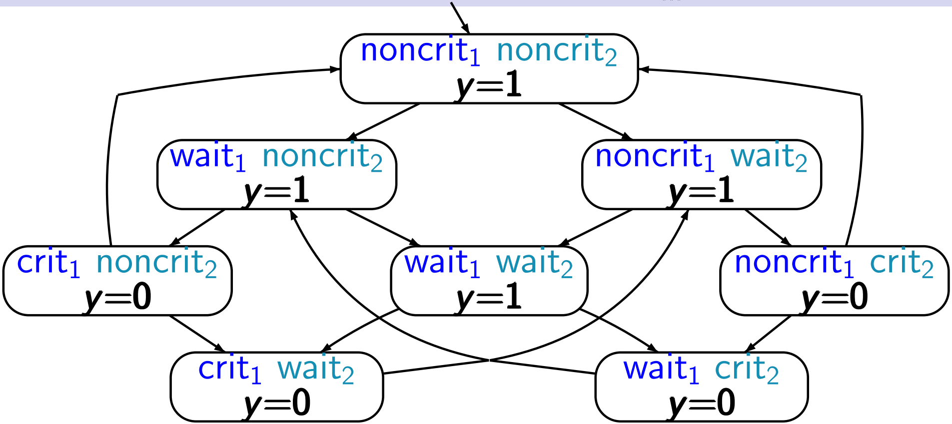
$$Traces_{fin}(\mathcal{T}) = \{\{a\}\varnothing^{n} : n \geq 0\} \cup \{\varnothing^{m} : m \geq 1\}$$

Mutual exclusion with semaphore



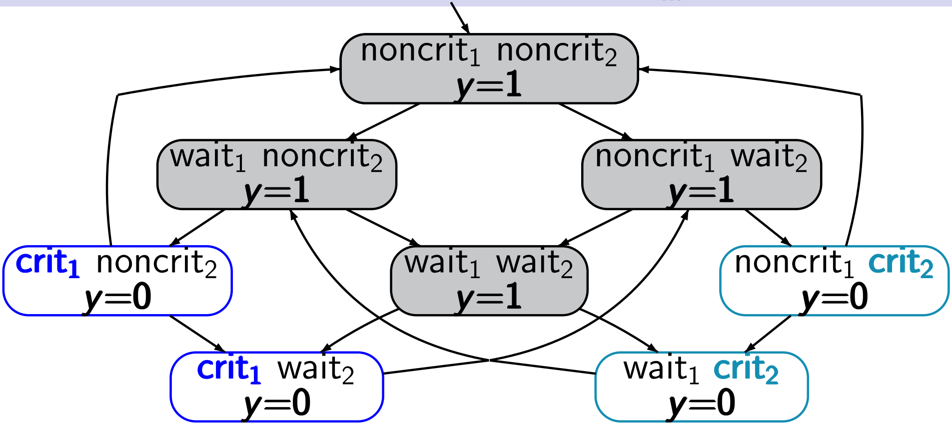
transition system $\mathcal{T}_{\mathcal{P}_1 ||| \mathcal{P}_2}$ arises by unfolding the composite program graph $\mathcal{P}_1 ||| \mathcal{P}_2$

Mutual exclusion with semaphore $\mathcal{T}_{\mathcal{P}_1 ||| \mathcal{P}_2}$



set of atomic propositions $AP = \{\mathbf{crit_1}, \mathbf{crit_2}\}$

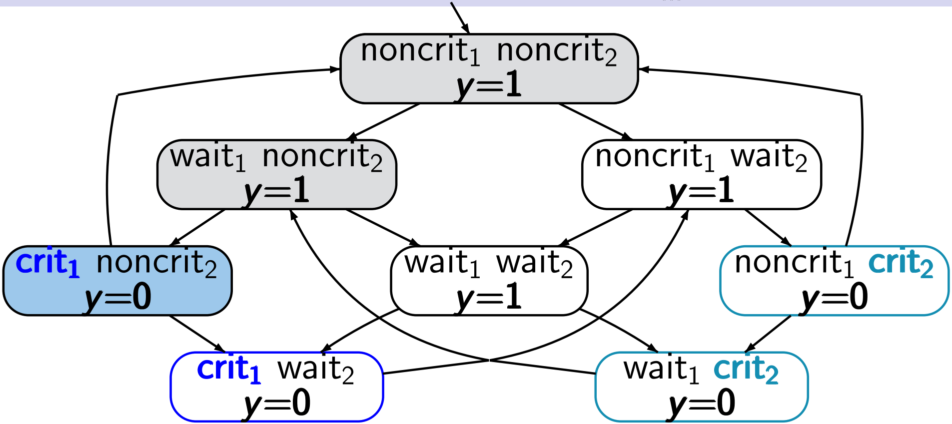
Mutual exclusion with semaphore $\mathcal{T}_{\mathcal{P}_1 ||| \mathcal{P}_2}$



set of atomic propositions $AP = \{crit_1, crit_2\}$

e.g., $L(\langle noncrit_1, noncrit_2, y{=}1\rangle) =$
$L(\langle wait_1, noncrit_2, y{=}1\rangle) = \varnothing$

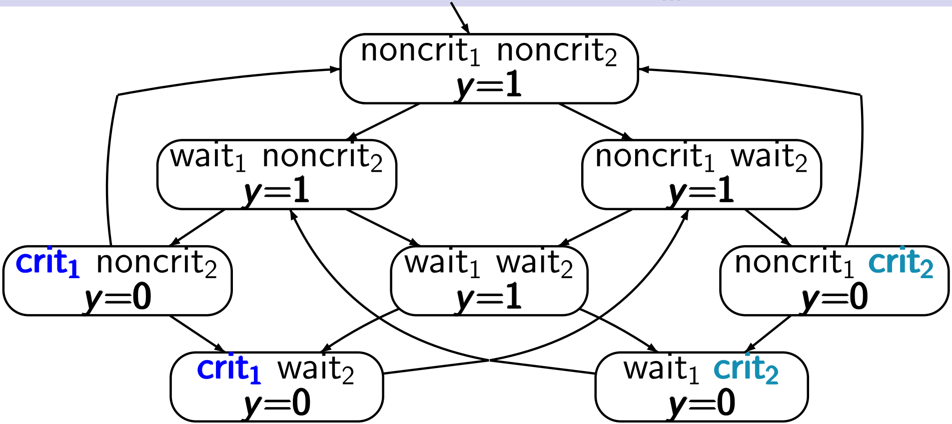
Mutual exclusion with semaphore $\mathcal{T}_{\mathcal{P}_1 ||| \mathcal{P}_2}$



set of atomic propositions $AP = \{crit_1, crit_2\}$

traces, e.g., $\varnothing\ \varnothing\ \{crit_1\}\ \varnothing\ \varnothing\ \{crit_1\}\ \varnothing\ \varnothing\ \{crit_1\} \dots$

Mutual exclusion with semaphore $\mathcal{T}_{\mathcal{P}_1 ||| \mathcal{P}_2}$

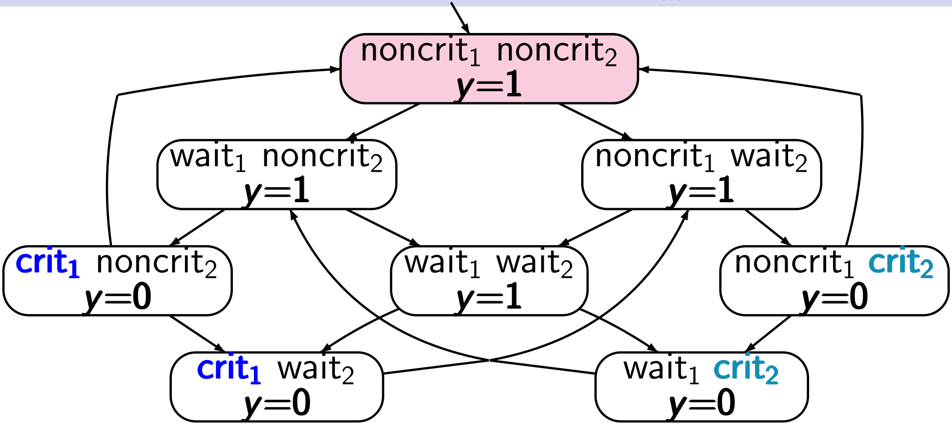


set of atomic propositions $AP = \{crit_1, crit_2\}$

traces, e.g., $\varnothing\ \varnothing\ \{crit_1\}\ \varnothing\ \varnothing\ \{crit_1\}\ \varnothing\ \varnothing\ \{crit_1\} \ldots$

$\varnothing\ \varnothing\ \varnothing\ \{crit_1\}\ \varnothing\ \{crit_2\}\ \{crit_2\}\ \varnothing \ldots$

Mutual exclusion with semaphore $\mathcal{T}_{\mathcal{P}_1 ||| \mathcal{P}_2}$

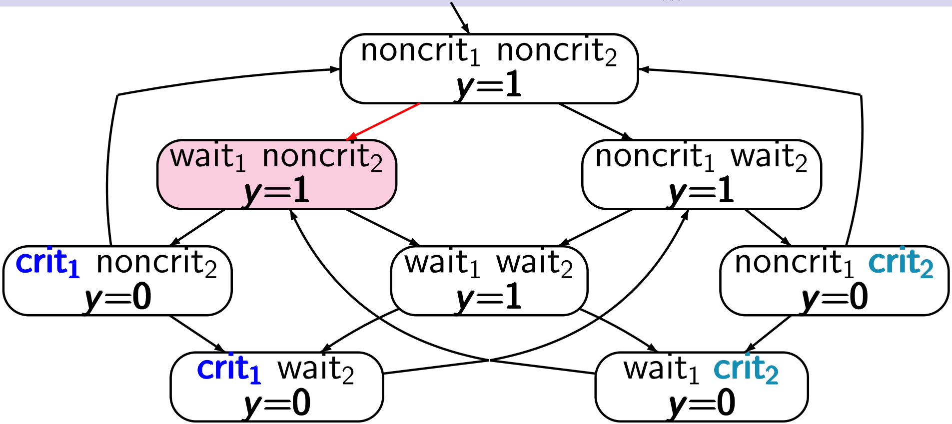


set of atomic propositions $AP = \{crit_1, crit_2\}$

traces, e.g., $\varnothing\ \varnothing\ \{crit_1\}\ \varnothing\ \varnothing\ \{crit_1\}\ \varnothing\ \varnothing\ \{crit_1\} \ldots$

$\varnothing\ \varnothing\ \varnothing\ \{crit_1\}\ \varnothing\ \{crit_2\}\ \{crit_2\}\ \varnothing \ldots$

Mutual exclusion with semaphore $\mathcal{T}_{\mathcal{P}_1 ||| \mathcal{P}_2}$

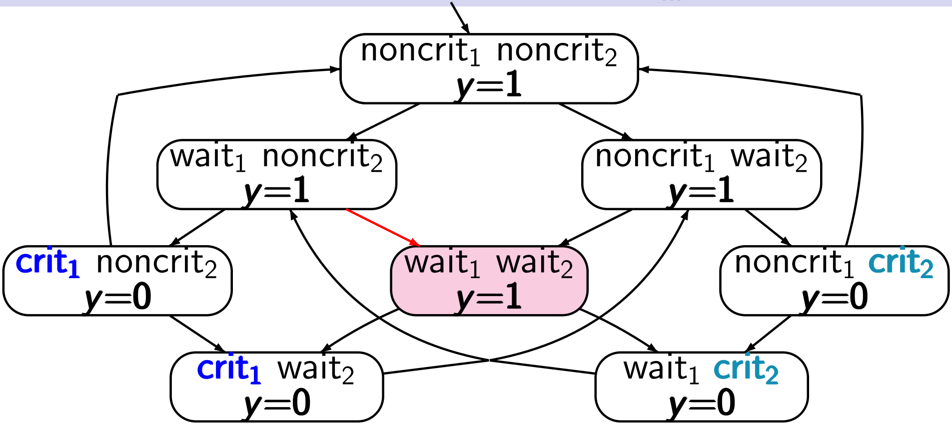


set of atomic propositions $AP = \{crit_1, crit_2\}$

traces, e.g., $\varnothing\ \varnothing\ \{crit_1\}\ \varnothing\ \varnothing\ \{crit_1\}\ \varnothing\ \varnothing\ \{crit_1\} \ldots$

$\varnothing\ \varnothing\ \varnothing\ \{crit_1\}\ \varnothing\ \{crit_2\}\ \{crit_2\}\ \varnothing \ldots$

Mutual exclusion with semaphore $\mathcal{T}_{\mathcal{P}_1 ||| \mathcal{P}_2}$

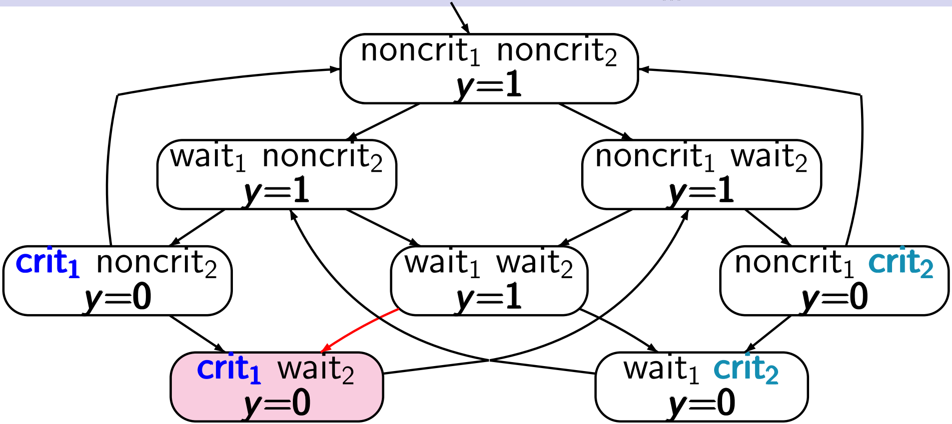


set of atomic propositions $AP = \{crit_1, crit_2\}$

traces, e.g., $\varnothing\ \varnothing\ \{crit_1\}\ \varnothing\ \varnothing\ \{crit_1\}\ \varnothing\ \varnothing\ \{crit_1\} \ldots$

$\varnothing\ \varnothing\ \varnothing\ \{crit_1\}\ \varnothing\ \{crit_2\}\ \{crit_2\}\ \varnothing \ldots$

Mutual exclusion with semaphore $\mathcal{T}_{\mathcal{P}_1 ||| \mathcal{P}_2}$

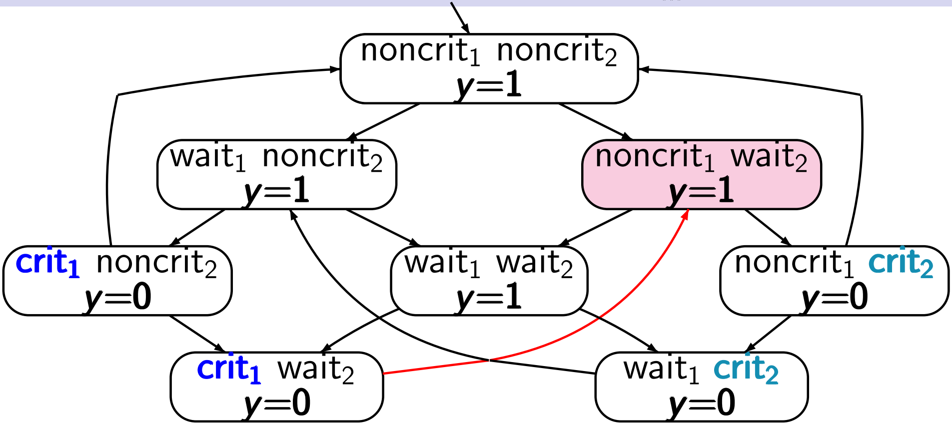


set of atomic propositions $AP = \{crit_1, crit_2\}$

traces, e.g., $\varnothing\ \varnothing\ \{crit_1\}\ \varnothing\ \varnothing\ \{crit_1\}\ \varnothing\ \varnothing\ \{crit_1\} \ldots$

$\varnothing\ \varnothing\ \varnothing\ \{crit_1\}\ \varnothing\ \{crit_2\}\ \{crit_2\}\ \varnothing \ldots$

Mutual exclusion with semaphore $\mathcal{T}_{\mathcal{P}_1 ||| \mathcal{P}_2}$

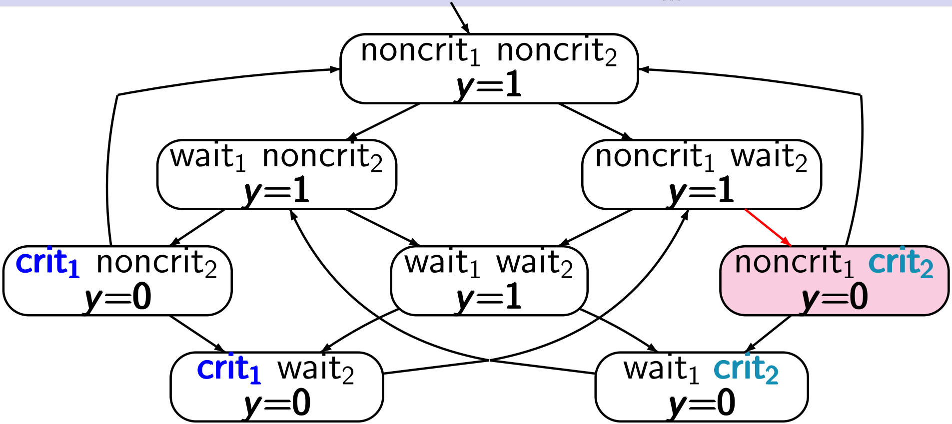


set of atomic propositions $AP = \{\text{crit}_1, \text{crit}_2\}$

traces, e.g., $\varnothing\ \varnothing\ \{\text{crit}_1\}\ \varnothing\ \varnothing\ \{\text{crit}_1\}\ \varnothing\ \varnothing\ \{\text{crit}_1\} \ldots$

$\varnothing\ \varnothing\ \varnothing\ \{\text{crit}_1\}\ \varnothing\ \{\text{crit}_2\}\ \{\text{crit}_2\}\ \varnothing \ldots$

Mutual exclusion with semaphore $\mathcal{T}_{\mathcal{P}_1 ||| \mathcal{P}_2}$

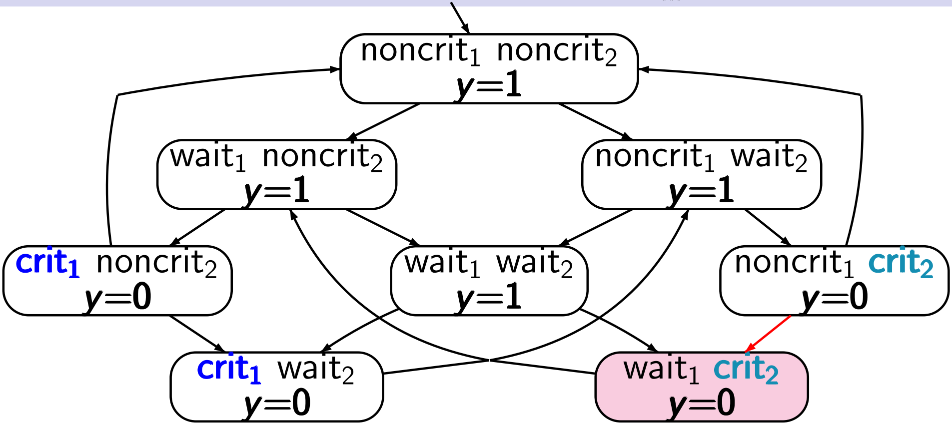


set of atomic propositions $AP = \{crit_1, crit_2\}$

traces, e.g., $\varnothing\ \varnothing\ \{crit_1\}\ \varnothing\ \varnothing\ \{crit_1\}\ \varnothing\ \varnothing\ \{crit_1\} \ldots$

$\varnothing\ \varnothing\ \varnothing\ \{crit_1\}\ \varnothing\ \{crit_2\}\ \{crit_2\}\ \varnothing \ldots$

Mutual exclusion with semaphore $\mathcal{T}_{\mathcal{P}_1 ||| \mathcal{P}_2}$

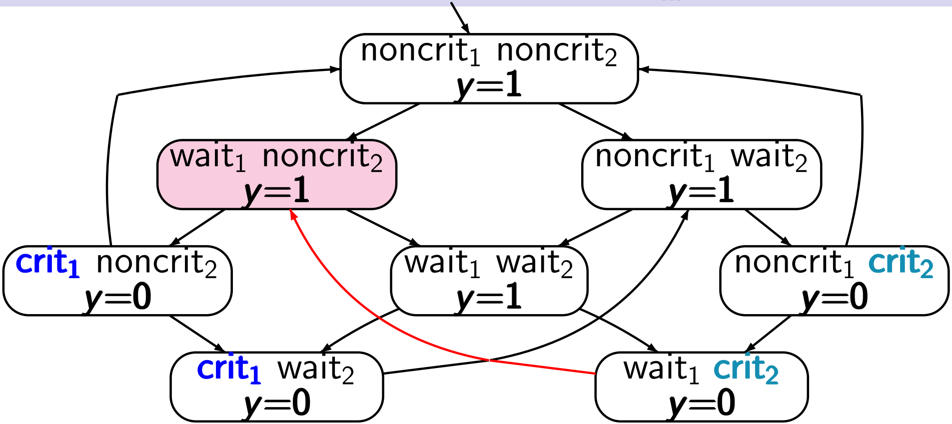


set of atomic propositions $AP = \{crit_1, crit_2\}$

traces, e.g., $\varnothing\ \varnothing\ \{crit_1\}\ \varnothing\ \varnothing\ \{crit_1\}\ \varnothing\ \varnothing\ \{crit_1\} \ldots$

$\varnothing\ \varnothing\ \varnothing\ \{crit_1\}\ \varnothing\ \{crit_2\}\ \{crit_2\}\ \varnothing \ldots$

Mutual exclusion with semaphore $\mathcal{T}_{\mathcal{P}_1 ||| \mathcal{P}_2}$

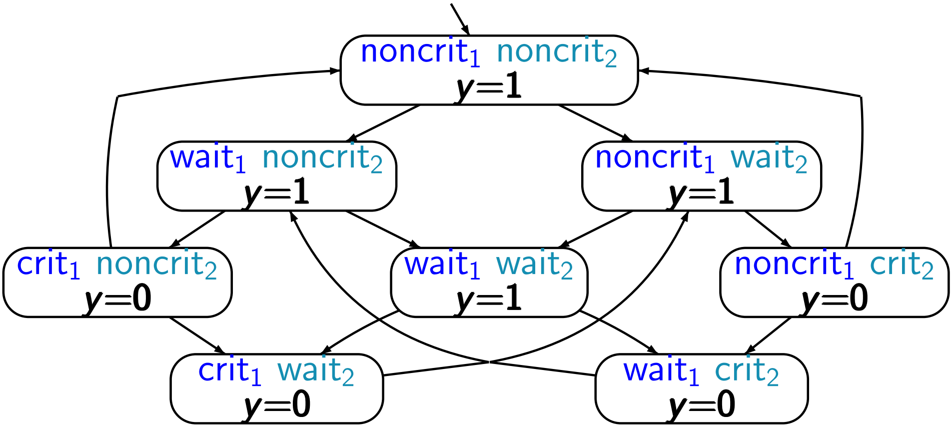


set of atomic propositions $AP = \{crit_1, crit_2\}$

traces, e.g., $\varnothing\ \varnothing\ \{crit_1\}\ \varnothing\ \varnothing\ \{crit_1\}\ \varnothing\ \varnothing\ \{crit_1\} \ldots$

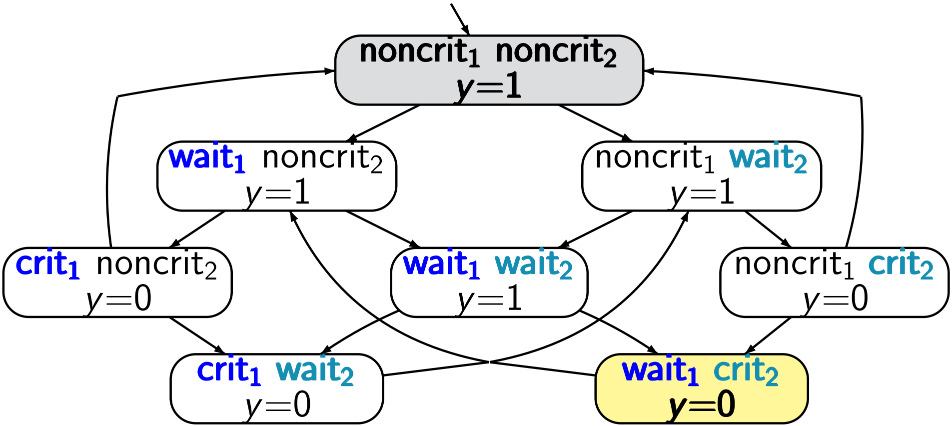
$\varnothing\ \varnothing\ \varnothing\ \{crit_1\}\ \varnothing\ \{crit_2\}\ \{crit_2\}\ \varnothing \ldots$

Mutual exclusion with semaphor $\mathcal{T}_{\mathcal{P}_1 ||| \mathcal{P}_2}$



set of propositions $AP = \{\mathbf{wait_1}, \mathbf{crit_1}, \mathbf{wait_2}, \mathbf{crit_2}\}$

Mutual exclusion with semaphor $\mathcal{T}_{\mathcal{P}_1|||\mathcal{P}_2}$

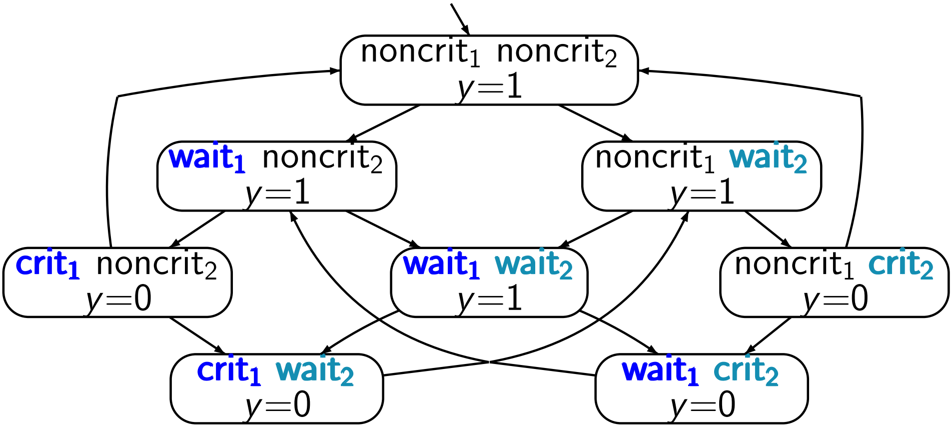


set of propositions $AP = \{\text{wait}_1, \text{crit}_1, \text{wait}_2, \text{crit}_2\}$

e.g., $L(\langle \text{noncrit}_1, \text{noncrit}_2, y{=}1\rangle) = \varnothing$

$L(\langle \text{wait}_1, \text{crit}_2, y{=}1\rangle) = \{\text{wait}_1, \text{crit}_2\}$

Mutual exclusion with semaphor $\mathcal{T}_{\mathcal{P}_1 ||| \mathcal{P}_2}$

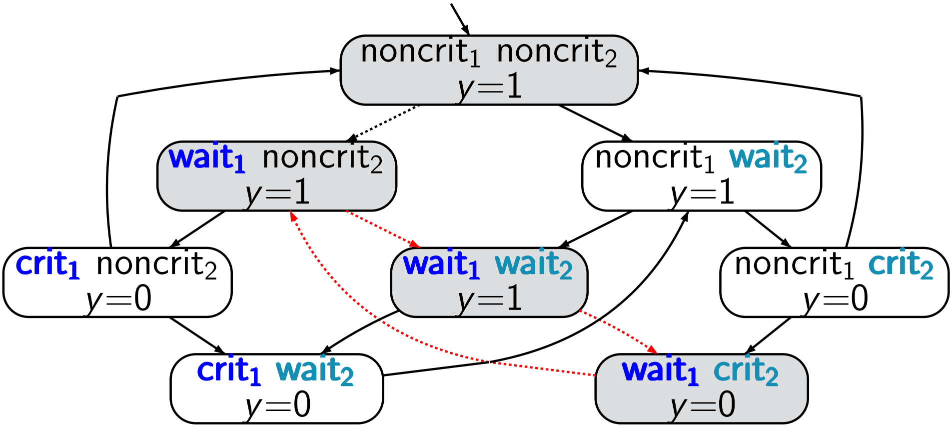


set of propositions $AP = \{wait_1, crit_1, wait_2, crit_2\}$

traces, e.g.,

$$\varnothing \left(\{wait_1\}\ \{wait_1, wait_2\}\ \{wait_1, crit_2\}\right)^{\omega}$$

Mutual exclusion with semaphor $\mathcal{T}_{\mathcal{P}_1 ||| \mathcal{P}_2}$



set of propositions $AP = \{wait_1, crit_1, wait_2, crit_2\}$

traces, e.g.,

$$\varnothing \left(\{wait_1\}\,\{wait_1, wait_2\}\,\{wait_1, crit_2\}\right)^\omega$$

Introduction

Modelling parallel systems

Linear Time Properties

- state-based and linear time view
- definition of linear time properties ⟵
- invariants and safety
- liveness and fairness

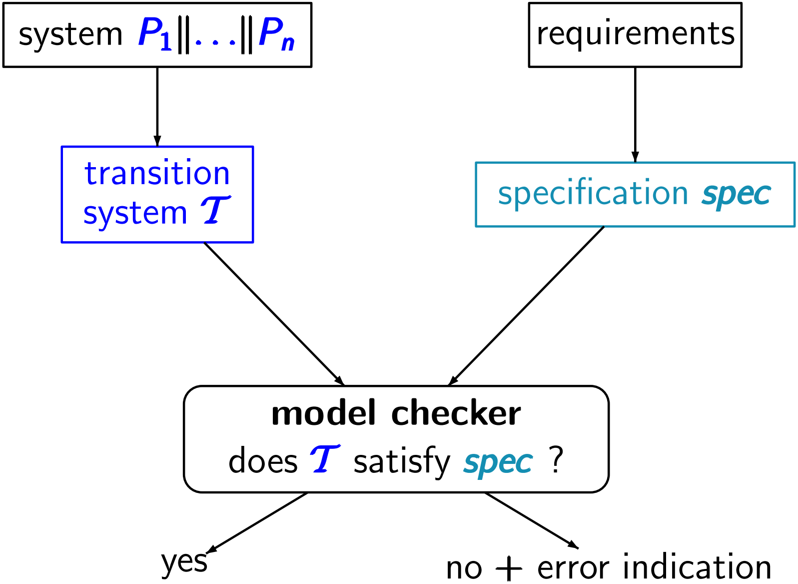
Regular Properties

Linear Temporal Logic

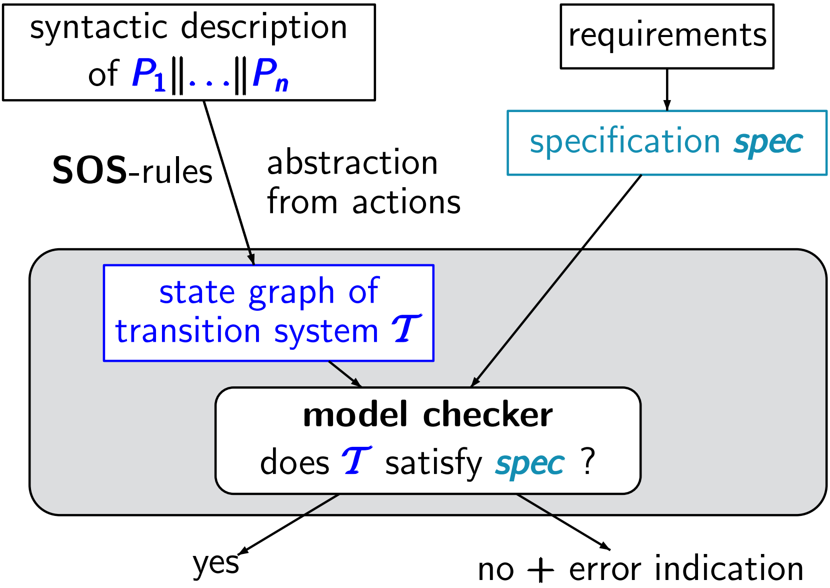
Computation-Tree Logic

Equivalences and Abstraction

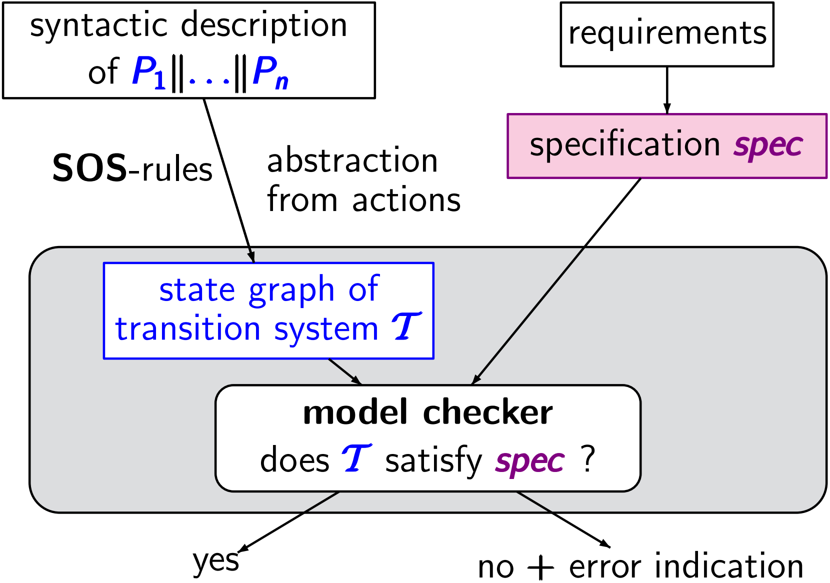
Model checking



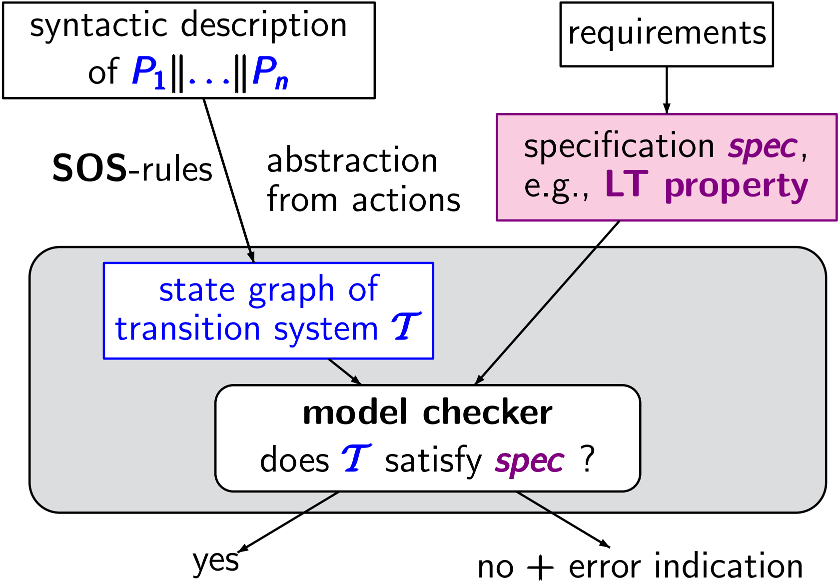
Model checking



Model checking



Model checking



Linear-time properties (LT properties)

Linear-time properties (LT properties)

for TS over AP without terminal states

An LT property over AP is a language E of infinite words over the alphabet $\Sigma = 2^{AP}$,

Linear-time properties (LT properties)

for TS over AP without terminal states

An LT property over AP is a language E of infinite words over the alphabet $\Sigma = 2^{AP}$, i.e., $E \subseteq \left(2^{AP}\right)^{\omega}$.

Linear-time properties (LT properties)

for TS over AP without terminal states

An LT property over AP is a language E of infinite words over the alphabet $\Sigma = 2^{AP}$, i.e., $E \subseteq \left(2^{AP}\right)^{\omega}$.

E.g., for mutual exclusion problems and
$AP = \{crit_1, crit_2, \ldots\}$

safety:

$MUTEX$ = set of all infinite words $A_0 \, A_1 \, A_2 \ldots$ over 2^{AP} such that for all $i \in \mathbb{N}$:
$crit_1 \notin A_i$ or $crit_2 \notin A_i$

LT properties for mutual exclusion protocols

$AP = \{wait_1, crit_1, wait_2, crit_2\}$

safety:

$MUTEX$ = set of all infinite words $A_0\, A_1\, A_2 \ldots$ over 2^{AP} such that for all $i \in \mathbb{N}$:
$crit_1 \notin A_i$ or $crit_2 \notin A_i$

$\varnothing\ \{wait_1\}\ \{crit_1\}\ \varnothing\ \{wait_1\}\ \{crit_1\}\ \ldots \qquad \in MUTEX$

LT properties for mutual exclusion protocols

$AP = \{\text{wait}_1, \text{crit}_1, \text{wait}_2, \text{crit}_2\}$

safety:

$MUTEX$ = set of all infinite words $A_0 A_1 A_2 \ldots$ over 2^{AP} such that for all $i \in \mathbb{N}$: $\text{crit}_1 \notin A_i$ or $\text{crit}_2 \notin A_i$

$\varnothing \{\text{wait}_1\} \{\text{crit}_1\} \varnothing \{\text{wait}_1\} \{\text{crit}_1\} \ldots \in MUTEX$

$\varnothing \{\text{wait}_1\} \{\text{crit}_1\} \{\text{crit}_1, \text{wait}_2\} \{\text{crit}_1, \text{crit}_2\} \ldots \notin MUTEX$

LT properties for mutual exclusion protocols

$AP = \{wait_1, crit_1, wait_2, crit_2\}$

safety:

$MUTEX$ = set of all infinite words $A_0\,A_1\,A_2\ldots$ over 2^{AP} such that for all $i \in \mathbb{N}$:
$crit_1 \notin A_i$ or $crit_2 \notin A_i$

$\varnothing\ \{wait_1\}\ \{crit_1\}\ \varnothing\ \{wait_1\}\ \{crit_1\}\ \ldots \quad \in MUTEX$

$\varnothing\ \{wait_1\}\ \{crit_1\}\ \{crit_1, wait_2\}\ \{crit_1, crit_2\} \ldots \quad \notin MUTEX$

$\varnothing\ \varnothing\ \{wait_1, crit_1, crit_2\} \ldots \quad \notin MUTEX$

LT properties for mutual exclusion protocols

$AP = \{wait_1, crit_1, wait_2, crit_2\}$

safety:

$MUTEX =$ set of all infinite words $A_0\, A_1\, A_2 \ldots$ over 2^{AP} such that for all $i \in \mathbb{N}$:

$crit_1 \notin A_i$ or $crit_2 \notin A_i$

liveness (starvation freedom):

$LIVE =$ set of all infinite words $A_0\, A_1\, A_2 \ldots$ s.t.

$$\overset{\infty}{\exists}\, i \in \mathbb{N}.wait_1 \in A_i \implies \overset{\infty}{\exists}\, i \in \mathbb{N}.crit_1 \in A_i$$

$$\wedge \quad \overset{\infty}{\exists}\, i \in \mathbb{N}.wait_2 \in A_i \implies \overset{\infty}{\exists}\, i \in \mathbb{N}.crit_2 \in A_i$$

Satisfaction relation for LT properties

Satisfaction relation for LT properties

An LT property over AP is a language E of infinite words over the alphabet $\Sigma = 2^{AP}$, i.e., $E \subseteq \left(2^{AP}\right)^{\omega}$.

Satisfaction relation for LT properties

An LT property over AP is a language E of infinite words over the alphabet $\Sigma = 2^{AP}$, i.e., $E \subseteq \left(2^{AP}\right)^{\omega}$.

Satisfaction relation $\models$ for TS:

If $\mathcal{T}$ is a TS (without terminal states) over AP and E an LT property over AP then

$$\mathcal{T} \models E \quad \text{iff} \quad \mathit{Traces}(\mathcal{T}) \subseteq E$$

Satisfaction relation for LT properties

An LT property over AP is a language E of infinite words over the alphabet $\Sigma = 2^{AP}$, i.e., $E \subseteq \left(2^{AP}\right)^{\omega}$.

Satisfaction relation $\models$ for TS and states:

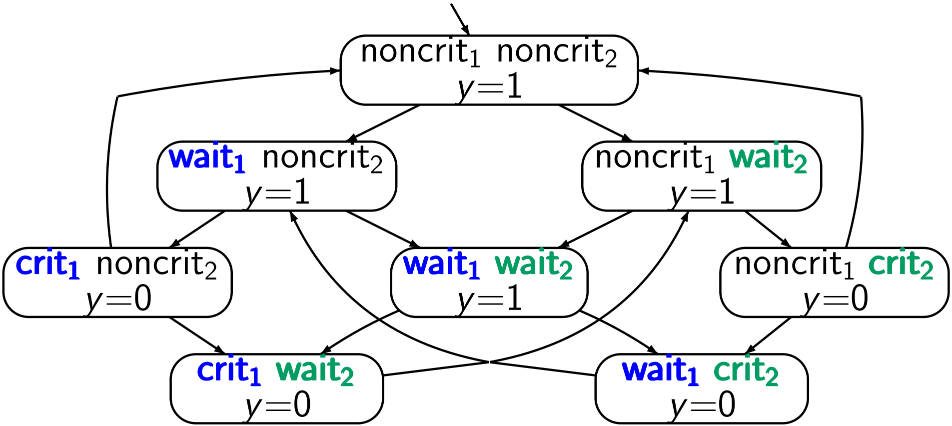
If $\mathcal{T}$ is a TS (without terminal states) over AP and E an LT property over AP then

$$\mathcal{T} \models E \quad \text{iff} \quad Traces(\mathcal{T}) \subseteq E$$

If s is a state in $\mathcal{T}$ then

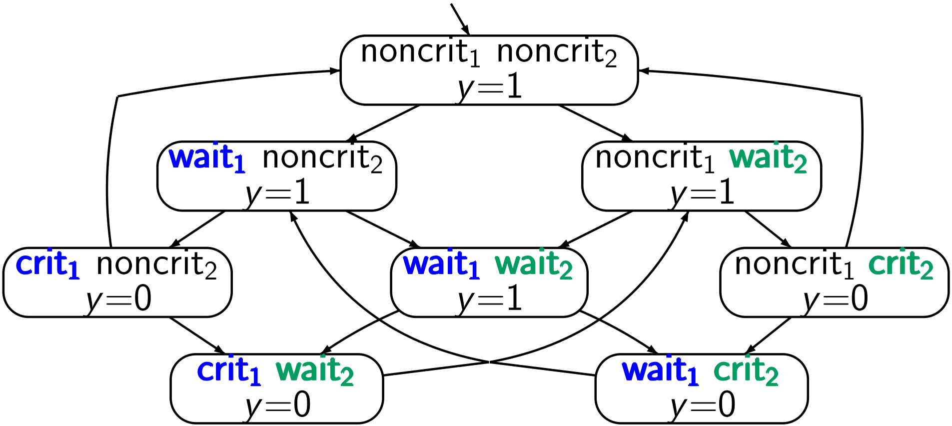
$$s \models E \quad \text{iff} \quad Traces(s) \subseteq E$$

Mutual exclusion with semaphore



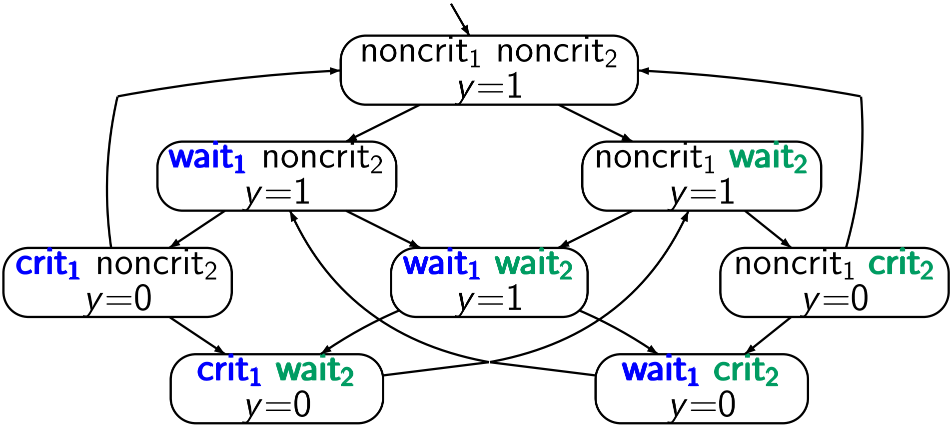
$\mathcal{T}_{Sem} \models MUTEX$

Mutual exclusion with semaphore



$\mathcal{T}_{Sem} \models MUTEX, \quad \mathcal{T}_{Sem} \models LIVE$?

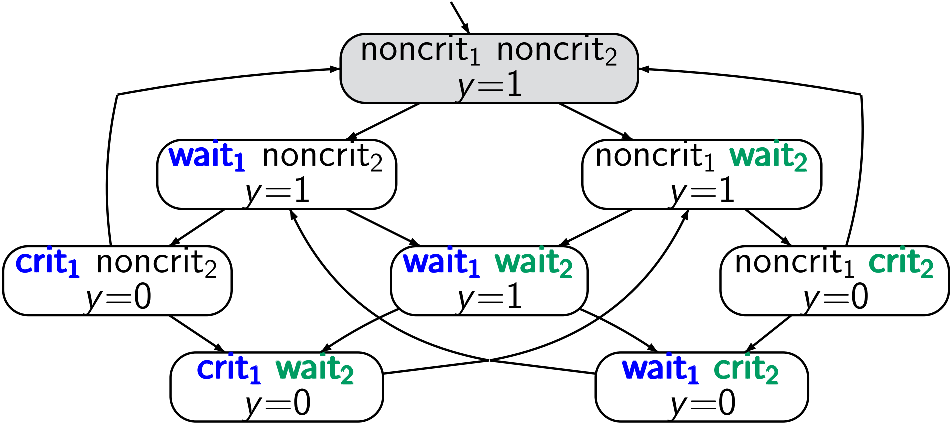
Mutual exclusion with semaphore



$$\mathcal{T}_{Sem} \models \mathit{MUTEX}, \quad \mathcal{T}_{Sem} \not\models \mathit{LIVE}$$

$$\varnothing\ \{\mathsf{wait}_1\}\ \big(\ \{\mathsf{wait}_1, \mathsf{wait}_2\}\ \{\mathsf{crit}_1, \mathsf{wait}_2\}\{\mathsf{wait}_2\}\ \big)^{\omega} \notin \mathit{LIVE}$$

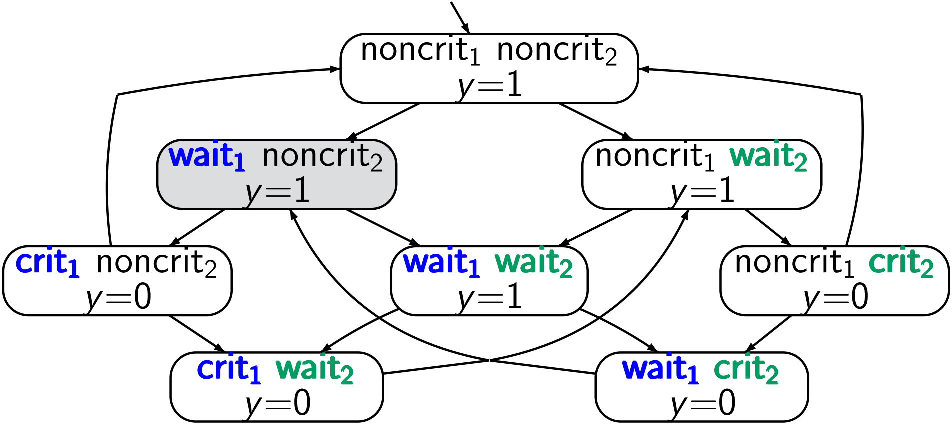
Mutual exclusion with semaphore



$$\mathcal{T}_{Sem} \models MUTEX, \quad \mathcal{T}_{Sem} \not\models LIVE$$

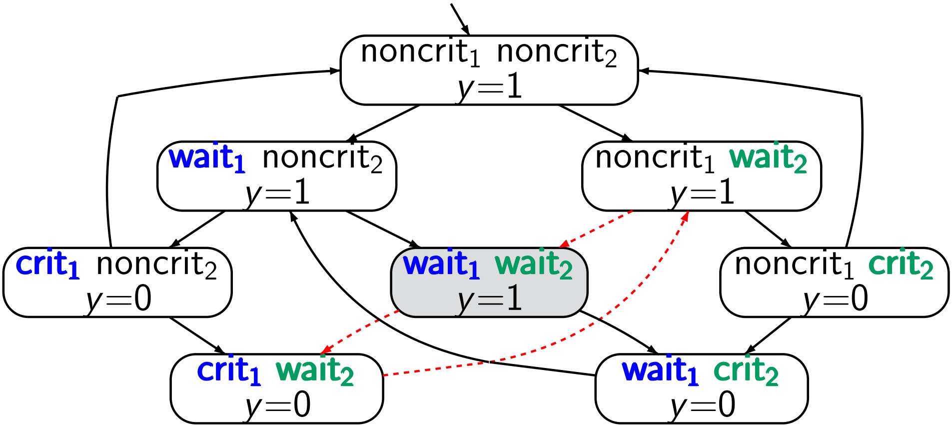
$$\varnothing \, \{wait_1\} \left(\{wait_1, wait_2\} \, \{crit_1, wait_2\} \{wait_2\} \right)^{\omega} \notin LIVE$$

Mutual exclusion with semaphore



$$\mathcal{T}_{Sem} \models \mathit{MUTEX}, \quad \mathcal{T}_{Sem} \not\models \mathit{LIVE}$$

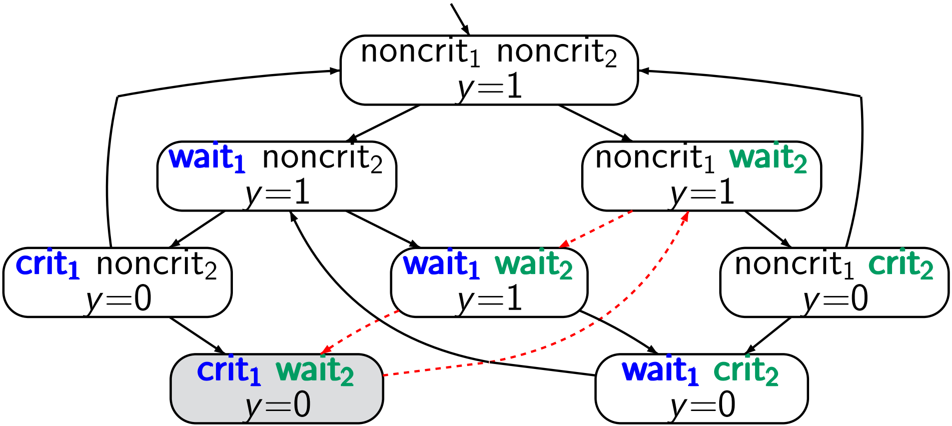
$$\varnothing \, \{\mathsf{wait}_1\} \, \big(\{\mathsf{wait}_1, \mathsf{wait}_2\} \, \{\mathsf{crit}_1, \mathsf{wait}_2\} \{\mathsf{wait}_2\} \big)^\omega \notin \mathit{LIVE}$$



$\mathcal{T}_{Sem} \models \mathit{MUTEX}, \quad \mathcal{T}_{Sem} \not\models \mathit{LIVE}$

$\varnothing \, \{\text{wait}_1\} \left(\{\text{wait}_1, \text{wait}_2\} \, \{\text{crit}_1, \text{wait}_2\} \{\text{wait}_2\} \right)^{\omega} \notin \mathit{LIVE}$

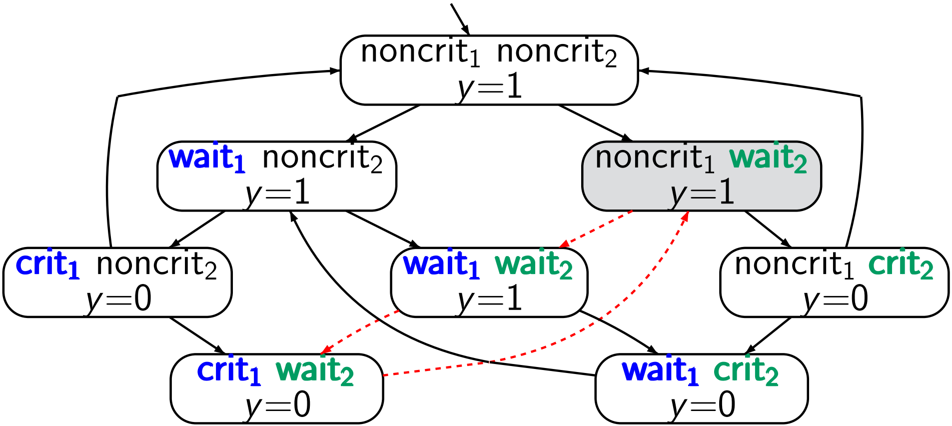
Mutual exclusion with semaphore



$\mathcal{T}_{Sem} \models MUTEX, \quad \mathcal{T}_{Sem} \not\models LIVE$

$\varnothing \, \{wait_1\} \, \big(\{wait_1, wait_2\} \, \{crit_1, wait_2\} \{wait_2\} \big)^\omega \notin LIVE$

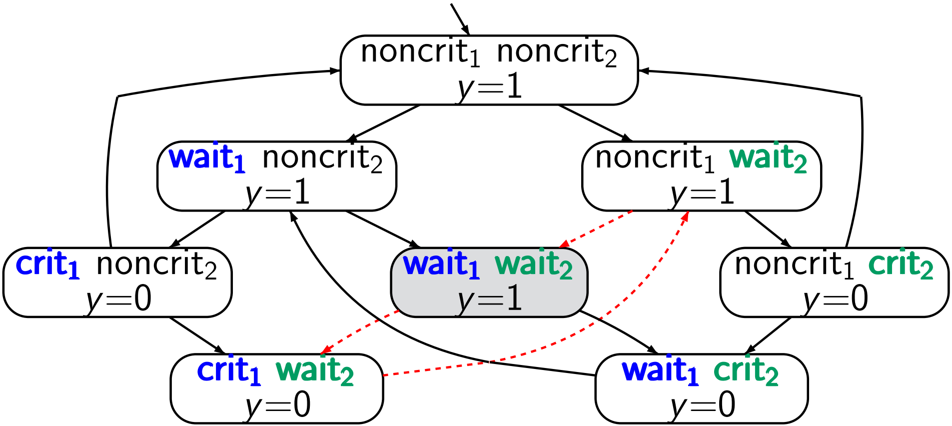
Mutual exclusion with semaphore



$\mathcal{T}_{Sem} \models MUTEX, \quad \mathcal{T}_{Sem} \not\models LIVE$

$\varnothing \, \{wait_1\} \left(\{wait_1, wait_2\} \, \{crit_1, wait_2\} \{wait_2\} \right)^\omega \notin LIVE$

Mutual exclusion with semaphore



$$\mathcal{T}_{Sem} \models MUTEX, \quad \mathcal{T}_{Sem} \not\models LIVE$$

$$\varnothing\ \{\text{wait}_1\}\ \big(\ \{\text{wait}_1, \text{wait}_2\}\ \{\text{crit}_1, \text{wait}_2\}\{\text{wait}_2\}\ \big)^{\omega} \notin LIVE$$

Peterson's mutual exclusion algorithm

Peterson's mutual exclusion algorithm

for competing processes $\mathcal{P}_1$ and $\mathcal{P}_2$,

using three additional shared variables

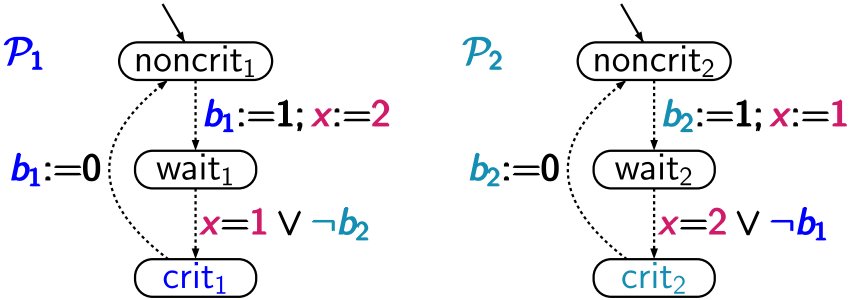
$b_1, b_2 \in \{0,1\}$, $x \in \{1,2\}$

Peterson's mutual exclusion algorithm

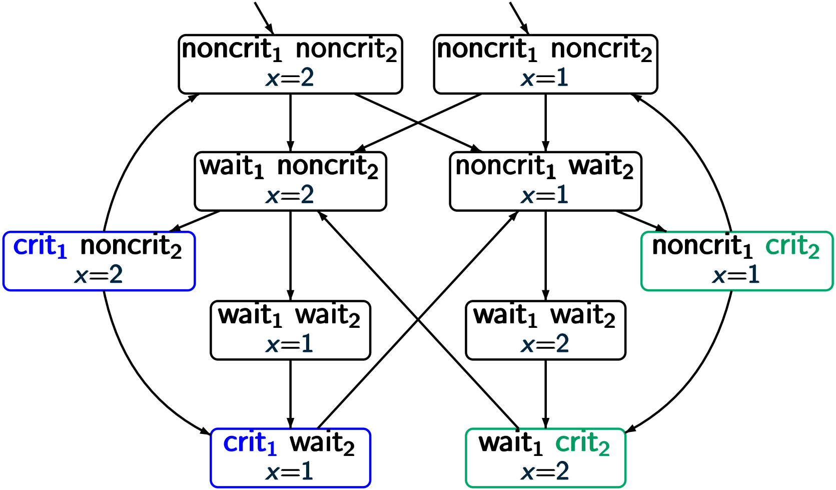
for competing processes $\mathcal{P}_1$ and $\mathcal{P}_2$,

using three additional shared variables

$b_1, b_2 \in \{0,1\}$, $x \in \{1,2\}$

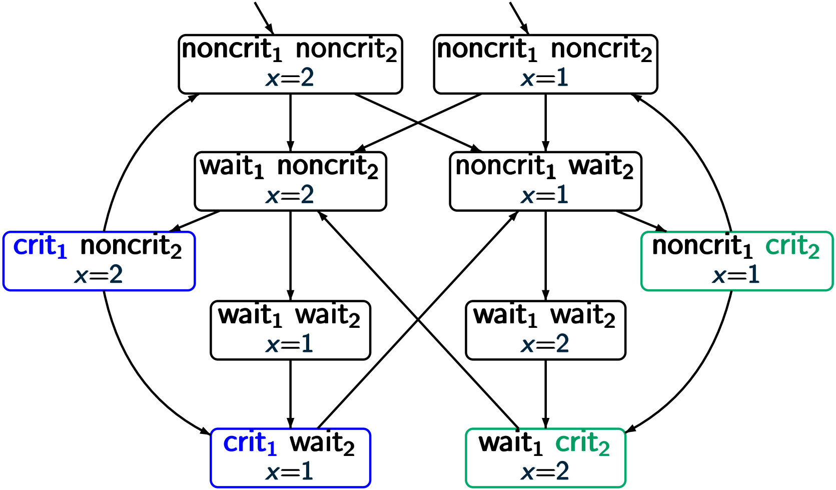


Peterson's mutual exclusion algorithm



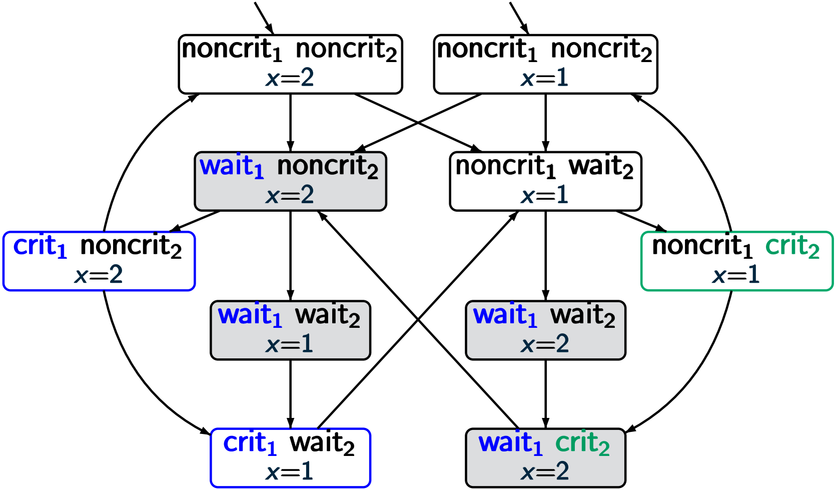
$\mathcal{T}_{Pet} \models \mathit{MUTEX}$

Peterson's mutual exclusion algorithm



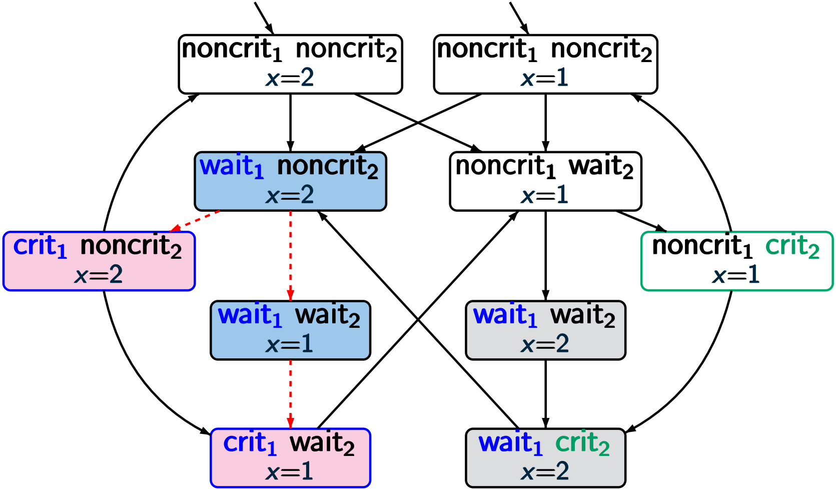
$\mathcal{T}_{Pet} \models \mathit{MUTEX}$ and $\mathcal{T}_{Pet} \models \mathit{LIVE}$

Peterson's mutual exclusion algorithm



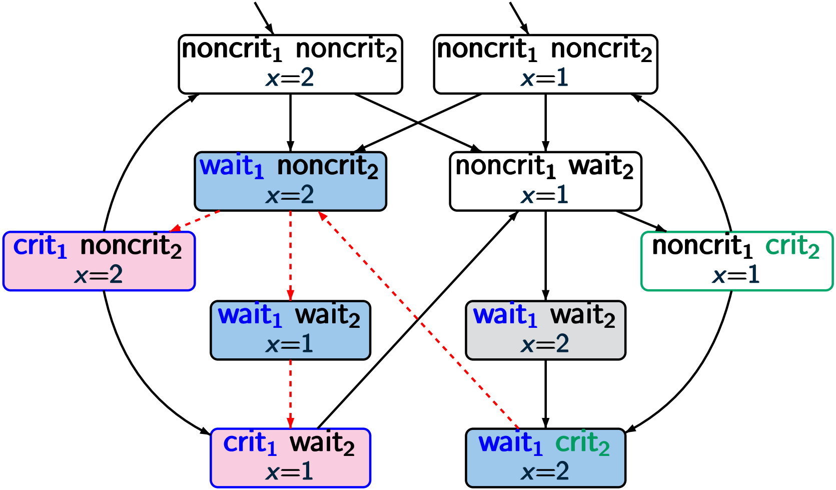
$\mathcal{T}_{Pet} \models MUTEX$ and $\mathcal{T}_{Pet} \models LIVE$

Peterson's mutual exclusion algorithm



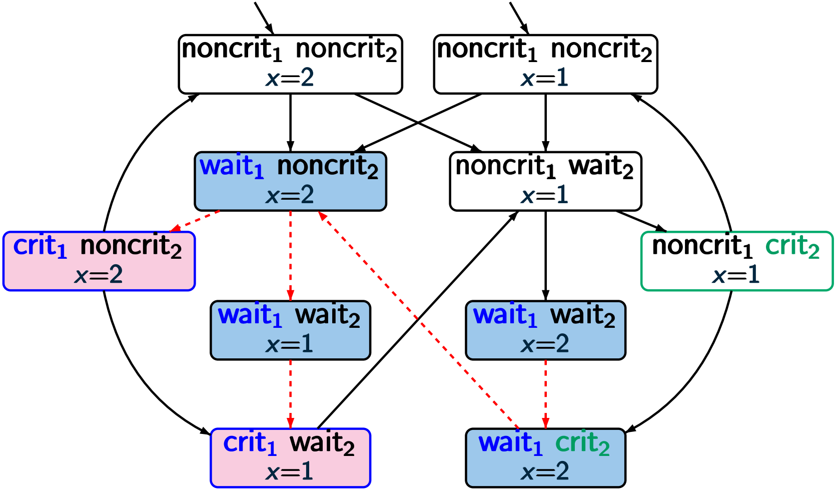
$\mathcal{T}_{Pet} \models \mathit{MUTEX}$ and $\mathcal{T}_{Pet} \models \mathit{LIVE}$

Peterson's mutual exclusion algorithm



$\mathcal{T}_{Pet} \models MUTEX$ and $\mathcal{T}_{Pet} \models LIVE$

Peterson's mutual exclusion algorithm



$\mathcal{T}_{Pet} \models \mathit{MUTEX}$ and $\mathcal{T}_{Pet} \models \mathit{LIVE}$

LT properties and trace inclusion

An LT property over AP is a language E of infinite words over the alphabet $\Sigma = 2^{AP}$, i.e., $E \subseteq \left(2^{AP}\right)^{\omega}$.

If $\mathcal{T}$ is a TS over AP then $\mathcal{T} \models E$ iff $Traces(\mathcal{T}) \subseteq E$.

LT properties and trace inclusion

An LT property over AP is a language E of infinite words over the alphabet $\Sigma = 2^{AP}$, i.e., $E \subseteq \left(2^{AP}\right)^{\omega}$.

If $\mathcal{T}$ is a TS over AP then $\mathcal{T} \models E$ iff $Traces(\mathcal{T}) \subseteq E$.

Consequence of these definitions:

If $\mathcal{T}_1$ and $\mathcal{T}_2$ are TS over AP then for all LT properties E over AP:

$$Traces(\mathcal{T}_1) \subseteq Traces(\mathcal{T}_2) \wedge \mathcal{T}_2 \models E \Longrightarrow \mathcal{T}_1 \models E$$

LT properties and trace inclusion

An LT property over AP is a language E of infinite words over the alphabet $\Sigma = 2^{AP}$, i.e., $E \subseteq (2^{AP})^{\omega}$.

If $\mathcal{T}$ is a TS over AP then $\mathcal{T} \models E$ iff $Traces(\mathcal{T}) \subseteq E$.

Consequence of these definitions:

If $\mathcal{T}_1$ and $\mathcal{T}_2$ are TS over AP then for all LT properties E over AP:

$$Traces(\mathcal{T}_1) \subseteq Traces(\mathcal{T}_2) \wedge \mathcal{T}_2 \models E \implies \mathcal{T}_1 \models E$$

note: $Traces(\mathcal{T}_1) \subseteq Traces(\mathcal{T}_2) \subseteq E$

LT properties and trace inclusion

An LT property over AP is a language E of infinite words over the alphabet $\Sigma = 2^{AP}$, i.e., $E \subseteq \left(2^{AP}\right)^{\omega}$.

If $\mathcal{T}$ is a TS over AP then $\mathcal{T} \models E$ iff $Traces(\mathcal{T}) \subseteq E$.

If $\mathcal{T}_1$ and $\mathcal{T}_2$ are TS over AP then the following statements are equivalent:

(1) $Traces(\mathcal{T}_1) \subseteq Traces(\mathcal{T}_2)$

(2) for all LT-properties E over AP: whenever $\mathcal{T}_2 \models E$ then $\mathcal{T}_1 \models E$

LT properties and trace inclusion

An LT property over AP is a language E of infinite words over the alphabet $\Sigma = 2^{AP}$, i.e., $E \subseteq \left(2^{AP}\right)^{\omega}$.

If $\mathcal{T}$ is a TS over AP then $\mathcal{T} \models E$ iff $Traces(\mathcal{T}) \subseteq E$.

If $\mathcal{T}_1$ and $\mathcal{T}_2$ are TS over AP then the following statements are equivalent:

(1) $Traces(\mathcal{T}_1) \subseteq Traces(\mathcal{T}_2)$

(2) for all LT-properties E over AP: whenever $\mathcal{T}_2 \models E$ then $\mathcal{T}_1 \models E$

$(1) \Longrightarrow (2)$: $\surd$

LT properties and trace inclusion

An LT property over AP is a language E of infinite words over the alphabet $\Sigma = 2^{AP}$, i.e., $E \subseteq \left(2^{AP}\right)^{\omega}$.

If $\mathcal{T}$ is a TS over AP then $\mathcal{T} \models E$ iff $Traces(\mathcal{T}) \subseteq E$.

If $\mathcal{T}_1$ and $\mathcal{T}_2$ are TS over AP then the following statements are equivalent:

(1) $Traces(\mathcal{T}_1) \subseteq Traces(\mathcal{T}_2)$

(2) for all LT-properties E over AP: whenever $\mathcal{T}_2 \models E$ then $\mathcal{T}_1 \models E$

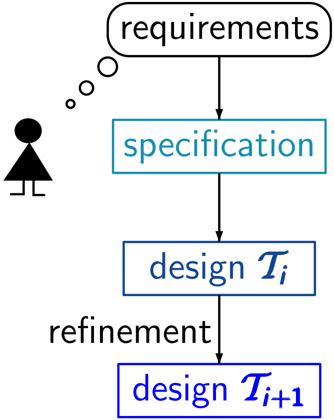
$(2) \Longrightarrow (1)$: consider $E = Traces(\mathcal{T}_2)$

Relevance of trace inclusion

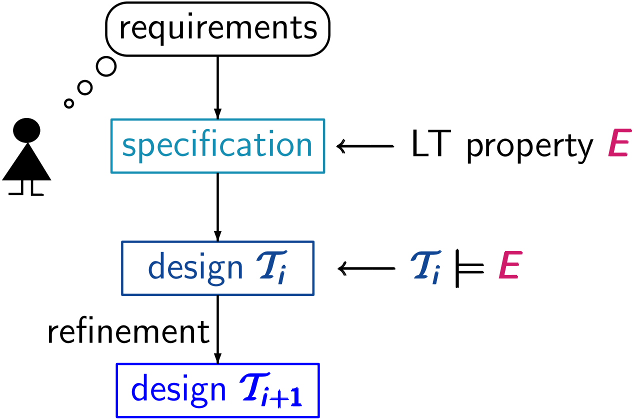
Trace inclusion appears naturally

- as an implementation/refinement relation
- when resolving nondeterminism
- in the context of abstractions

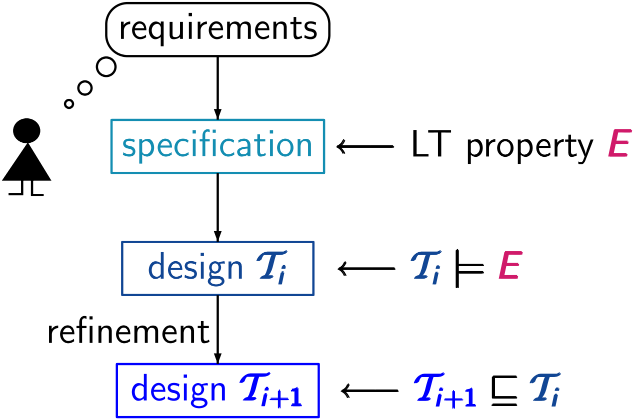
Software design cycle



Software design cycle



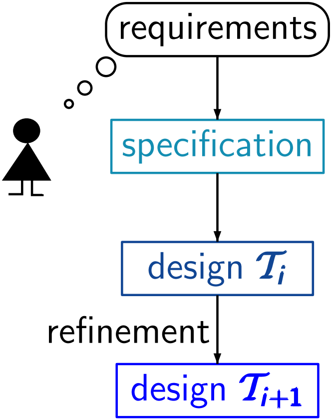
Software design cycle



implementation/refinement relation $\sqsubseteq$:

$\mathcal{T}_{i+1} \sqsubseteq \mathcal{T}_i$ iff "$\mathcal{T}_{i+1}$ correctly implements $\mathcal{T}_i$"

Trace inclusion as an implementation relation



$\longleftarrow \mathcal{T}_i \models E$

$\longleftarrow \mathcal{T}_{i+1} \sqsubseteq \mathcal{T}_i$

trace inclusion

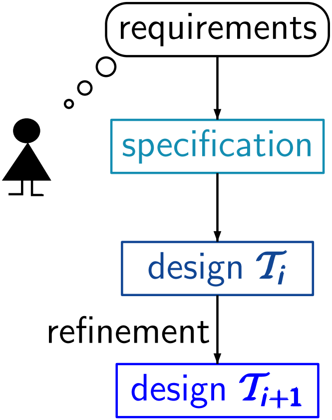
$\mathcal{T}_{i+1} \sqsubseteq \mathcal{T}_i$ iff

$Traces(\mathcal{T}_{i+1}) \subseteq Traces(\mathcal{T}_i)$

implementation/refinement relation $\sqsubseteq$:

$\mathcal{T}_{i+1} \sqsubseteq \mathcal{T}_i$ iff "$\mathcal{T}_{i+1}$ correctly implements $\mathcal{T}_i$"

Trace inclusion as an implementation relation



trace inclusion

$\mathcal{T}_{i+1} \sqsubseteq \mathcal{T}_i$ iff

$\mathit{Traces}(\mathcal{T}_{i+1}) \subseteq \mathit{Traces}(\mathcal{T}_i)$

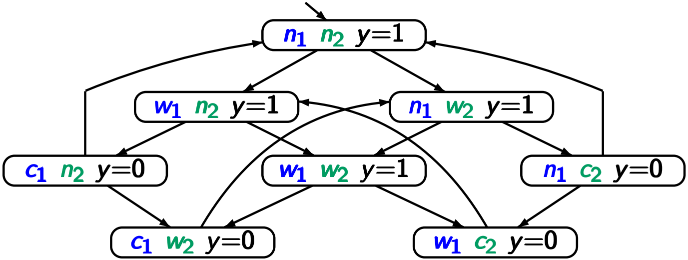
$\longleftarrow \mathcal{T}_i \models E$

$\longleftarrow \mathcal{T}_{i+1} \sqsubseteq \mathcal{T}_i$ implies $\mathcal{T}_{i+1} \models E$

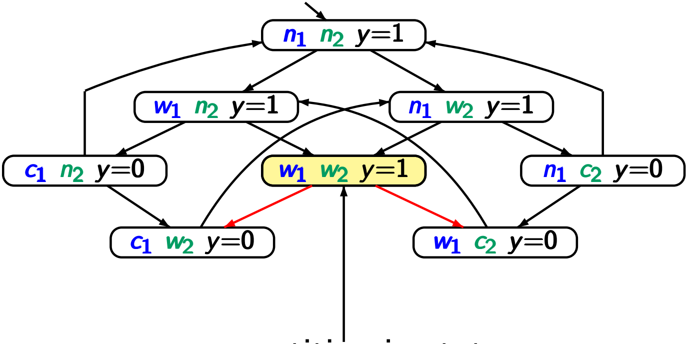
implementation/refinement relation $\sqsubseteq$:

$\mathcal{T}_{i+1} \sqsubseteq \mathcal{T}_i$ iff "$\mathcal{T}_{i+1}$ correctly implements $\mathcal{T}_i$"

Mutual exclusion with semaphore



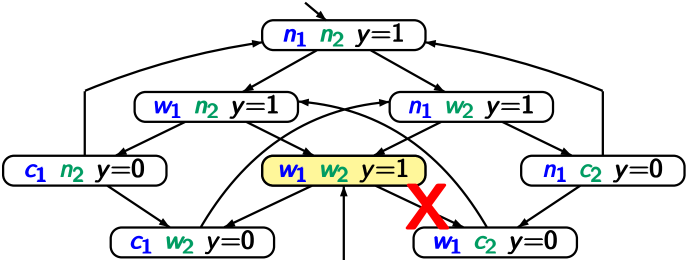
Mutual exclusion with semaphore



competition in state

$\langle wait_1\ wait_2\ y{=}1 \rangle$

Mutual exclusion with semaphore

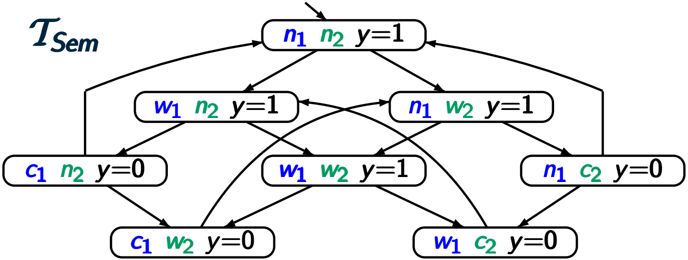


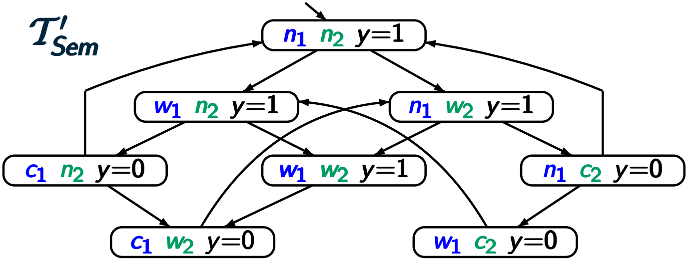
competition in state

$\langle wait_1\ wait_2\ y{=}1 \rangle$

resolve the nondeterminism by giving priority to process P_1

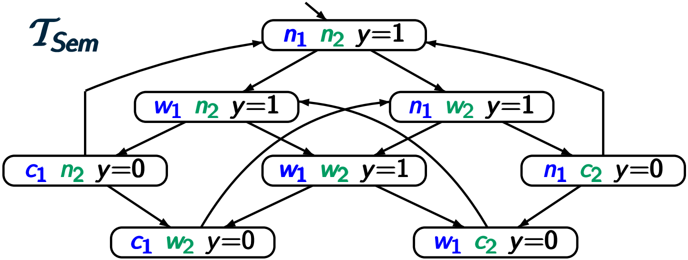
Mutual exclusion with semaphore



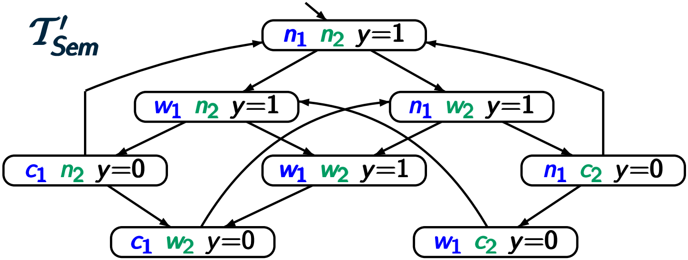


Mutual exclusion with semaphore

$\mathcal{T}_{Sem}$

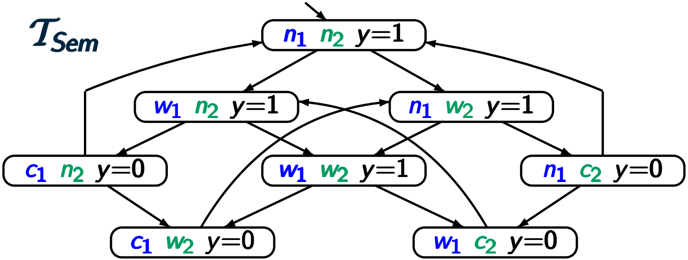


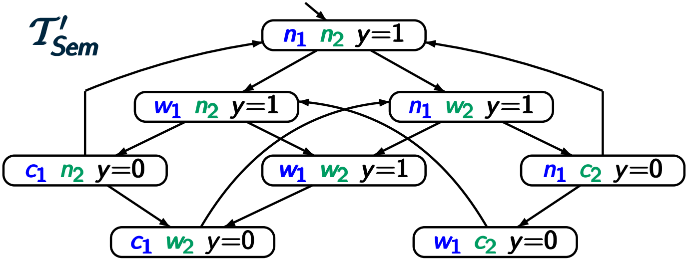
$\mathcal{T}'_{Sem}$



$$Paths(\mathcal{T}'_{Sem}) \subseteq Paths(\mathcal{T}_{Sem})$$

Mutual exclusion with semaphore

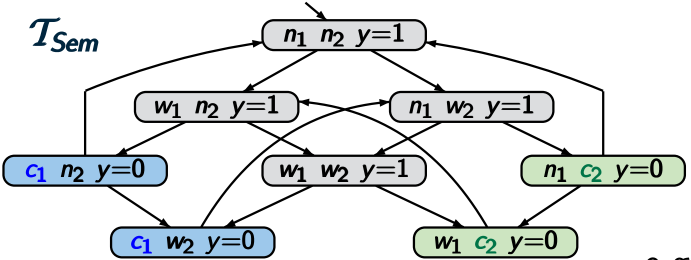




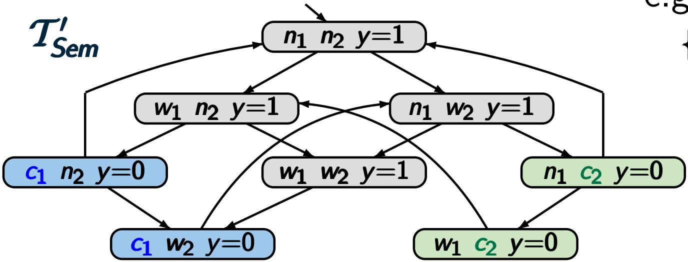
$Traces(\mathcal{T}'_{Sem}) \subseteq Traces(\mathcal{T}_{Sem})$ for any AP

Mutual exclusion with semaphore

$\mathcal{T}_{Sem}$



$\mathcal{T}'_{Sem}$



e.g., for $AP = \{crit_1, crit_2\}$

$$Traces(\mathcal{T}_{Sem}) \models E \text{ implies } Traces(\mathcal{T}'_{Sem}) \models E \text{ for any } E$$

Relevance of trace inclusion

Trace inclusion appears naturally

- as an implementation/refinement relation
- when resolving nondeterminism ⟵

  e.g., $\mathit{Traces}(\mathcal{T}'_{\mathit{Sem}}) \subseteq \mathit{Traces}(\mathcal{T}_{\mathit{Sem}})$
- in the context of abstractions

Relevance of trace inclusion

Trace inclusion appears naturally

- as an implementation/refinement relation
- when resolving nondeterminism

whenever $\mathcal{T}'$ results from $\mathcal{T}$ by a scheduling policy for resolving nondeterministic choices in $\mathcal{T}$ then

$$\mathit{Traces}(\mathcal{T}') \subseteq \mathit{Traces}(\mathcal{T})$$

- in the context of abstractions

Relevance of trace inclusion

Trace inclusion appears naturally

- as an implementation/refinement relation
- when resolving nondeterminism
- in the context of abstractions $\longleftarrow$

Trace inclusion and data abstraction


```
  ⋮
x:=7; y:=5;
WHILE x>0 DO
      x:=x−1;
      y:=y+1
OD
  ⋮
```

Trace inclusion and data abstraction

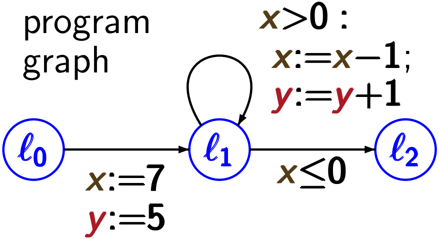

```
        ⋮
ℓ0   x:=7; y:=5;
ℓ1   WHILE x>0 DO
            x:=x−1;
            y:=y+1
     OD
ℓ2      ⋮
```

does $\ell_2 \wedge odd(y)$ never hold ?

Trace inclusion and data abstraction


```
        ⋮
ℓ0   x:=7;  y:=5;
ℓ1   WHILE x>0 DO
           x:=x−1;
           y:=y+1
     OD
ℓ2      ⋮
```

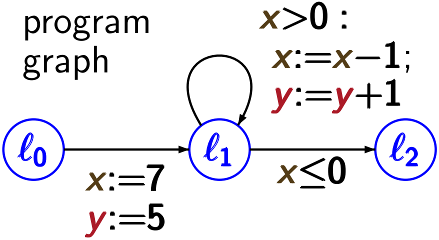
program graph



does $\ell_2 \wedge odd(y)$ never hold ?

```
        ⋮
ℓ0   x:=7;  y:=5;
ℓ1   WHILE x>0 DO
            x:=x−1;
            y:=y+1
     OD
ℓ2      ⋮
```

program graph



let $\mathcal{T}$ be the associated TS

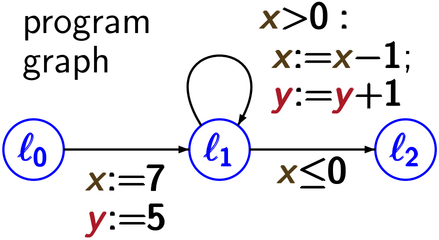
does $\ell_2 \wedge odd(y)$ never hold ?

$\longleftarrow$ $\mathcal{T} \models$ "never $\ell_2 \wedge odd(y)$" ?

Trace inclusion and data abstraction


```
        ⋮
ℓ0   x:=7;  y:=5;
ℓ1   WHILE x>0 DO
              x:=x−1;
              y:=y+1
     OD
ℓ2      ⋮
```

program graph



let $\mathcal{T}$ be the associated TS

does $\ell_2 \wedge odd(y)$ never hold ? $\longleftarrow$ $\mathcal{T} \models$ "never $\ell_2 \wedge odd(y)$" ?

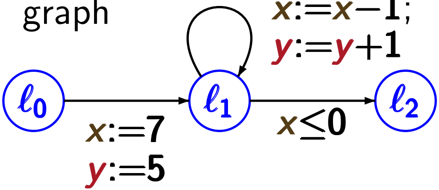
data abstraction w.r.t. the predicates

$x>0$, $x=0$, $x \equiv_2 y$

Trace inclusion and data abstraction


```
        ⋮
ℓ0   x:=7;  y:=5;
ℓ1   WHILE x>0 DO
           x:=x−1;
           y:=y+1
     OD
ℓ2      ⋮
```

program graph



let $\mathcal{T}$ be the associated TS

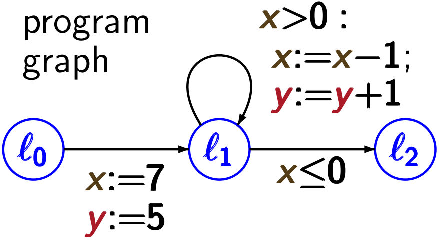
does $\ell_2 \wedge odd(y)$ never hold ? $\longleftarrow$ $\mathcal{T} \models$ "never $\ell_2 \wedge odd(y)$" ?

data abstraction w.r.t. the predicates

$x>0$, $x=0$, $x \equiv_2 y$ $\longleftarrow$ i.e., $x-y$ is even

```
      ⋮
ℓ0   x:=7;  y:=5;
ℓ1   WHILE x>0 DO
           x:=x−1;
           y:=y+1
     OD
ℓ2    ⋮
```

program graph

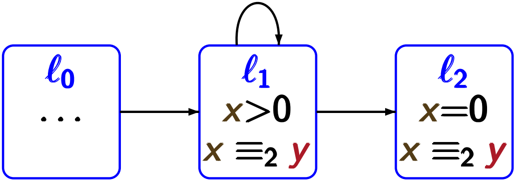


let $\mathcal{T}$ be the associated TS

does $\ell_2 \wedge odd(y)$ never hold ?

data abstraction w.r.t. the predicates

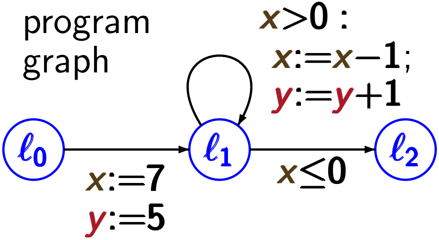
$x>0$, $x=0$, $x \equiv_2 y$



abstract transition system $\mathcal{T}'$

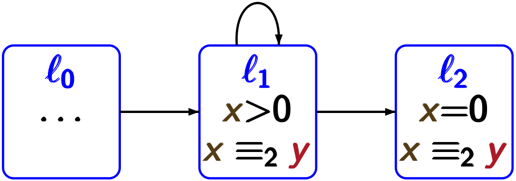
Trace inclusion and data abstraction

$$
\begin{array}{ll}
 & \vdots \\
\ell_0 & x{:=}7;\ \ y{:=}5; \\
\ell_1 & \texttt{WHILE}\ x{>}0\ \texttt{DO} \\
 & \qquad x{:=}x{-}1; \\
 & \qquad y{:=}y{+}1 \\
 & \texttt{OD} \\
\ell_2 & \vdots
\end{array}
$$



let $\mathcal{T}$ be the associated TS

does $\ell_2 \wedge odd(y)$
never hold ?



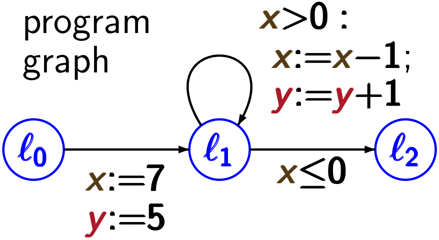
$\mathcal{T}' \models$ "never $\ell_2 \wedge odd(y)$"

data abstraction w.r.t.
the predicates
$x{>}0$, $x{=}0$, $x \equiv_2 y$

Trace inclusion and data abstraction

```
        ⋮
ℓ0   x:=7;  y:=5;
ℓ1   WHILE x>0 DO
           x:=x−1;
           y:=y+1
     OD
ℓ2      ⋮
```

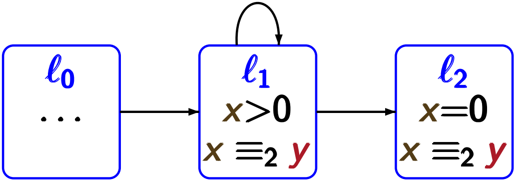
program graph



let $\mathcal{T}$ be the associated TS

does $\ell_2 \wedge odd(y)$ never hold ?

data abstraction w.r.t. the predicates $x>0$, $x=0$, $x \equiv_2 y$



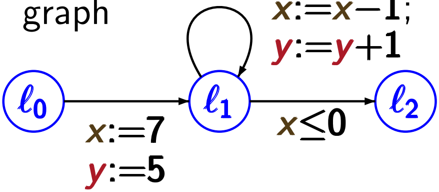
$\mathcal{T}' \models$ "never $\ell_2 \wedge odd(y)$"

$Traces(\mathcal{T}) \subseteq Traces(\mathcal{T}')$

Trace inclusion and data abstraction

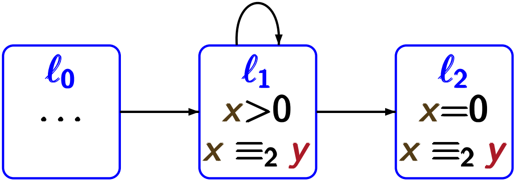

```
        ⋮
ℓ0   x:=7;  y:=5;
ℓ1   WHILE x>0 DO
          x:=x−1;
          y:=y+1
     OD
ℓ2      ⋮
```

program graph



let $\mathcal{T}$ be the associated TS

does $\ell_2 \wedge odd(y)$ never hold ?



$\mathcal{T} \models$ "never $\ell_2 \wedge odd(y)$" $\Leftarrow \begin{cases} \mathcal{T}' \models \text{"never } \ell_2 \wedge odd(y)\text{"} \\ Traces(\mathcal{T}) \subseteq Traces(\mathcal{T}') \end{cases}$

Trace equivalence

Trace equivalence

Transition systems $\mathcal{T}_1$ and $\mathcal{T}_2$ over the same set AP of atomic propositions are called trace equivalent iff

$$Traces(\mathcal{T}_1) = Traces(\mathcal{T}_2)$$

Trace equivalence

Transition systems $\mathcal{T}_1$ and $\mathcal{T}_2$ over the same set AP of atomic propositions are called trace equivalent iff

$$Traces(\mathcal{T}_1) = Traces(\mathcal{T}_2)$$

i.e., trace equivalence requires trace inclusion in both directions

Trace equivalence

Transition systems $\mathcal{T}_1$ and $\mathcal{T}_2$ over the same set AP of atomic propositions are called trace equivalent iff

$$Traces(\mathcal{T}_1) = Traces(\mathcal{T}_2)$$

i.e., trace equivalence requires trace inclusion in both directions

Trace equivalent TS satisfy the **same LT properties**

LT properties and trace relations

Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be TS over AP.

The following statements are equivalent:

(1) $Traces(\mathcal{T}_1) \subseteq Traces(\mathcal{T}_2)$

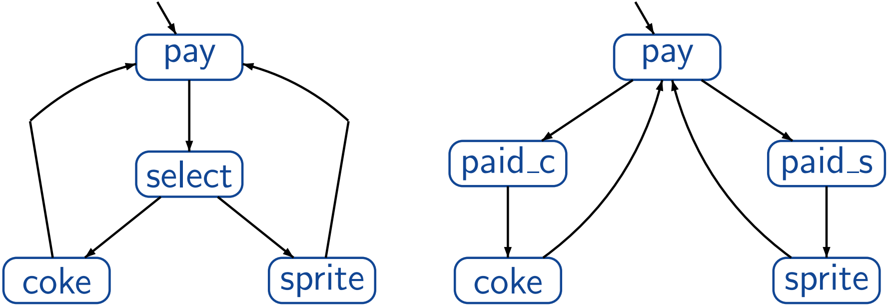
(2) for all LT-properties E: $\mathcal{T}_2 \models E \Longrightarrow \mathcal{T}_1 \models E$

The following statements are equivalent:

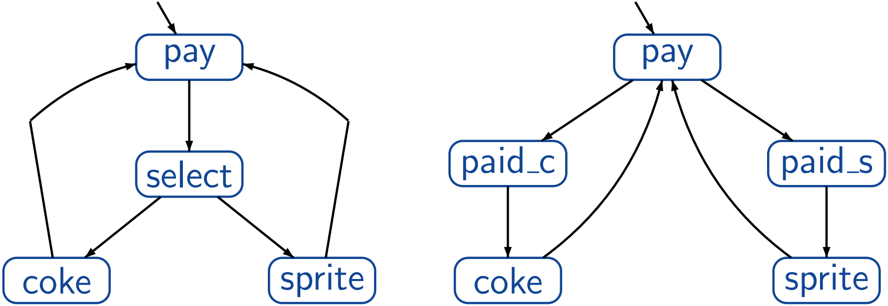
(1) $Traces(\mathcal{T}_1) = Traces(\mathcal{T}_2)$

(2) for all LT-properties E: $\mathcal{T}_1 \models E$ iff $\mathcal{T}_2 \models E$

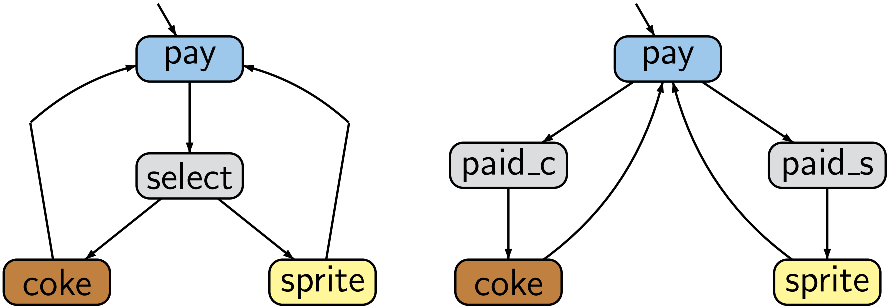
Trace equivalent beverage machines



Trace equivalent beverage machines

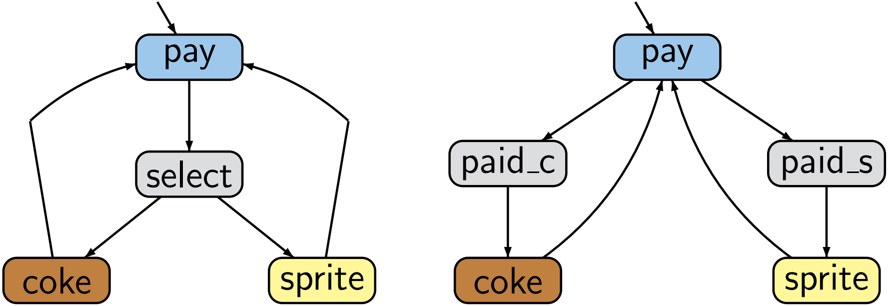


set of atomic propositions $AP = \{pay, coke, sprite\}$



set of atomic propositions $AP = \{\textit{pay}, \textit{coke}, \textit{sprite}\}$

Trace equivalent beverage machines



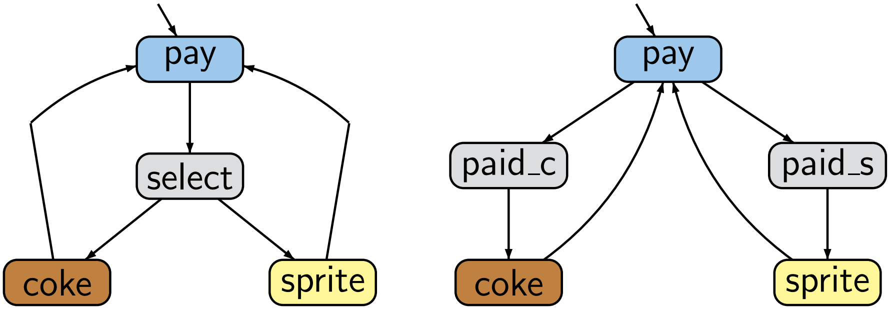
set of atomic propositions $AP = \{pay, coke, sprite\}$

$Traces(\mathcal{T}_1) = Traces(\mathcal{T}_2) =$ set of all infinite words

$\{pay\} \varnothing \{drink_1\} \{pay\} \varnothing \{drink_2\} \ldots$

where $drink_1, drink_2, \ldots \in \{coke, sprite\}$

Trace equivalent beverage machines



set of atomic propositions $AP = \{pay, coke, sprite\}$

$Traces(\mathcal{T}_1) = Traces(\mathcal{T}_2) =$ set of all infinite words

$\{pay\}\, \varnothing\, \{drink_1\}\, \{pay\}\, \varnothing\, \{drink_2\}\, \ldots$

$\mathcal{T}_1$ and $\mathcal{T}_2$ satisfy the same LT-properties over AP

Classification of LT-properties

safety properties *"nothing bad will happen"*

liveness properties *"something good will happen"*

Classification of LT-properties

safety properties *"nothing bad will happen"*

examples:

- mutual exclusion
- deadlock freedom
- "every red phase is preceded by a yellow phase"

liveness properties *"something good will happen"*

Classification of LT-properties

safety properties *"nothing bad will happen"*

examples:

- mutual exclusion
- deadlock freedom
- "every red phase is preceded by a yellow phase"

liveness properties *"something good will happen"*

examples:

- "each waiting process will eventually enter its critical section"
- "each philosopher will eat infinitely often"

Classification of LT-properties

safety properties *"nothing bad will happen"*

examples:

- mutual exclusion
- deadlock freedom

} special case: **invariants** *"no bad state will be reached"*

- "every red phase is preceded by a yellow phase"

liveness properties *"something good will happen"*

examples:

- "each waiting process will eventually enter its critical section"
- "each philosopher will eat infinitely often"

Propositional logic

$$\Phi \ ::= \ \mathit{true} \ \Big| \ a \ \Big| \ \Phi_1 \wedge \Phi_2 \ \Big| \ \neg\Phi$$

Propositional logic

$$\Phi ::= \mathit{true} \mid a \mid \Phi_1 \wedge \Phi_2 \mid \neg\Phi$$

atomic proposition, i.e., $a \in AP$

Propositional logic

$$\Phi \ ::= \ \textit{true} \ \big| \ a \ \big| \ \Phi_1 \wedge \Phi_2 \ \big| \ \neg\Phi \ \big| \ \Phi_1 \vee \Phi_2 \ \big| \ \Phi_1 \rightarrow \Phi_2 \ \big| \ \ldots$$

atomic proposition, i.e., $a \in AP$

Propositional logic

$$\Phi \ ::= \ \textit{true} \ \big| \ a \ \big| \ \Phi_1 \wedge \Phi_2 \ \big| \ \neg\Phi \ \big| \ \Phi_1 \vee \Phi_2 \ \big| \ \Phi_1 \rightarrow \Phi_2 \ \big| \ \ldots$$

atomic proposition, i.e., $a \in AP$

semantics: interpretation over a subsets of AP

$$\Phi \ ::= \ true \ \big| \ a \ \big| \ \Phi_1 \wedge \Phi_2 \ \big| \ \neg\Phi \ \big| \ \Phi_1 \vee \Phi_2 \ \big| \ \Phi_1 \rightarrow \Phi_2 \ \big| \ \ldots$$

↑ atomic proposition, i.e., $a \in AP$

semantics: Let $A \subseteq AP$

$A \models true$		
$A \models a$	iff	$a \in A$
$A \models \Phi_1 \wedge \Phi_2$	iff	$A \models \Phi_1$ and $A \models \Phi_2$
$A \models \neg\Phi$	iff	$A \not\models \Phi$

Propositional logic

$$\Phi \ ::= \ \textit{true} \ \Big| \ a \ \Big| \ \Phi_1 \wedge \Phi_2 \ \Big| \ \neg\Phi \ \Big| \ \Phi_1 \vee \Phi_2 \ \Big| \ \Phi_1 \rightarrow \Phi_2 \ \Big| \ \ldots$$

atomic proposition, i.e., $a \in AP$

semantics: Let $A \subseteq AP$

$A \models \textit{true}$		
$A \models a$	iff	$a \in A$
$A \models \Phi_1 \wedge \Phi_2$	iff	$A \models \Phi_1$ and $A \models \Phi_2$
$A \models \neg\Phi$	iff	$A \not\models \Phi$

e.g., $\{a, b\} \not\models (a \rightarrow \neg b) \vee c$ $\quad$ $\{a, b\} \models a \vee c$

Propositional logic

$$\Phi \ ::= \ \textit{true} \ \big| \ a \ \big| \ \Phi_1 \wedge \Phi_2 \ \big| \ \neg\Phi \ \big| \ \Phi_1 \vee \Phi_2 \ \big| \ \Phi_1 \rightarrow \Phi_2 \ \big| \ \ldots$$

atomic proposition, i.e., $a \in AP$

semantics: Let $A \subseteq AP$

$A \models \textit{true}$		
$A \models a$	iff	$a \in A$
$A \models \Phi_1 \wedge \Phi_2$	iff	$A \models \Phi_1$ and $A \models \Phi_2$
$A \models \neg\Phi$	iff	$A \not\models \Phi$

for state s of a TS over AP: $s \models \Phi$ iff $L(s) \models \Phi$

Invariant

Invariant

Let E be an LT property over AP.

E is called an invariant if there exists a propositional formula Φ over AP such that

$$E = \{ A_0 A_1 A_2 \ldots \in \left(2^{AP}\right)^\omega : \forall i \geq 0. A_i \models \Phi \}$$

Invariant

Let E be an LT property over AP.

E is called an invariant if there exists a propositional formula Φ over AP such that

$$E = \left\{ A_0 A_1 A_2 \ldots \in \left(2^{AP}\right)^{\omega} : \forall i \geq 0.\, A_i \models \Phi \right\}$$

Φ is called the invariant condition of E.

Examples for invariants

mutual exclusion (safety):

$$MUTEX = \begin{array}{l}\text{set of all infinite words } A_0\, A_1\, A_2 \ldots \text{ s.t.} \\ \forall i \in \mathbb{N}.\ \ \mathsf{crit}_1 \notin A_i \ \text{ or } \ \mathsf{crit}_2 \notin A_i\end{array}$$

here: $AP = \{\mathsf{crit}_1, \mathsf{crit}_2, \ldots\}$

mutual exclusion (safety):

$$MUTEX = \text{ set of all infinite words } A_0\,A_1\,A_2\ldots \text{ s.t. } \forall i \in \mathbb{N}.\ \ \text{crit}_1 \notin A_i \ \text{ or } \ \text{crit}_2 \notin A_i$$

invariant condition: $\Phi = \neg\text{crit}_1 \vee \neg\text{crit}_2$

here: $AP = \{\text{crit}_1, \text{crit}_2, \ldots\}$

Examples for invariants

mutual exclusion (safety):

$$MUTEX = \begin{array}{c}\text{set of all infinite words } A_0\, A_1\, A_2 \ldots \text{ s.t.}\\ \forall i \in \mathbb{N}.\ \ crit_1 \notin A_i \ \text{ or } \ crit_2 \notin A_i\end{array}$$

invariant condition: $\Phi = \neg crit_1 \vee \neg crit_2$

deadlock freedom for 5 dining philosophers:

$$DF = \begin{array}{c}\text{set of all infinite words } A_0\, A_1\, A_2 \ldots \text{ s.t.}\\ \forall i \in \mathbb{N}\ \exists \mathbf{j} \in \{0,1,2,3,4\}.\ wait_j \notin A_i\end{array}$$

here: $AP = \{wait_j : 0 \leq j \leq 4\} \cup \{\ldots\}$

Examples for invariants

mutual exclusion (safety):

$$\mathit{MUTEX} = \begin{array}{c}\text{set of all infinite words } A_0\,A_1\,A_2\ldots \text{ s.t.}\\ \forall i \in \mathbb{N}.\ \ \mathsf{crit}_1 \notin A_i \ \text{ or } \ \mathsf{crit}_2 \notin A_i\end{array}$$

invariant condition: $\Phi = \neg\mathsf{crit}_1 \vee \neg\mathsf{crit}_2$

deadlock freedom for 5 dining philosophers:

$$\mathit{DF} = \begin{array}{c}\text{set of all infinite words } A_0\,A_1\,A_2\ldots \text{ s.t.}\\ \forall i \in \mathbb{N}\ \exists j \in \{0,1,2,3,4\}.\ \mathsf{wait}_j \notin A_i\end{array}$$

invariant condition: **?**

here: $AP = \{\mathsf{wait}_j : 0 \leq j \leq 4\} \cup \{\ldots\}$

Examples for invariants

mutual exclusion (safety):

$$\textit{MUTEX} = \begin{array}{c} \text{set of all infinite words } A_0\, A_1\, A_2 \ldots \text{ s.t.} \\ \forall i \in \mathbb{N}.\ \ \mathsf{crit}_1 \notin A_i \ \text{ or } \ \mathsf{crit}_2 \notin A_i \end{array}$$

invariant condition: $\Phi = \neg\mathsf{crit}_1 \vee \neg\mathsf{crit}_2$

deadlock freedom for 5 dining philosophers:

$$\textit{DF} = \begin{array}{c} \text{set of all infinite words } A_0\, A_1\, A_2 \ldots \text{ s.t.} \\ \forall i \in \mathbb{N}\ \exists j \in \{0, 1, 2, 3, 4\}.\ \mathsf{wait}_j \notin A_i \end{array}$$

invariant condition:

$$\Phi = \neg\mathsf{wait}_0 \vee \neg\mathsf{wait}_1 \vee \neg\mathsf{wait}_2 \vee \neg\mathsf{wait}_3 \vee \neg\mathsf{wait}_4$$

here: $AP = \{\mathsf{wait}_j : 0 \leq j \leq 4\} \cup \{\ldots\}$

Satisfaction of invariants

Let E be an LT property over AP. E is called an invariant if there exists a propositional formula Φ s.t.

$$E = \{ A_0 A_1 A_2 \ldots \in \left(2^{AP}\right)^{\omega} : \forall i \geq 0.\, A_i \models \Phi \}$$

Satisfaction of invariants

Let E be an LT property over AP. E is called an invariant if there exists a propositional formula Φ s.t.

$$E = \{ A_0 A_1 A_2 \ldots \in \left(2^{AP}\right)^{\omega} : \forall i \geq 0.\, A_i \models \Phi \}$$

Let $\mathcal{T}$ be a TS over AP without terminal states. Then:

$\mathcal{T} \models E$ iff $trace(\pi) \in E$ for all $\pi \in Paths(\mathcal{T})$

Satisfaction of invariants

Let E be an LT property over AP. E is called an invariant if there exists a propositional formula Φ s.t.

$$E = \left\{ A_0 A_1 A_2 \ldots \in \left(2^{AP}\right)^{\omega} : \forall i \geq 0.\, A_i \models \Phi \right\}$$

Let $\mathcal{T}$ be a TS over AP without terminal states. Then:

$\mathcal{T} \models E$ iff $trace(\pi) \in E$ for all $\pi \in Paths(\mathcal{T})$

iff $s \models \Phi$ for all states s on a path of $\mathcal{T}$

Let E be an LT property over AP. E is called an invariant if there exists a propositional formula Φ s.t.

$$E = \{ A_0 A_1 A_2 \ldots \in \left(2^{AP}\right)^\omega : \forall i \geq 0.\, A_i \models \Phi \}$$

Let $\mathcal{T}$ be a TS over AP without terminal states. Then:

$\mathcal{T} \models E$ iff $trace(\pi) \in E$ for all $\pi \in Paths(\mathcal{T})$

iff $s \models \Phi$ for all states s on a path of $\mathcal{T}$

iff $s \models \Phi$ for all states $s \in Reach(\mathcal{T})$

↑ set of reachable states in $\mathcal{T}$

Satisfaction of invariants

Let E be an LT property over AP. E is called an invariant if there exists a propositional formula Φ s.t.

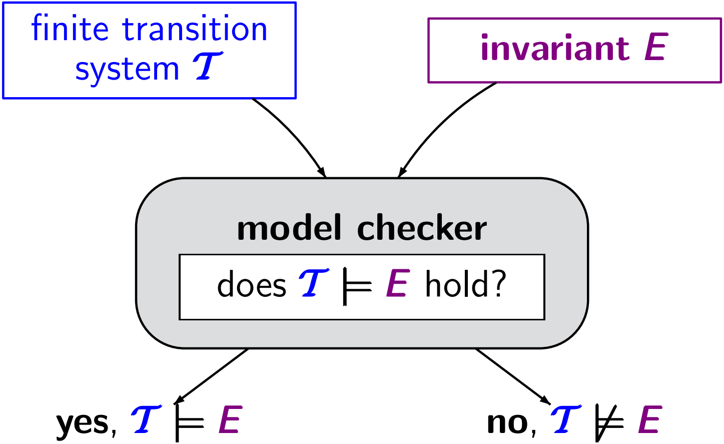
$$E = \left\{ A_0 A_1 A_2 \ldots \in \left(2^{AP}\right)^{\omega} : \forall i \geq 0.\, A_i \models \Phi \right\}$$

Let $\mathcal{T}$ be a TS over AP without terminal states. Then:

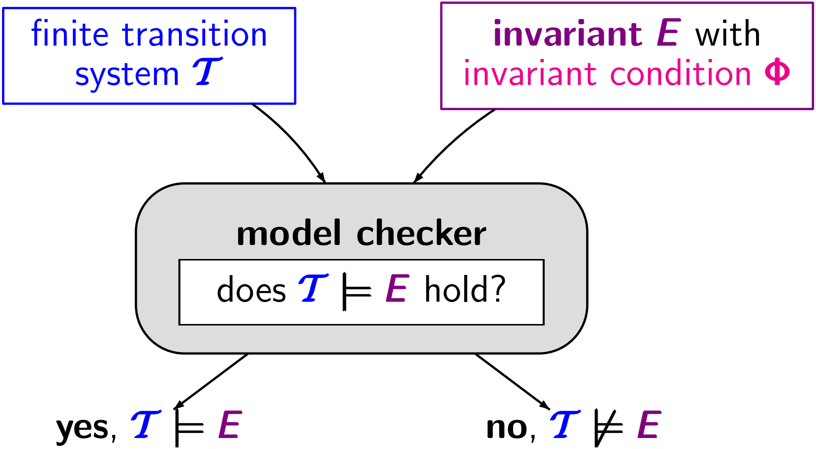
$$\begin{array}{lll} \mathcal{T} \models E & \text{iff} & trace(\pi) \in E \text{ for all } \pi \in Paths(\mathcal{T}) \\ & \text{iff} & s \models \Phi \text{ for all states } s \text{ on a path of } \mathcal{T} \\ & \text{iff} & s \models \Phi \text{ for all states } s \in Reach(\mathcal{T}) \end{array}$$

i.e., Φ holds in all initial states and is invariant under all transitions

Invariant checking

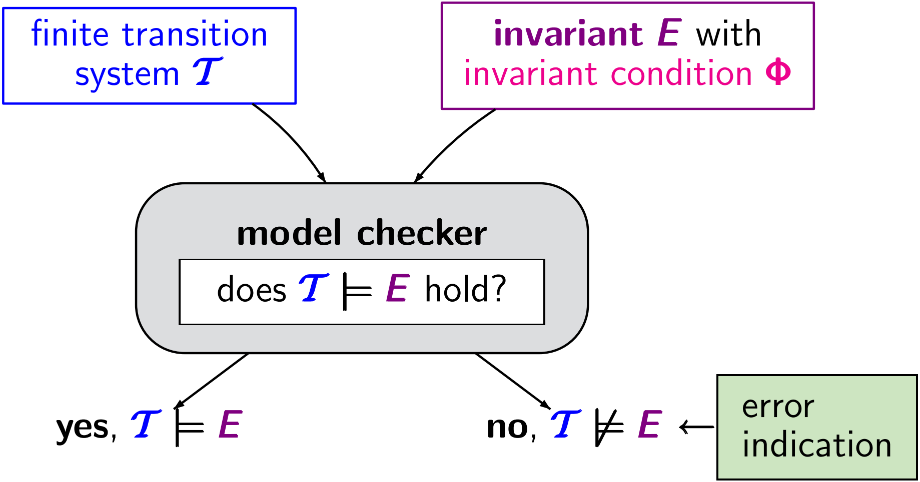


Invariant checking



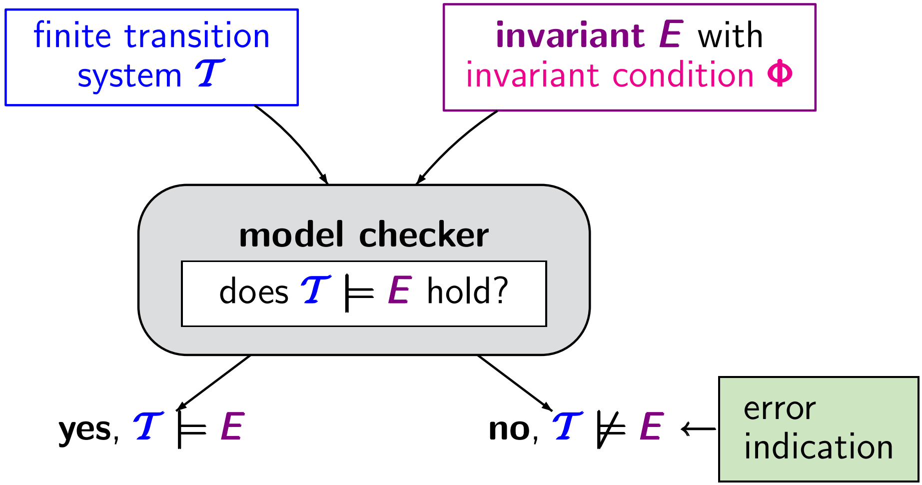
perform a graph analysis (**DFS** or **BFS**) to check whether $s \models \Phi$ for all $s \in Reach(\mathcal{T})$

Invariant checking



perform a graph analysis (**DFS** or **BFS**) to check whether $s \models \Phi$ for all $s \in Reach(\mathcal{T})$

Invariant checking



error indication: initial path fragment $s_0\, s_1 \ldots s_{n-1} s_n$
such that $s_i \models \Phi$ for $0 \leq i < n$ and $s_n \not\models \Phi$

DFS-based invariant checking

input: finite transition system $\mathcal{T}$, invariant condition Φ

DFS-based invariant checking

input: finite transition system $\mathcal{T}$, invariant condition Φ

```
FOR ALL s_0 ∈ S_0 DO
      IF DFS(s_0, Φ) THEN
            return "no"
      FI
OD
return "yes"
```

DFS-based invariant checking

input: finite transition system $\mathcal{T}$, invariant condition Φ

```
FOR ALL s_0 ∈ S_0 DO
      IF DFS(s_0, Φ) THEN
            return "no"
      FI
OD
return "yes"
```

$DFS(s_0, \Phi)$ returns "true" iff depth-first search from state s_0 leads to some state t with $t \not\models \Phi$

DFS-based invariant checking

input: finite transition system $\mathcal{T}$, invariant condition Φ

```
π := ∅  ⟵  stack for error indication
FOR ALL s0 ∈ S0 DO
      IF DFS(s0, Φ) THEN
           return "no" and reverse(π)
      FI
OD
return "yes"
```

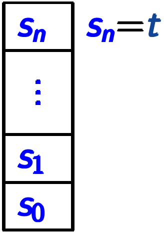
$DFS(s_0, \Phi)$ returns "true" iff depth-first search from state s_0 leads to some state t with $t \not\models \Phi$

DFS-based invariant checking

input: finite transition system $\mathcal{T}$, invariant condition Φ

$\pi := \varnothing$ ⟵ stack for error indication

```
FOR ALL s0 ∈ S0 DO
      IF DFS(s0, Φ) THEN
            return "no" and reverse(π)
      FI
OD
return "yes"
```



$DFS(s_0, \Phi)$ returns "true" iff depth-first search from state s_0 leads to some state t with $t \not\models \Phi$

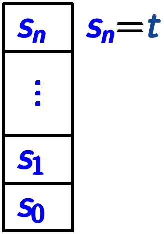
DFS-based invariant checking

input: finite transition system $\mathcal{T}$, invariant condition Φ

$U := \varnothing$ ⟵ stores the "processed" states

$\pi := \varnothing$ ⟵ stack for error indication

```
FOR ALL s0 ∈ S0 DO
      IF DFS(s0, Φ) THEN
            return "no" and reverse(π)
      FI
OD
return "yes"
```



$DFS(s_0, \Phi)$ returns "true" iff depth-first search from state s_0 leads to some state t with $t \not\models \Phi$

Recursive algorithm $DFS(s, \Phi)$

"searches" for a path fragment $s \ldots t$ with $t \not\models \Phi$

Recursive algorithm $DFS(s, \Phi)$

"searches" for a path fragment $s \ldots t$ with $t \not\models \Phi$

```
IF s ∉ U THEN
    IF s ⊭ Φ THEN return "true" FI
    IF s ⊨ Φ THEN

          ⋮

    FI
FI
return "false"
```

Recursive algorithm $DFS(s, \Phi)$

"searches" for a path fragment $s \ldots t$ with $t \not\models \Phi$

```
IF s ∉ U THEN
     IF s ⊭ Φ THEN return "true" FI
     IF s ⊨ Φ THEN
          insert s in U;



     FI
FI
return "false"
```

Recursive algorithm $DFS(s, \Phi)$

"searches" for a path fragment $s \ldots t$ with $t \not\models \Phi$

```
IF s ∉ U THEN
    IF s ⊭ Φ THEN return "true" FI
    IF s ⊨ Φ THEN
        insert s in U;
        FOR ALL s' ∈ Post(s) DO
            IF DFS(s', Φ) THEN
                return "true" FI
        OD
    FI
FI
return "false"
```

Recursive algorithm $DFS(s, \Phi)$

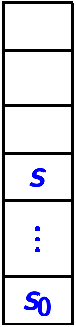
"searches" for a path fragment $s \ldots t$ with $t \not\models \Phi$

```
Push(π, s);
IF s ∉ U THEN
    IF s ⊭ Φ THEN return "true" FI
    IF s ⊨ Φ THEN
        insert s in U;
        FOR ALL s' ∈ Post(s) DO
            IF DFS(s', Φ) THEN
                return "true" FI
        OD
    FI
FI
Pop(π); return "false"
```

Recursive algorithm $DFS(s, \Phi)$

"searches" for a path fragment $s \ldots t$ with $t \not\models \Phi$

$Push(\pi, s)$;
IF $s \notin U$ THEN
  IF $s \not\models \Phi$ THEN return "true" FI
  IF $s \models \Phi$ THEN
    insert s in U;
    FOR ALL $s' \in Post(s)$ DO
      IF $DFS(s', \Phi)$ THEN
        return "true" FI
    OD
  FI
FI
$Pop(\pi)$; return "false"

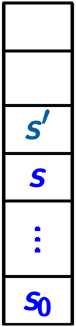


initial state

Recursive algorithm $DFS(s, \Phi)$

"searches" for a path fragment $s \ldots t$ with $t \not\models \Phi$

```
Push(π, s);
IF s ∉ U THEN
     IF s ⊭ Φ THEN return "true" FI
     IF s ⊨ Φ THEN
          insert s in U;
          FOR ALL s' ∈ Post(s) DO
               IF DFS(s', Φ) THEN
                    return "true" FI
          OD
     FI
FI
Pop(π); return "false"
```

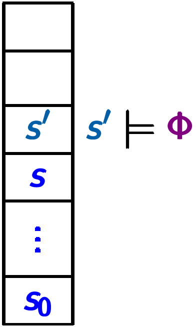


initial state

Recursive algorithm $DFS(s, \Phi)$

"searches" for a path fragment $s \ldots t$ with $t \not\models \Phi$

$Push(\pi, s)$;
IF $s \notin U$ THEN
    IF $s \not\models \Phi$ THEN return "true" FI
    IF $s \models \Phi$ THEN
        insert s in U;
        FOR ALL $s' \in Post(s)$ DO
            IF $DFS(s', \Phi)$ THEN
                return "true" FI
        OD
    FI
FI
$Pop(\pi)$; return "false"

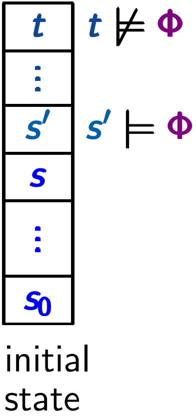


initial state

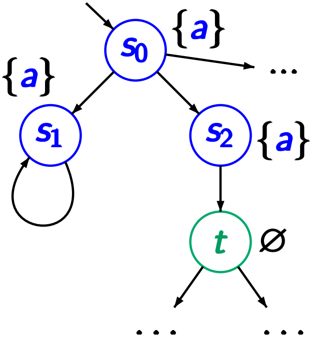
Recursive algorithm $DFS(s, \Phi)$

"searches" for a path fragment $s \ldots s' \ldots t$ with $t \not\models \Phi$

$Push(\pi, s)$;
IF $s \notin U$ THEN
    IF $s \not\models \Phi$ THEN return "true" FI
    IF $s \models \Phi$ THEN
        insert s in U;
        FOR ALL $s' \in Post(s)$ DO
            IF $DFS(s', \Phi)$ THEN
                return "true" FI
        OD
    FI
FI
$Pop(\pi)$; return "false"



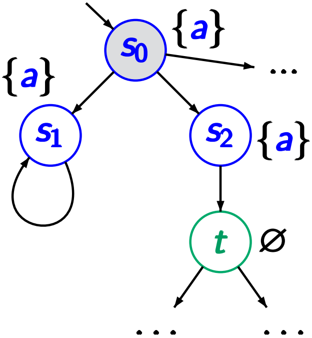
Example: invariant checking



invariant
condition a

$$s_0, s_1, s_2 \models a$$
$$t \not\models a$$

Example: invariant checking



invariant
condition a

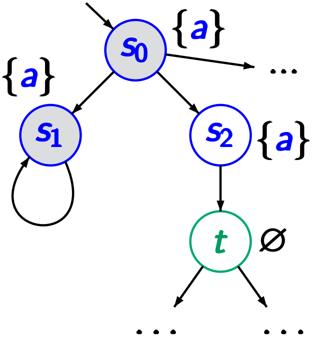
$s_0, s_1, s_2 \models a$
$t \not\models a$

$DFS(s_0, a)$

stack π

s_0

Example: invariant checking



invariant condition a

$$s_0, s_1, s_2 \models a$$
$$t \not\models a$$

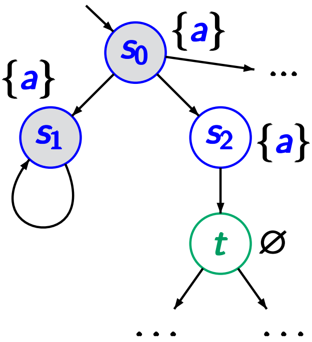
$DFS(s_0, a)$

$DFS(s_1, a)$

stack π

s_1
s_0

Example: invariant checking

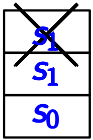


$DFS(s_0, a)$

$DFS(s_1, a)$

$DFS(s_1, a)$

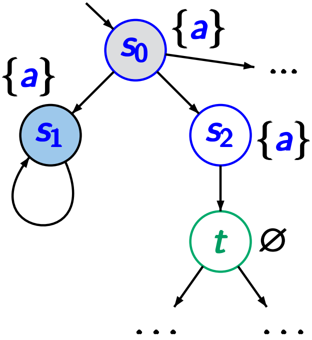
stack π



invariant
condition a

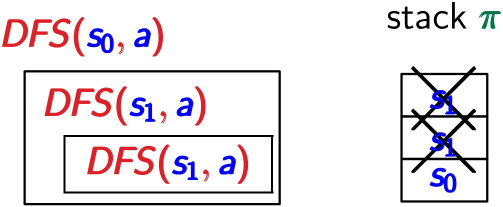
$$s_0, s_1, s_2 \models a$$
$$t \not\models a$$

Example: invariant checking

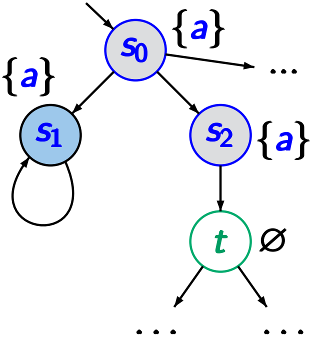


invariant condition a

$$s_0, s_1, s_2 \models a$$
$$t \not\models a$$



Example: invariant checking



invariant
condition a

$$s_0, s_1, s_2 \models a$$
$$t \not\models a$$

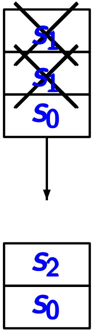
$DFS(s_0, a)$

$DFS(s_1, a)$

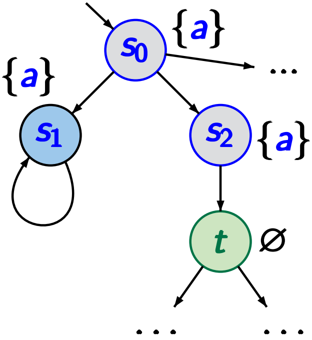
$DFS(s_1, a)$

$DFS(s_2, a)$

stack π



Example: invariant checking



invariant
condition a

$s_0, s_1, s_2 \models a$
$t \not\models a$

$DFS(s_0, a)$

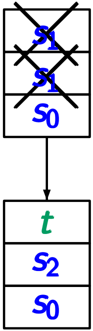
$DFS(s_1, a)$

$DFS(s_1, a)$

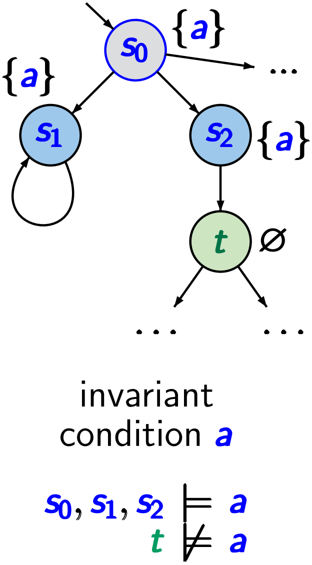
$DFS(s_2, a)$

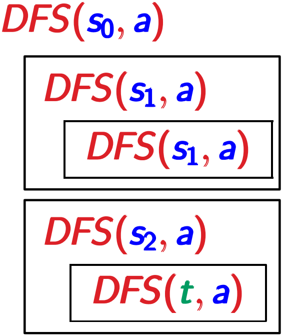
$DFS(t, a)$

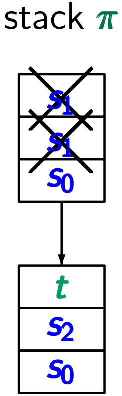
stack π



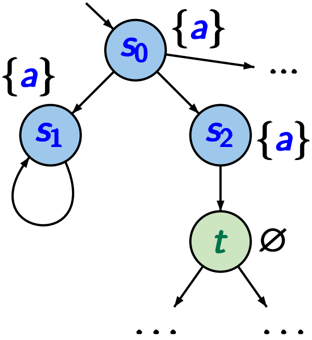
Example: invariant checking





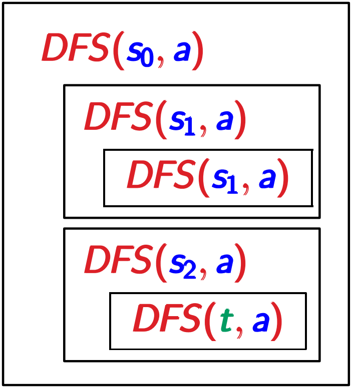


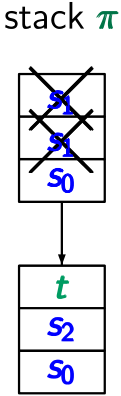
Example: invariant checking



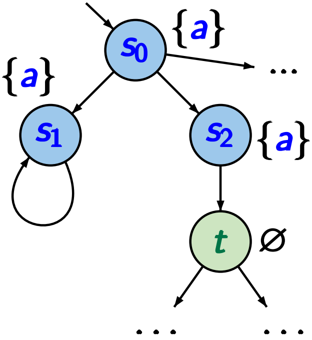
invariant condition a

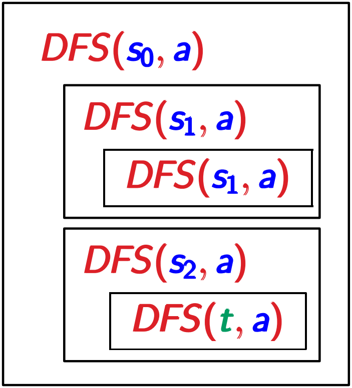
$$s_0, s_1, s_2 \models a$$
$$t \not\models a$$

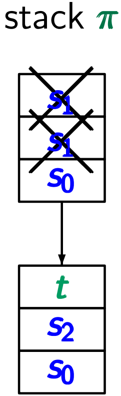




Example: invariant checking





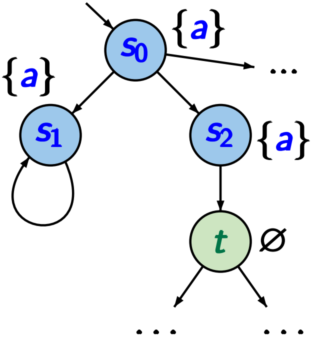


invariant condition a

$$s_0, s_1, s_2 \models a$$
$$t \not\models a$$

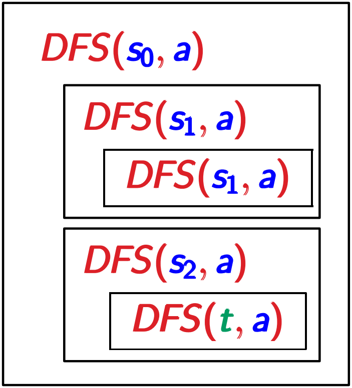
$s_0 \not\models$ "always a"

Example: invariant checking

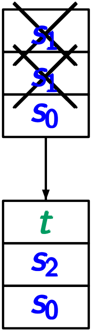


invariant condition a

$$s_0, s_1, s_2 \models a$$
$$t \not\models a$$



stack π



$s_0 \not\models$ "always a" $\longleftarrow$ error indication: $s_0\ s_2\ t$