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

↓ 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

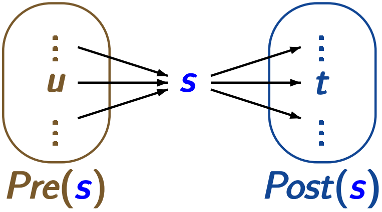
state graph $G_{\mathcal{T}}$

- 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 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

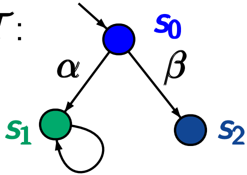
path of state s $\widehat{=}$ maximal path fragment starting in state s

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

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

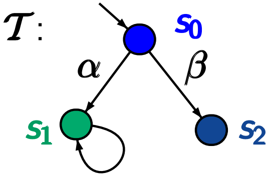
Paths of a TS

$\mathcal{T}$:



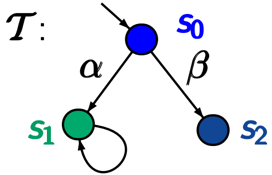
How many paths are there in $\mathcal{T}$?

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 $s_0\, s_2$

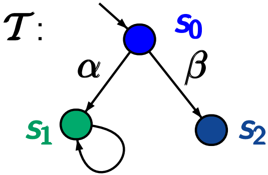


How many paths are there in $\mathcal{T}$?

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

$$
\begin{aligned}
\mathit{Paths}(s_1) &= \text{set of all maximal paths fragments starting in } s_1 \\
&= \{ s_1^\omega \} \text{ where } s_1^\omega = s_1\, s_1\, s_1\, s_1 \ldots
\end{aligned}
$$

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

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

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

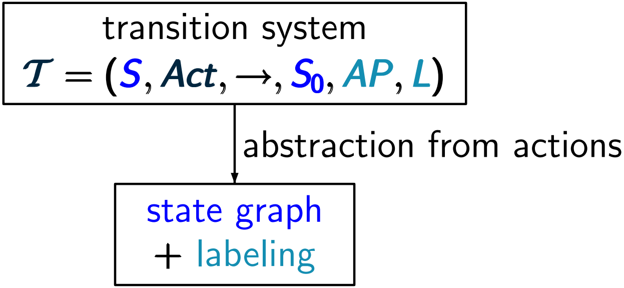
Equivalences and Abstraction

Linear-time vs branching-time

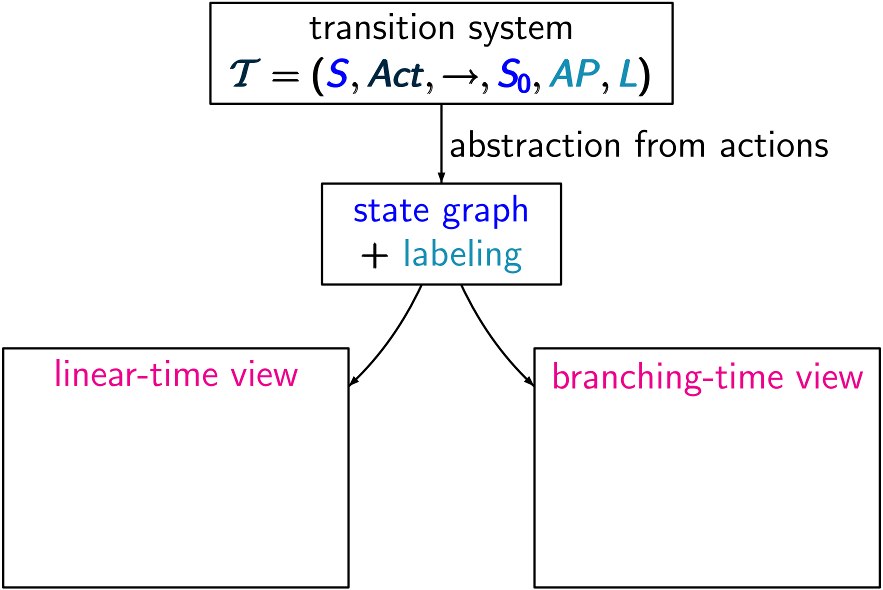
transition system

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

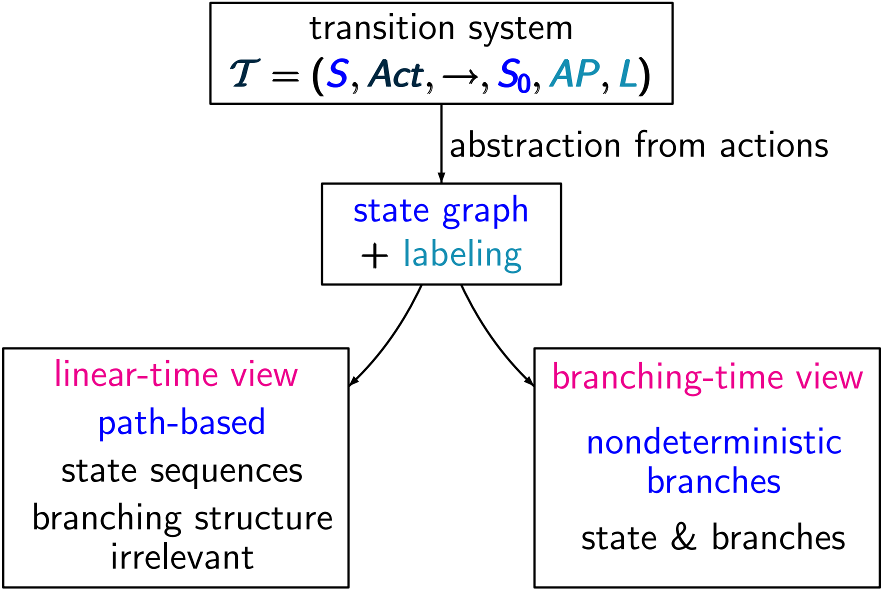
Linear-time vs branching-time



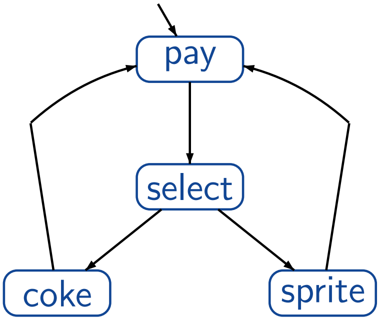
Linear-time vs branching-time



Linear-time vs branching-time

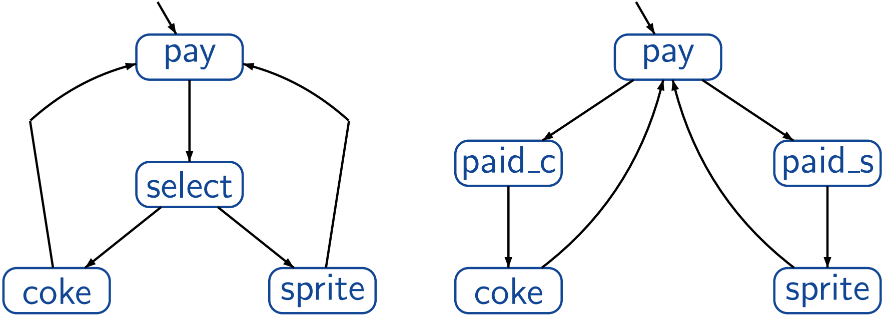


Example: vending machine



vending machine with
1 coin deposit
select drink after
having paid

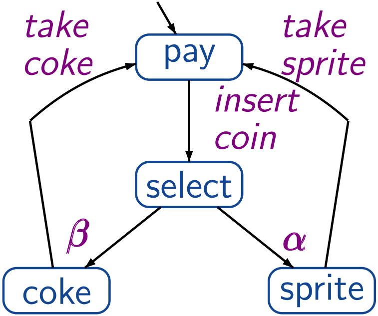
Example: vending machine

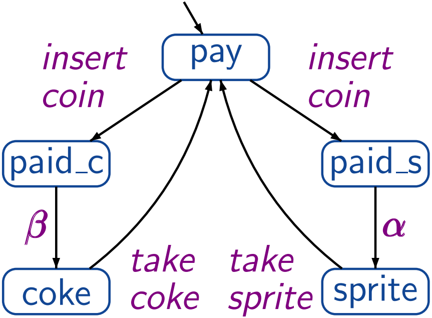


vending machine with
1 coin deposit
select drink after
having paid

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

Example: vending machine

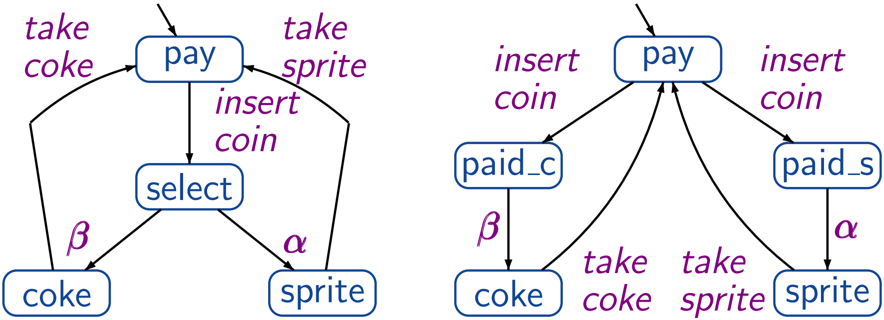




vending machine with
1 coin deposit
select drink after having paid

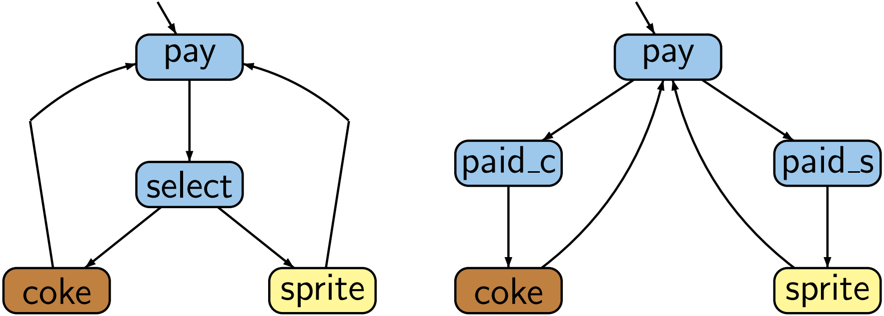
vending machine with
2 coin deposits
select drink by inserting the coin

Example: vending machine



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

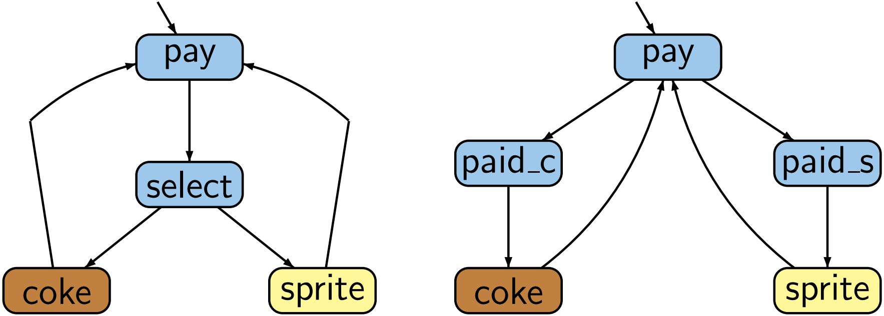
Example: vending machine



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

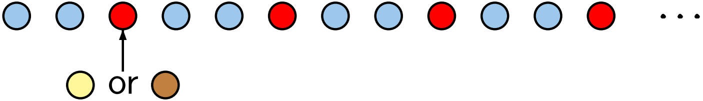
e.g., $L(\text{coke}) = \{coke\}$, $L(\text{pay}) = \varnothing$

Example: vending machine

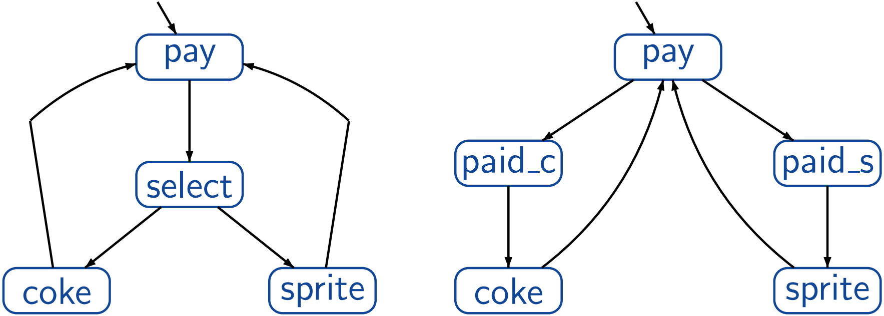


state based view: abstracts from actions and projects onto atomic propositions, e.g. $AP = \{coke, sprite\}$

linear time: all observable behaviors are of the form

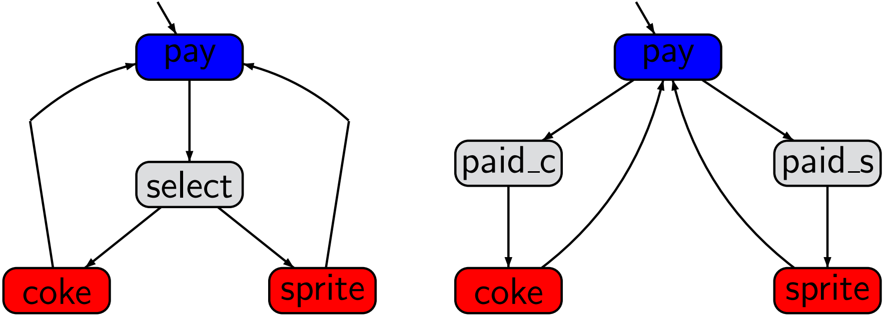


Example: vending machine



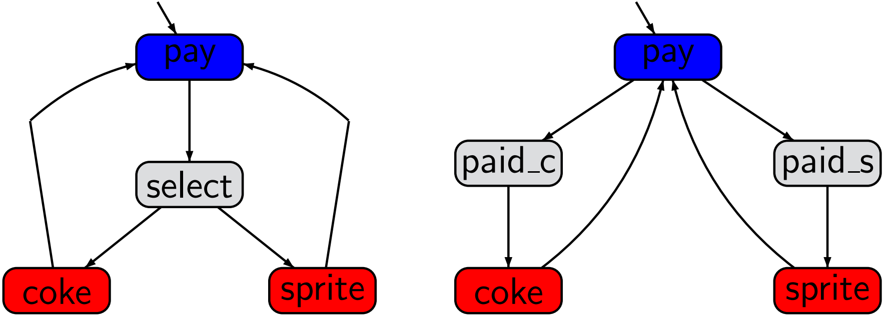
state based view: abstracts from actions and projects on atomc propositions, e.g., $AP = \{pay, drink\}$

Example: vending machine



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

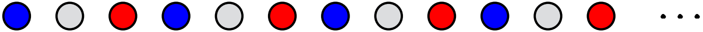
Example: vending machine



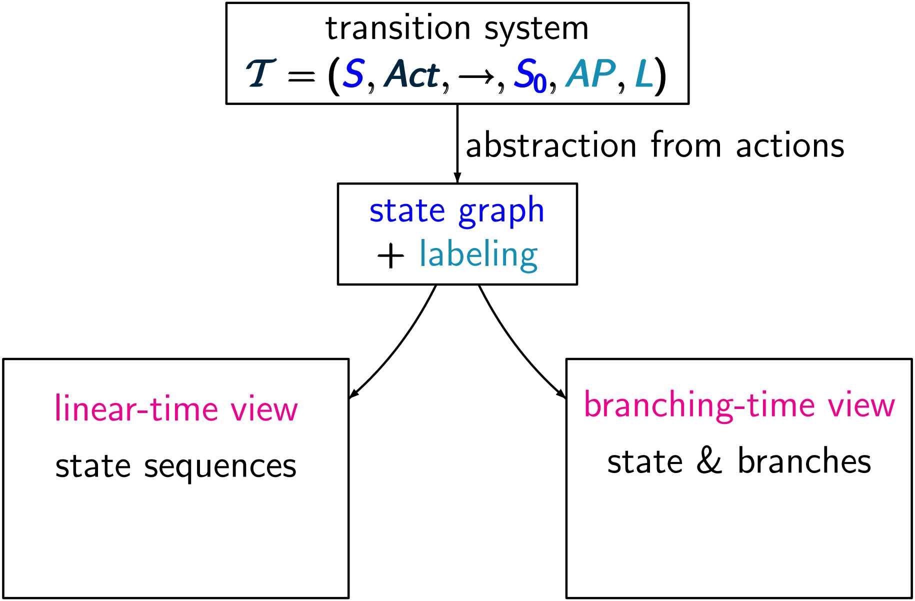
state based view: abstracts from actions and projects on atomc propositions, e.g., $AP = \{pay, drink\}$

linear & branching time:

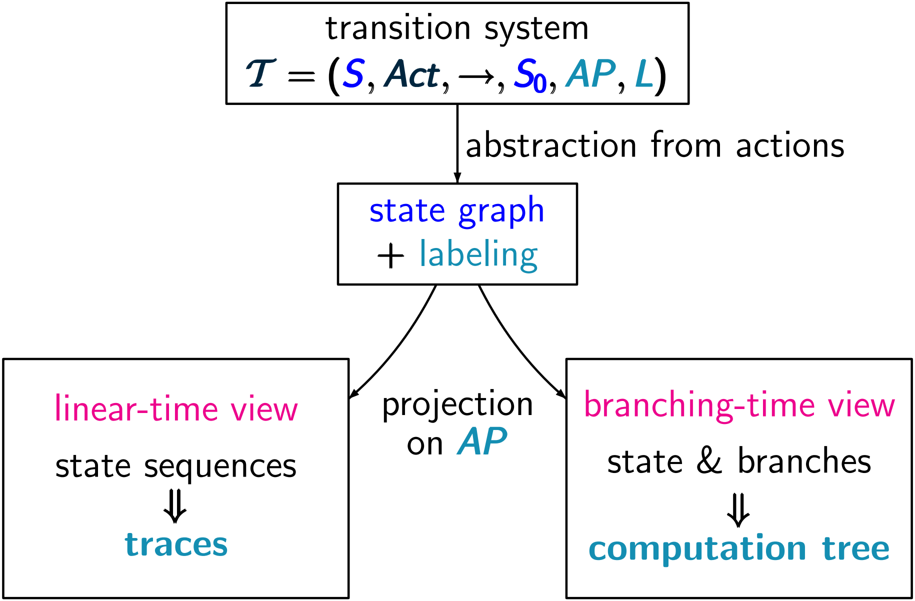
all observable behaviors have the form



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 $L : S \rightarrow 2^{AP}$

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

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

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:

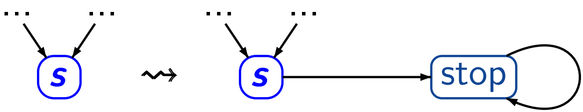
Treatment of terminal states

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:



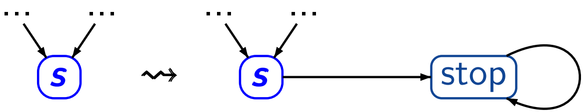
Treatment of terminal states

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

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

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

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

↑ initial, maximal path fragment

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

↑ 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

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

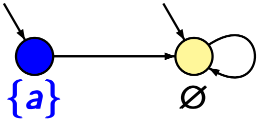
↑ initial, finite path fragment

Example: traces

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

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

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



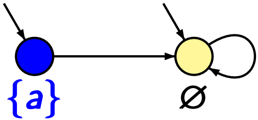
TS $\mathcal{T}$ with a single atomic proposition a

Example: traces

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

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

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

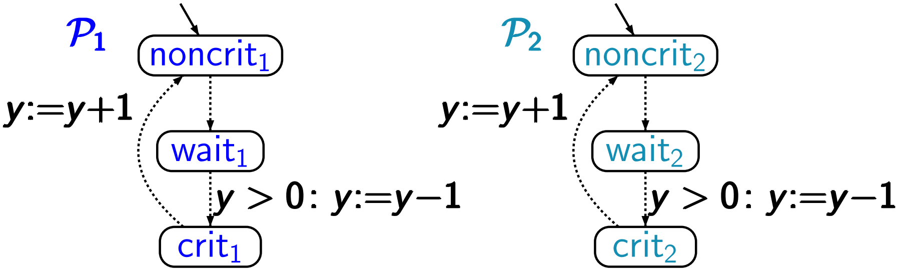


TS $\mathcal{T}$ with a single atomic proposition a

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

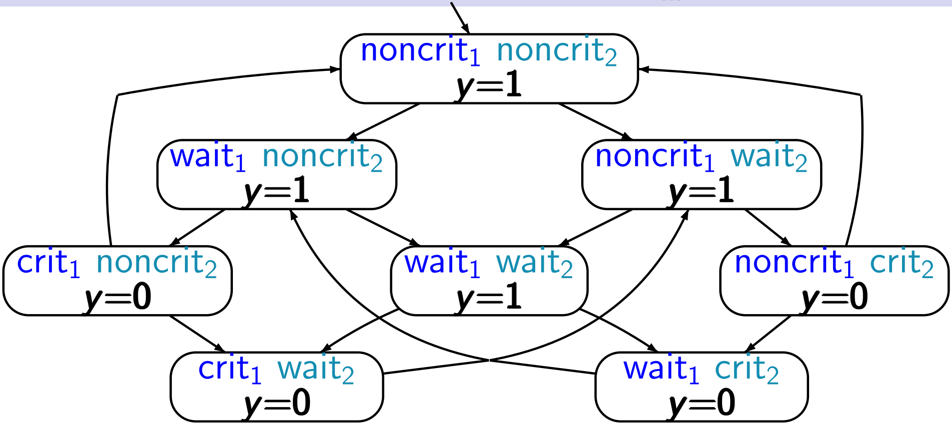
$$Traces_{fin}(\mathcal{T}) = \{\{a\}\varnothing^{n} : n \geq 0\} \cup \{\varnothing^{m} : m \geq 1\}$$

Mutual exclusion with semaphore



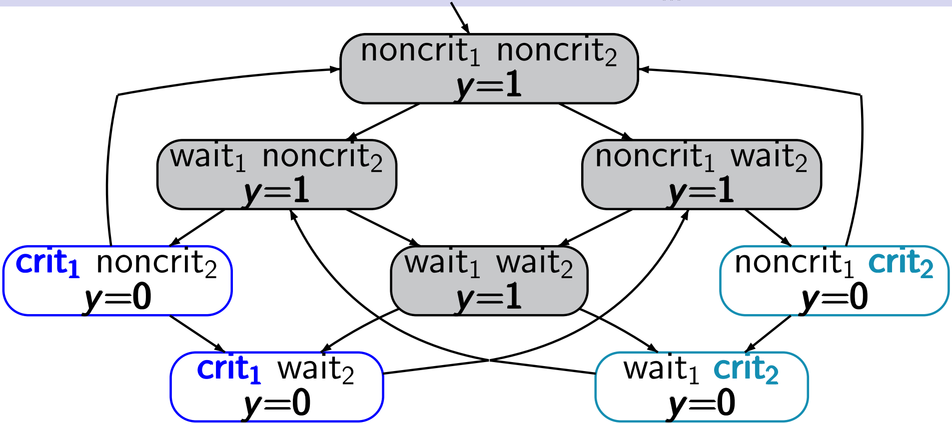
transition system $\mathcal{T}_{\mathcal{P}_1 ||| \mathcal{P}_2}$ arises by unfolding the composite program graph $\mathcal{P}_1 \; ||| \; \mathcal{P}_2$

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



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

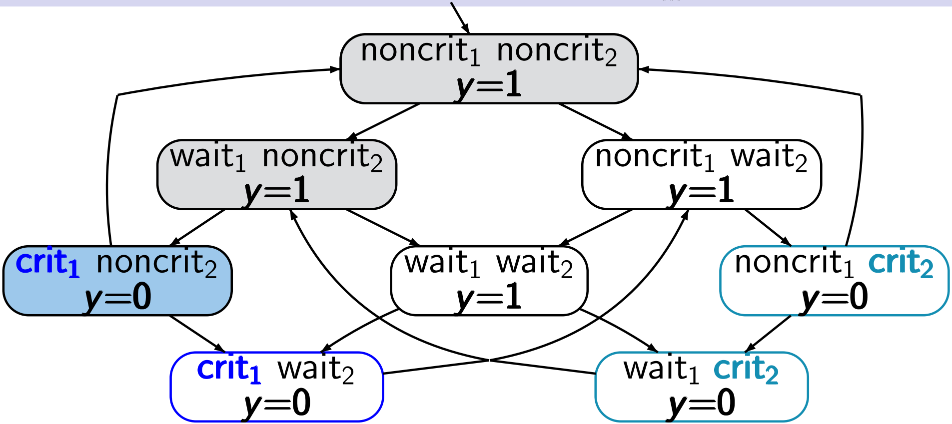
Mutual exclusion with semaphore $\mathcal{T}_{\mathcal{P}_1 ||| \mathcal{P}_2}$



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

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

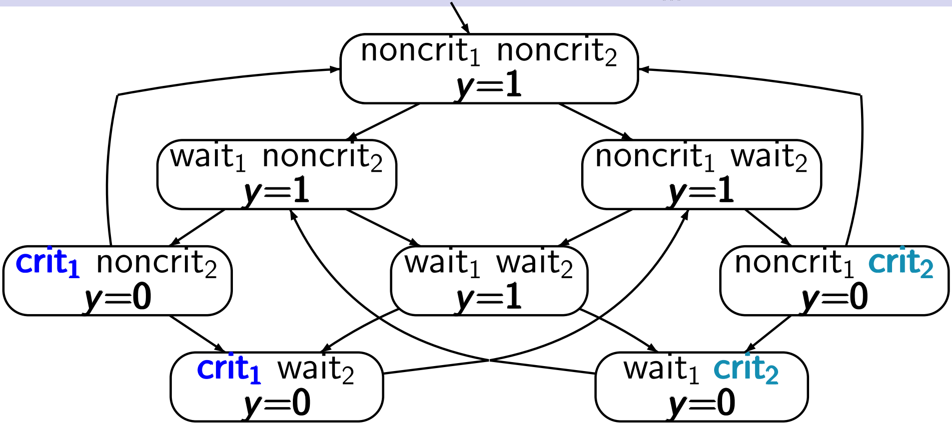
Mutual exclusion with semaphore $\mathcal{T}_{\mathcal{P}_1 ||| \mathcal{P}_2}$



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

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

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

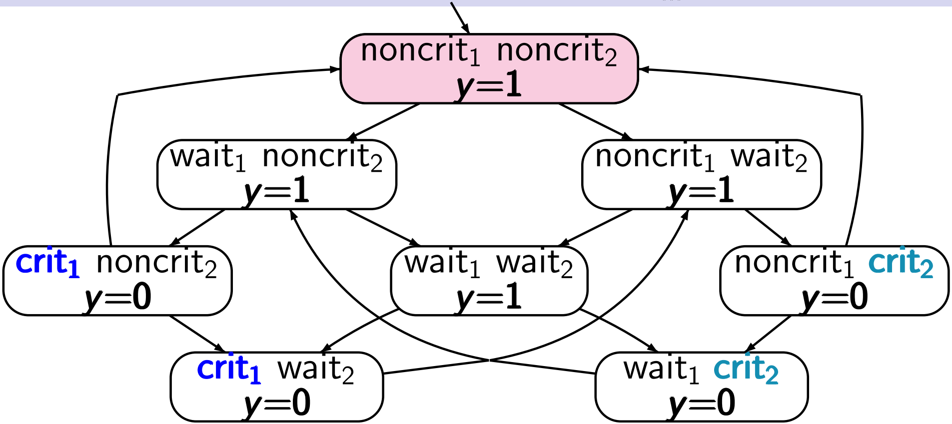


set of atomic propositions $AP = \{crit_1, crit_2\}$

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 $\mathcal{T}_{\mathcal{P}_1 ||| \mathcal{P}_2}$

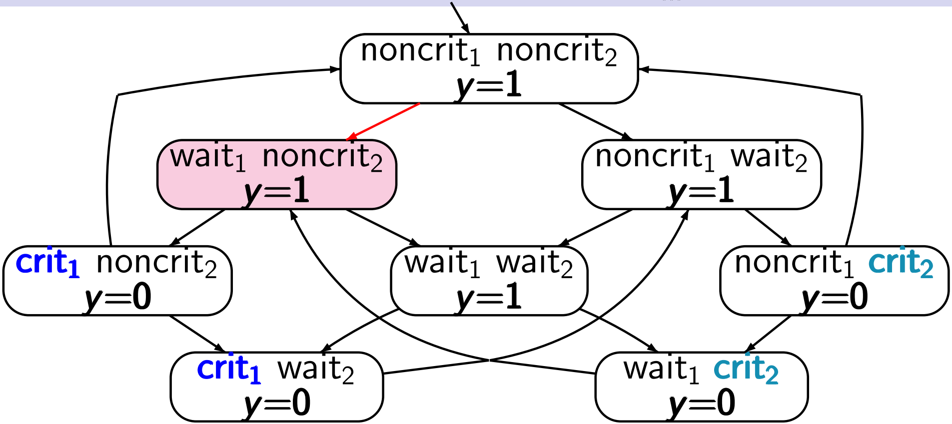


set of atomic propositions $AP = \{crit_1, crit_2\}$

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 $\mathcal{T}_{\mathcal{P}_1 ||| \mathcal{P}_2}$

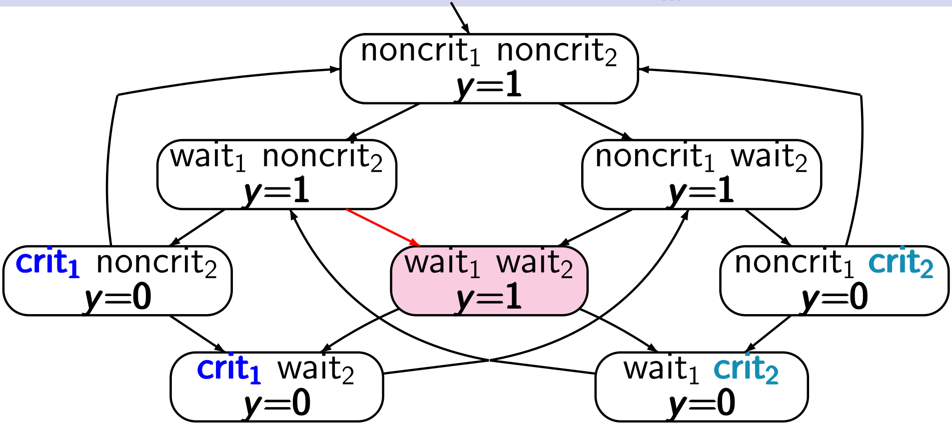


set of atomic propositions $AP = \{\mathsf{crit}_1, \mathsf{crit}_2\}$

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

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

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

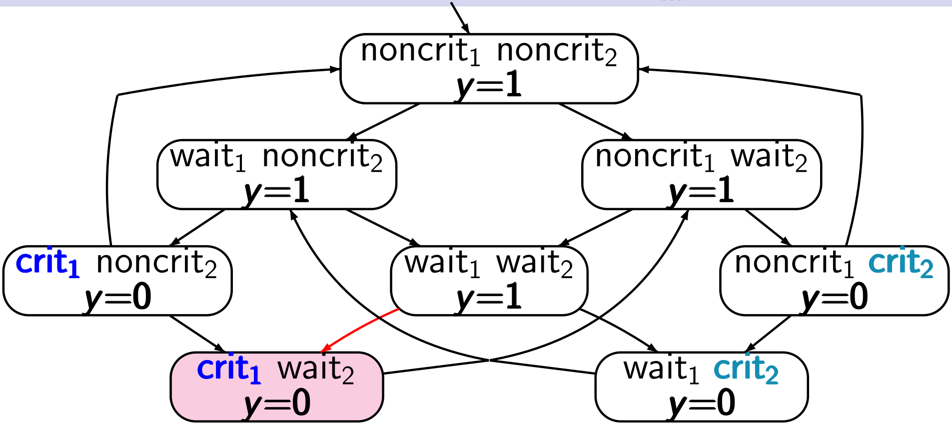


set of atomic propositions $AP = \{crit_1, crit_2\}$

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 $\mathcal{T}_{\mathcal{P}_1 ||| \mathcal{P}_2}$

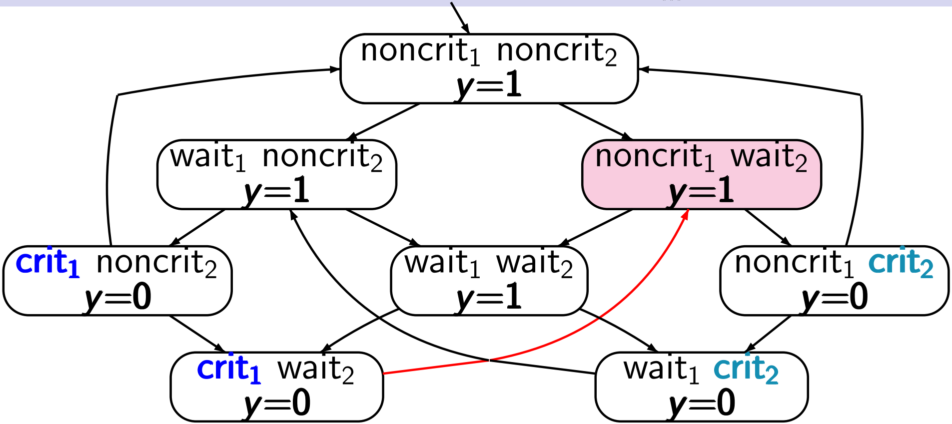


set of atomic propositions $AP = \{crit_1, crit_2\}$

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 $\mathcal{T}_{\mathcal{P}_1 ||| \mathcal{P}_2}$

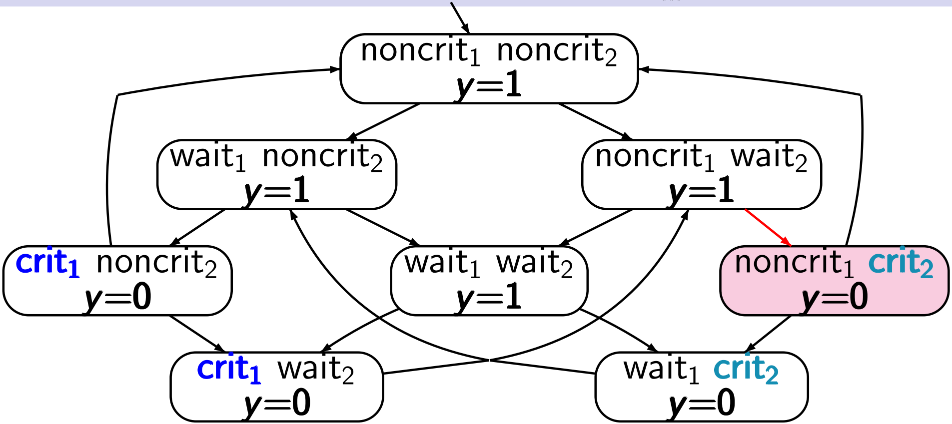


set of atomic propositions $AP = \{crit_1, crit_2\}$

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 $\mathcal{T}_{\mathcal{P}_1 ||| \mathcal{P}_2}$

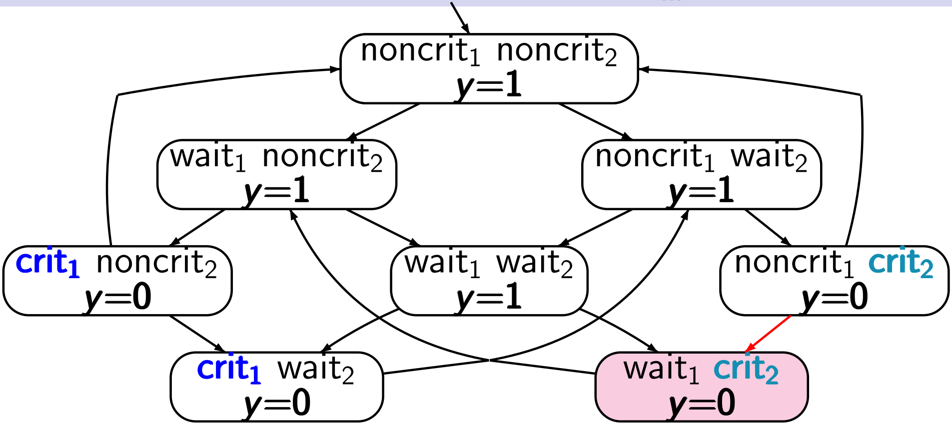


set of atomic propositions $AP = \{crit_1, crit_2\}$

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 $\mathcal{T}_{\mathcal{P}_1 ||| \mathcal{P}_2}$

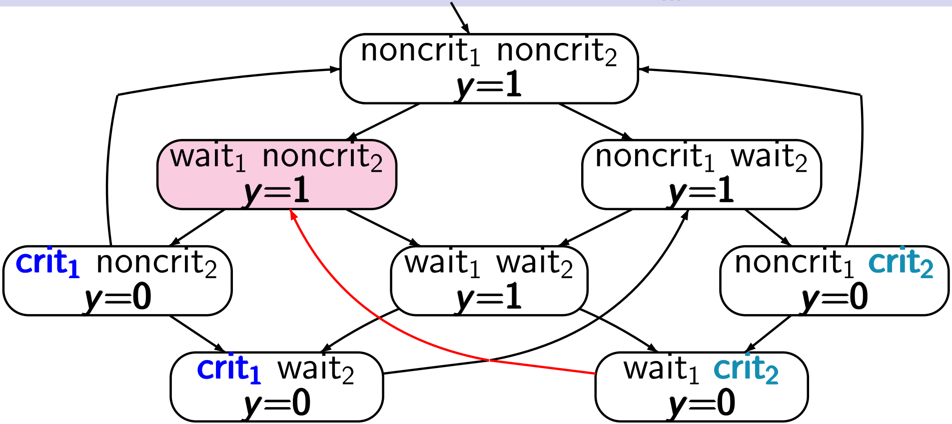


set of atomic propositions $AP = \{crit_1, crit_2\}$

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 $\mathcal{T}_{\mathcal{P}_1 ||| \mathcal{P}_2}$

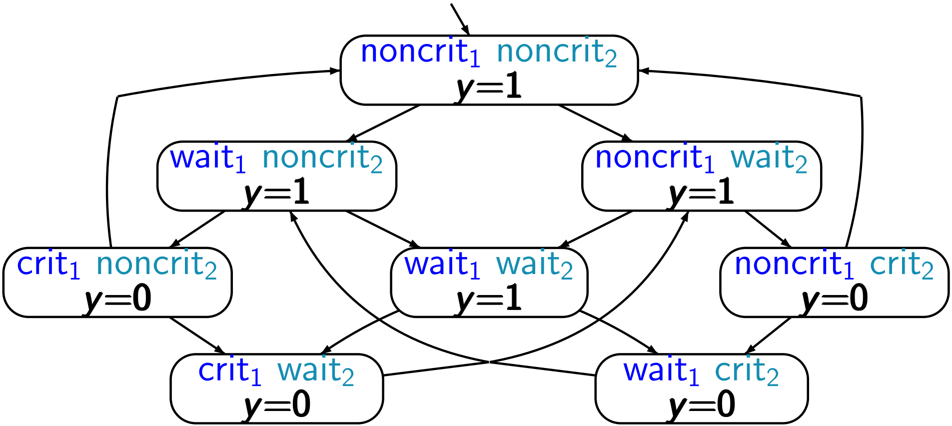


set of atomic propositions $AP = \{crit_1, crit_2\}$

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

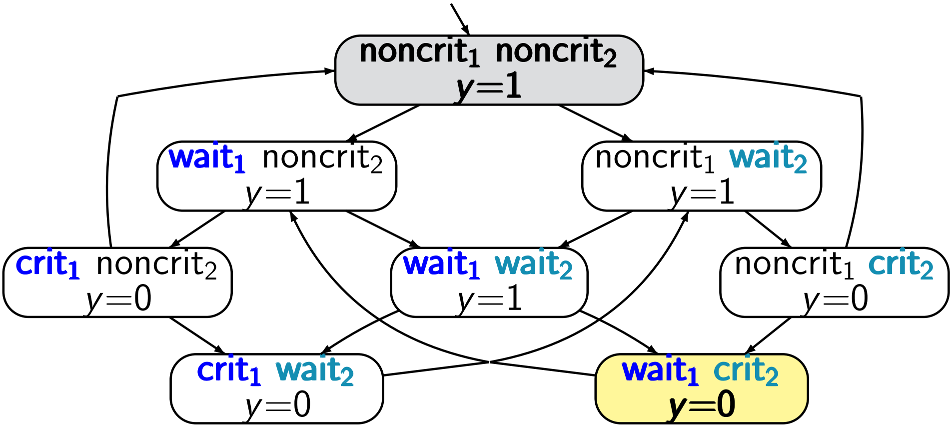
$\varnothing\ \varnothing\ \varnothing\ \{crit_1\}\ \varnothing\ \{crit_2\}\ \{crit_2\}\ \varnothing \ldots$

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 $\mathcal{T}_{\mathcal{P}_1 ||| \mathcal{P}_2}$

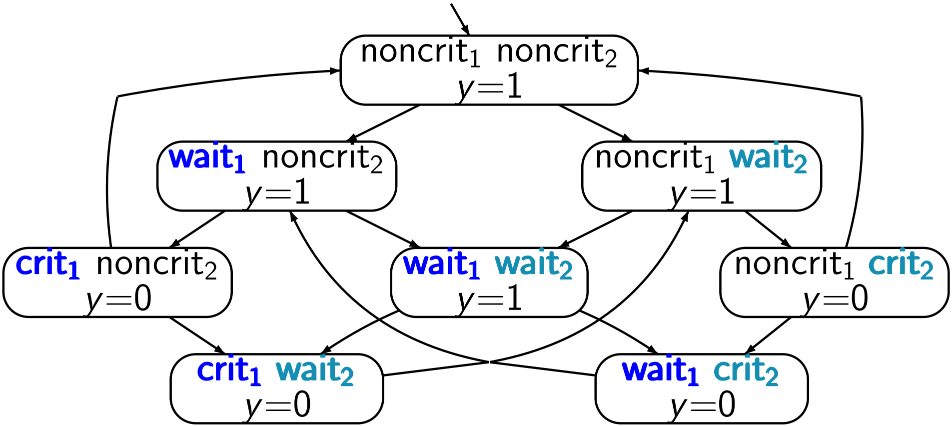


set of propositions $AP = \{\text{wait}_1, \text{crit}_1, \text{wait}_2, \text{crit}_2\}$

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 $\mathcal{T}_{\mathcal{P}_1 ||| \mathcal{P}_2}$

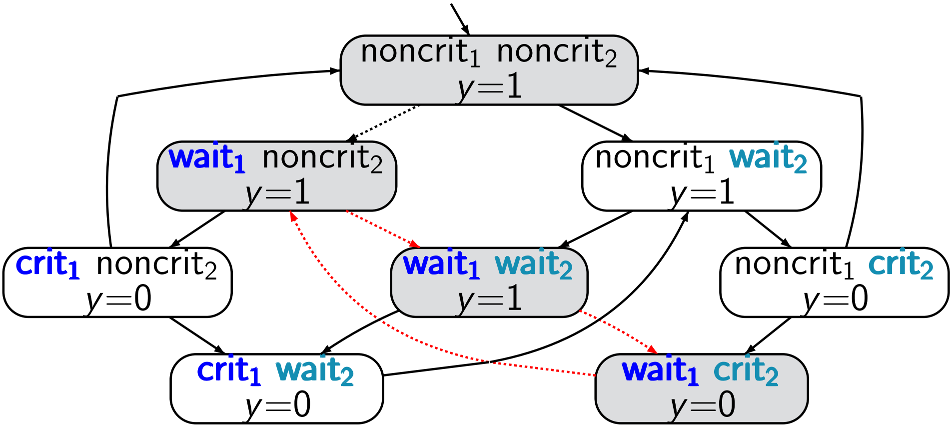


set of propositions $AP = \{wait_1, crit_1, wait_2, crit_2\}$

traces, e.g.,

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

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

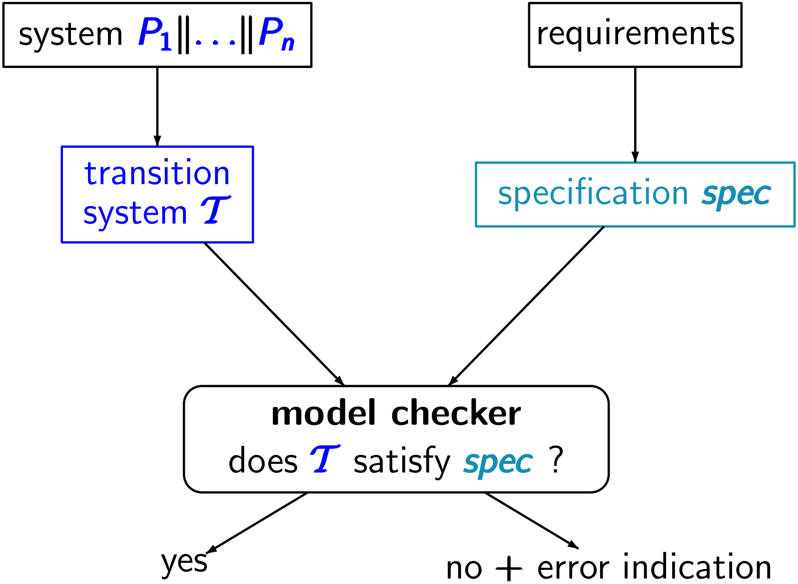


set of propositions $AP = \{wait_1, crit_1, wait_2, crit_2\}$

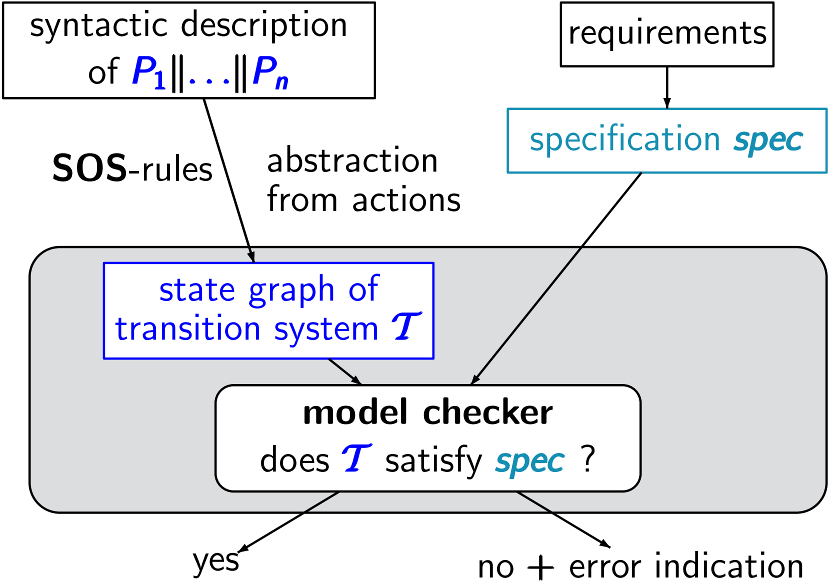
traces, e.g.,

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

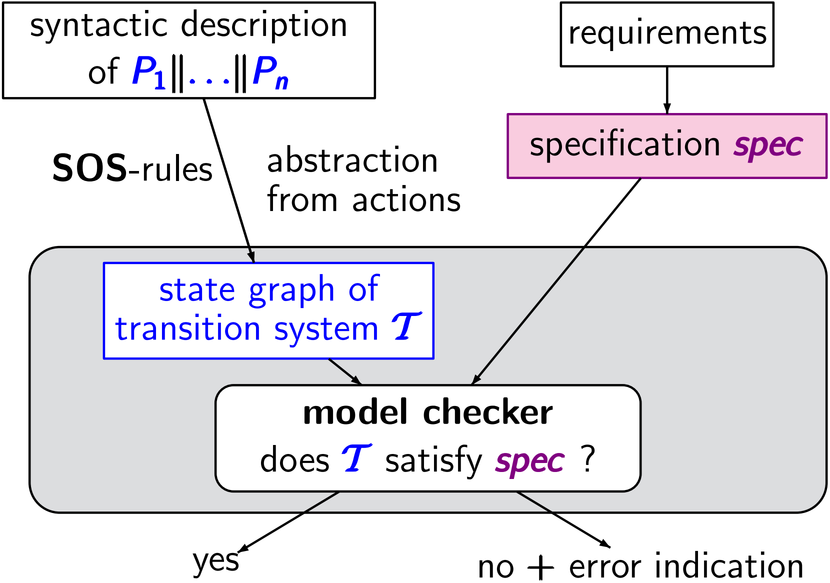
Model checking



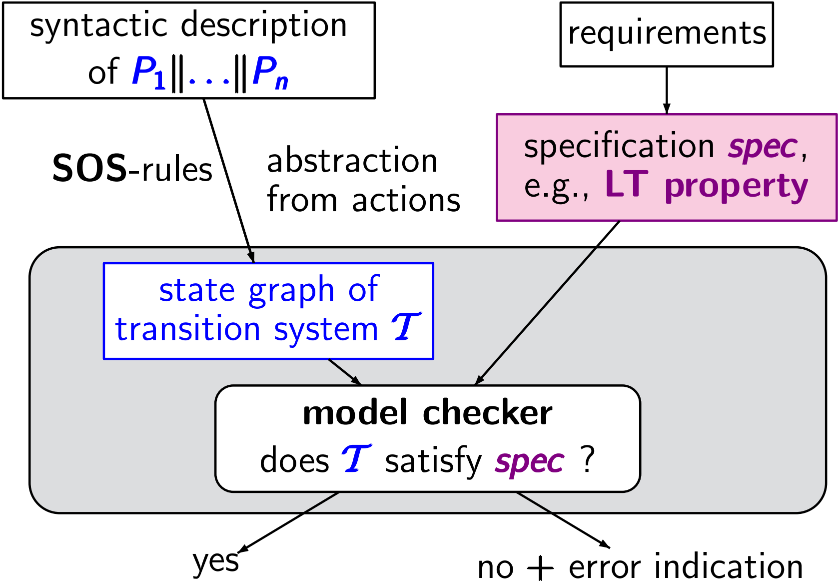
Model checking



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 $\Sigma = 2^{AP}$, 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

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

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 $\text{crit}_2 \notin A_i$

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

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 $crit_2 \notin A_i$

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

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

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 $crit_2 \notin A_i$

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

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

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

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 $crit_2 \notin A_i$

liveness (starvation freedom):

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

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

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

Satisfaction relation for LT properties

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:

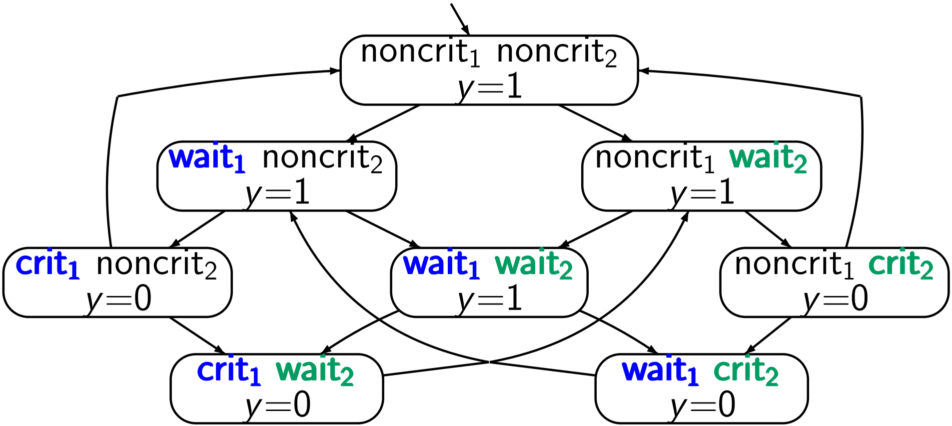
If $\mathcal{T}$ is a TS (without terminal states) over AP and E an LT property over AP then

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

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

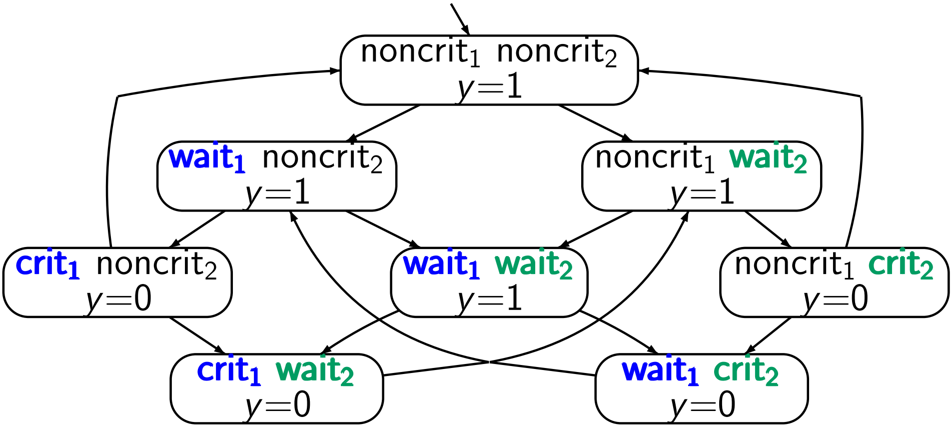
$$s \models E \quad \text{iff} \quad Traces(s) \subseteq E$$

Mutual exclusion with semaphore



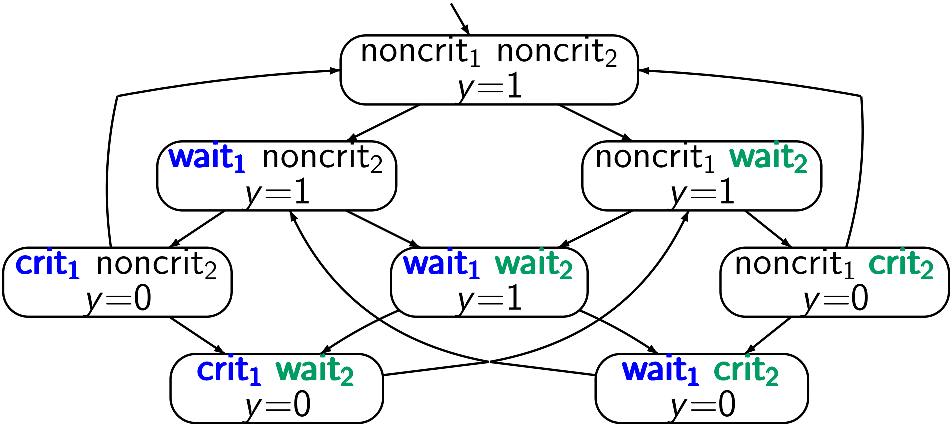
$\mathcal{T}_{Sem} \models MUTEX$

Mutual exclusion with semaphore



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

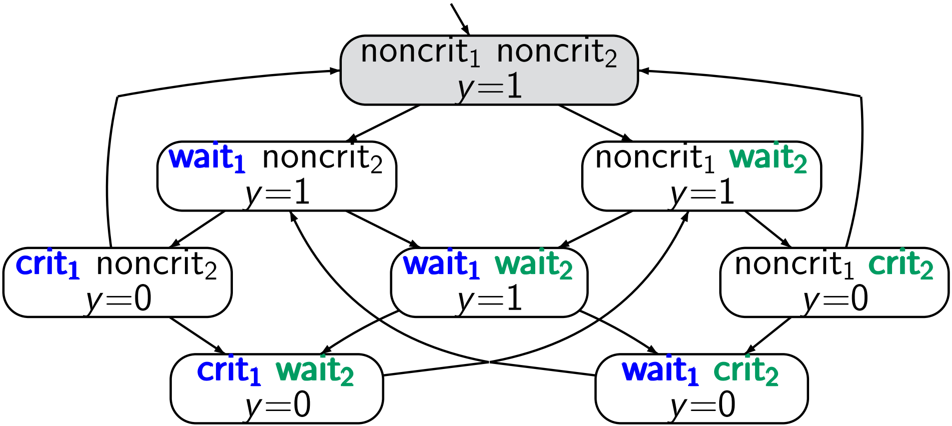
Mutual exclusion with semaphore



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

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

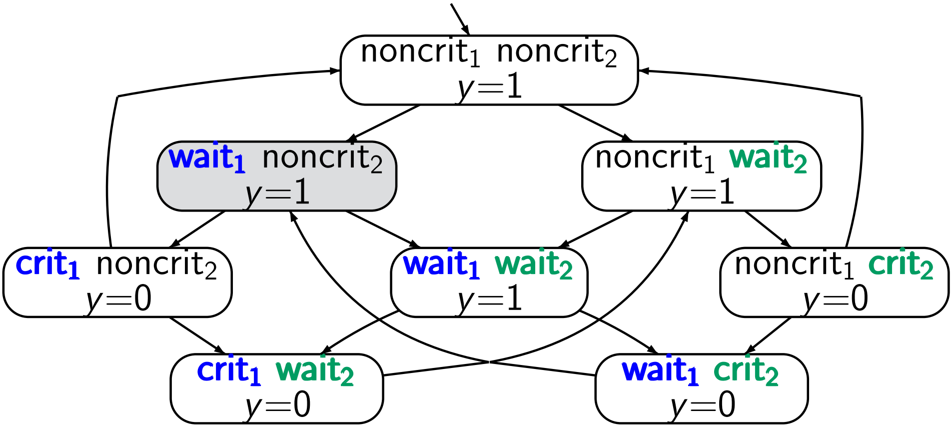
Mutual exclusion with semaphore



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

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

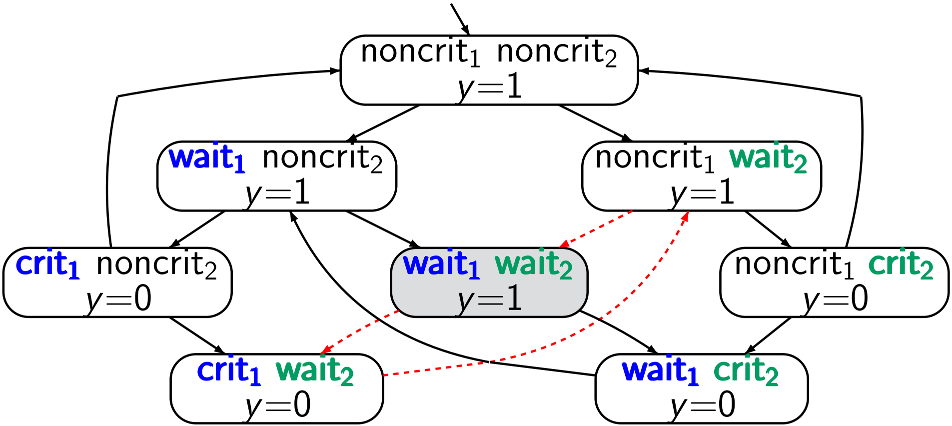
Mutual exclusion with semaphore



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

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

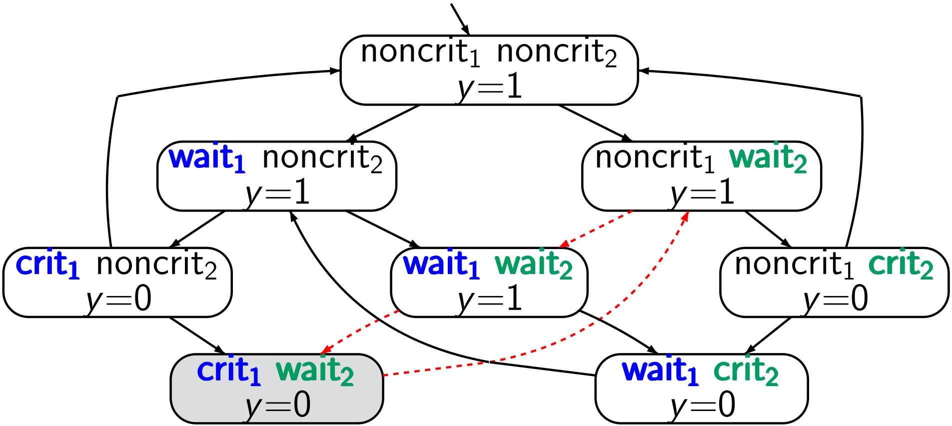
Mutual exclusion with semaphore



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

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

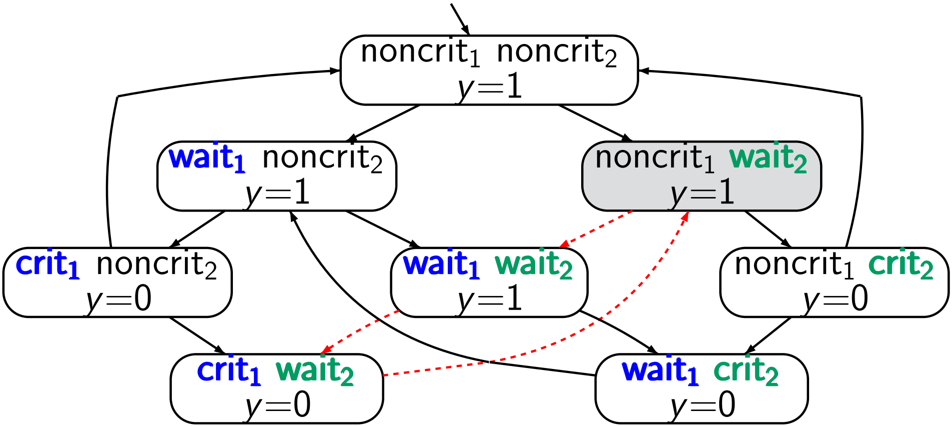
Mutual exclusion with semaphore



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

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

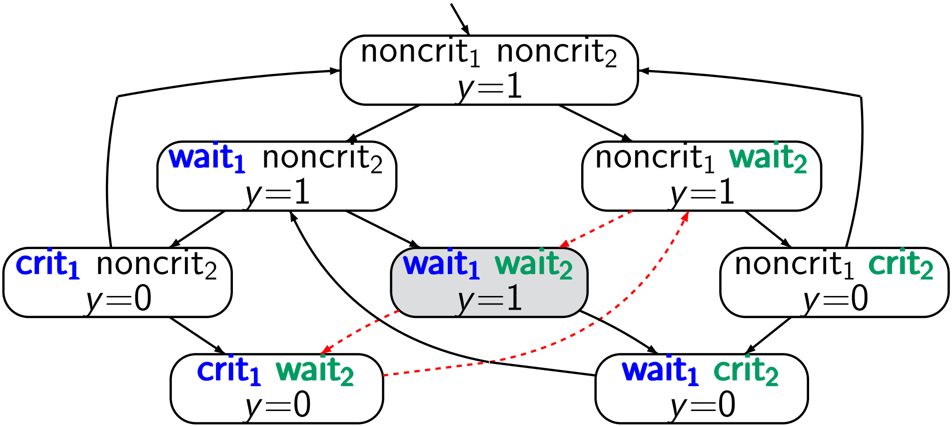
Mutual exclusion with semaphore



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

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

Mutual exclusion with semaphore



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

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

Peterson's mutual exclusion algorithm

Peterson's mutual exclusion algorithm

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

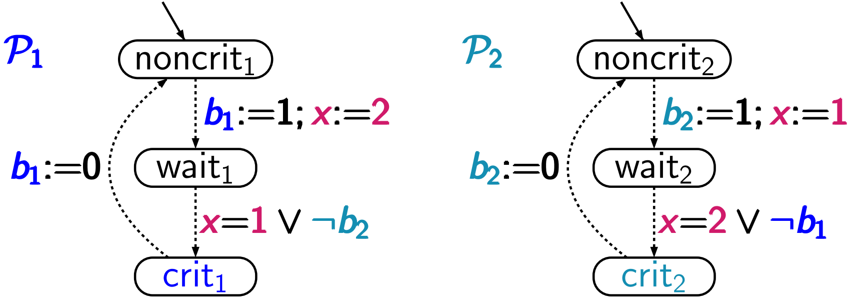
using three additional shared variables

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

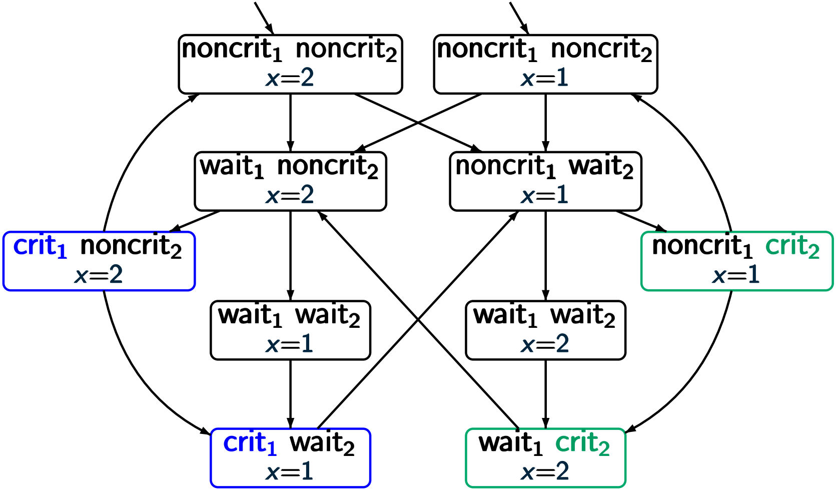
Peterson's mutual exclusion algorithm

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

using three additional shared variables
$b_1, b_2 \in \{0,1\}$, $x \in \{1,2\}$

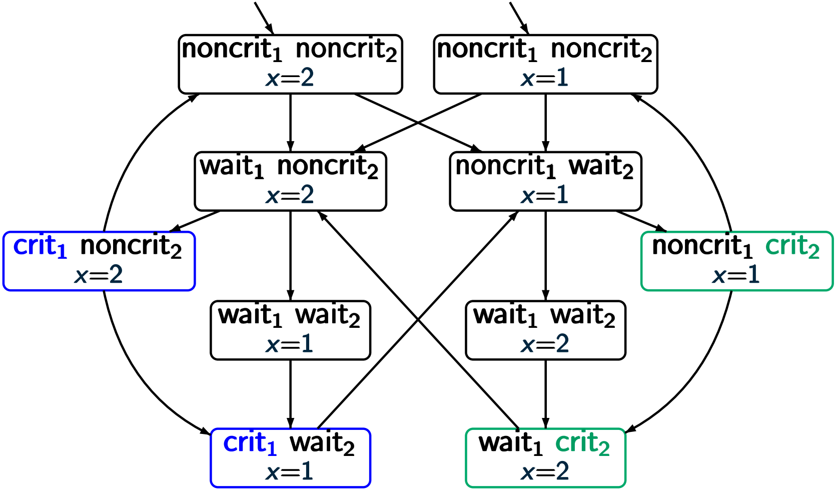


Peterson's mutual exclusion algorithm



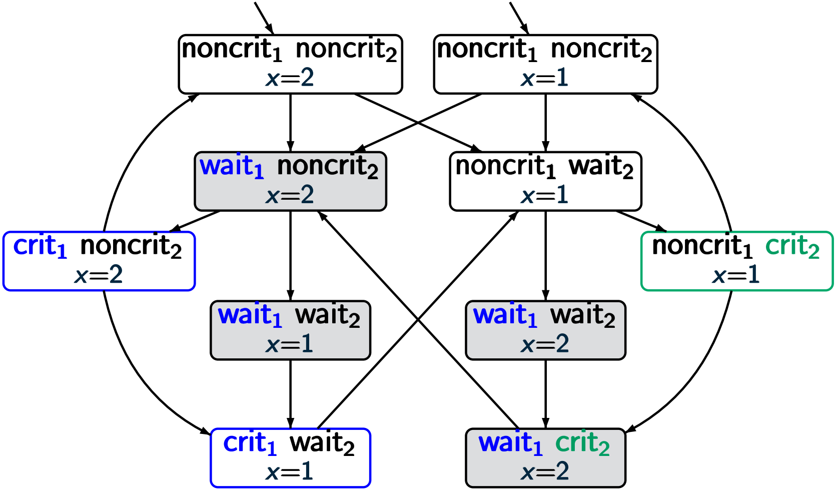
$\mathcal{T}_{Pet} \models MUTEX$

Peterson's mutual exclusion algorithm



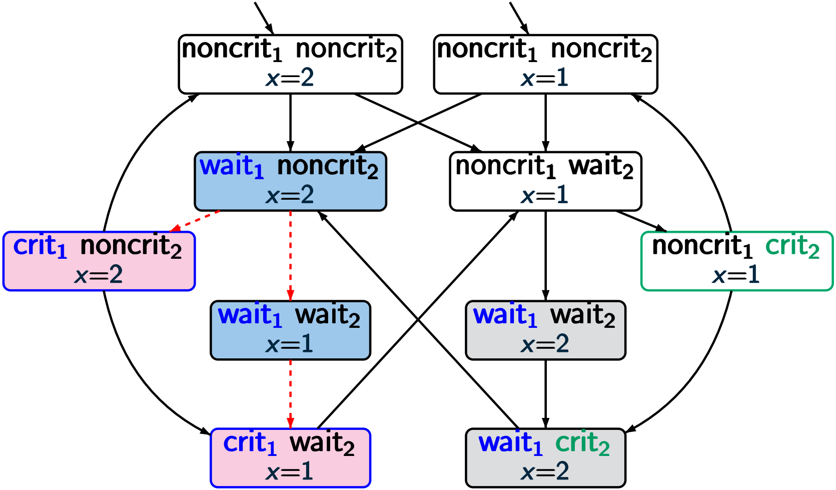
$\mathcal{T}_{Pet} \models MUTEX$ and $\mathcal{T}_{Pet} \models LIVE$

Peterson's mutual exclusion algorithm



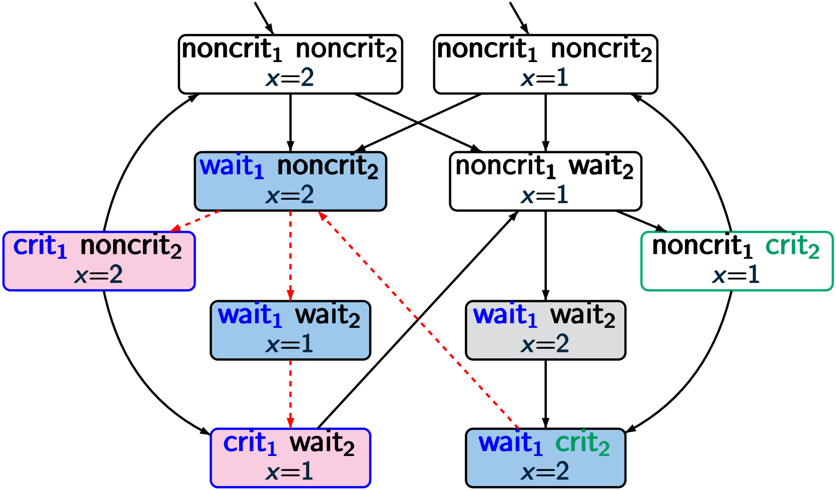
$\mathcal{T}_{Pet} \models \mathit{MUTEX}$ and $\mathcal{T}_{Pet} \models \mathit{LIVE}$

Peterson's mutual exclusion algorithm



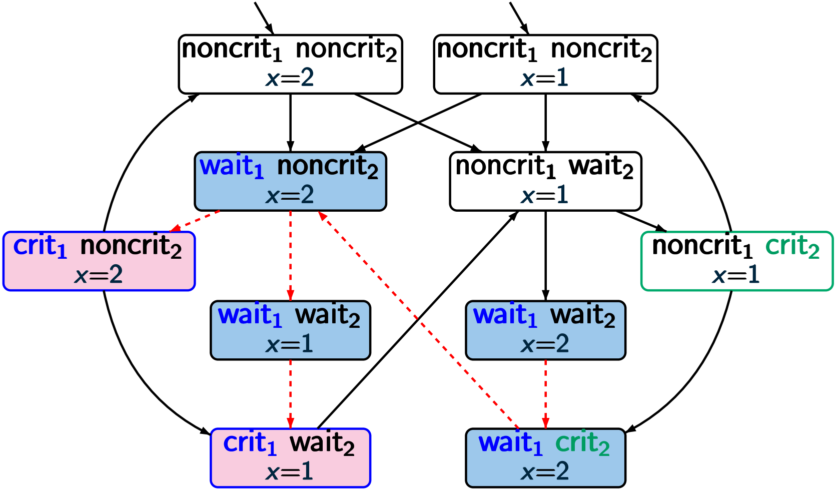
$\mathcal{T}_{Pet} \models \mathit{MUTEX}$ and $\mathcal{T}_{Pet} \models \mathit{LIVE}$

Peterson's mutual exclusion algorithm



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

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., $E \subseteq (2^{AP})^{\omega}$.

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

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., $E \subseteq (2^{AP})^\omega$.

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 $\mathcal{T}_1 \models E$

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

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 $\mathcal{T}_1 \models E$

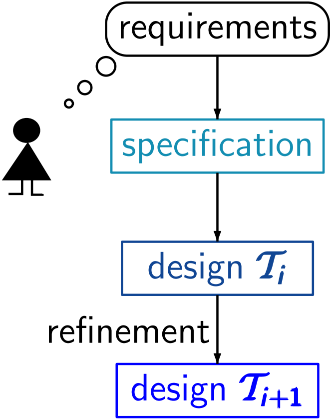
$(2) \Longrightarrow (1)$: consider $E = Traces(\mathcal{T}_2)$

Relevance of trace inclusion

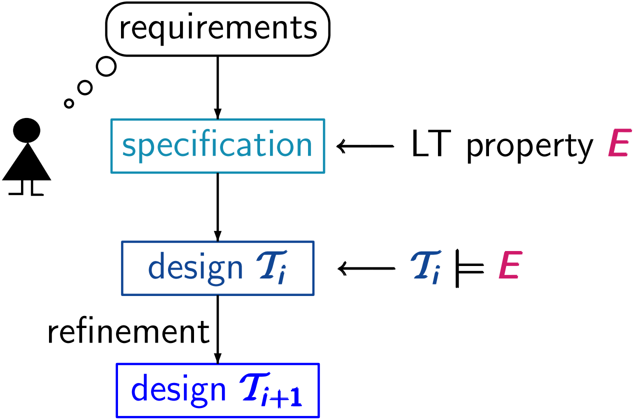
Trace inclusion appears naturally

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

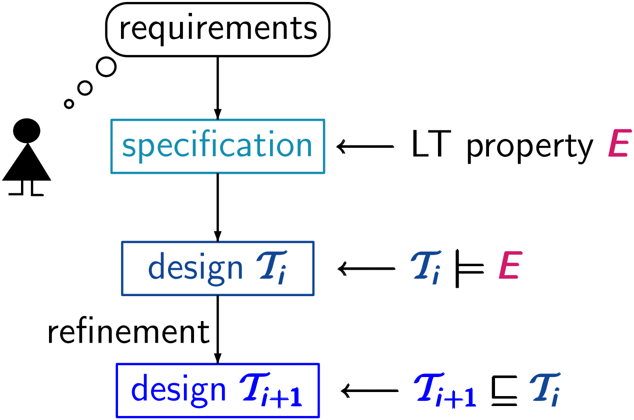
Software design cycle



Software design cycle



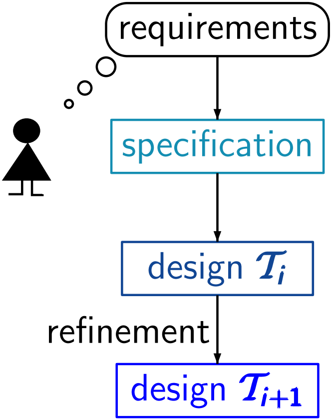
Software design cycle



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

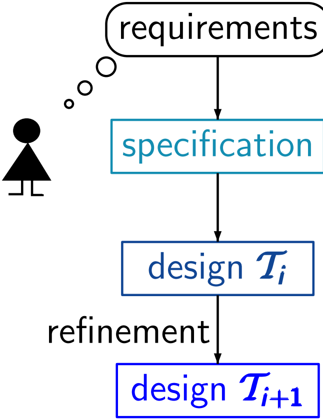
$\mathcal{T}_{i+1} \sqsubseteq \mathcal{T}_i$ iff

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

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

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

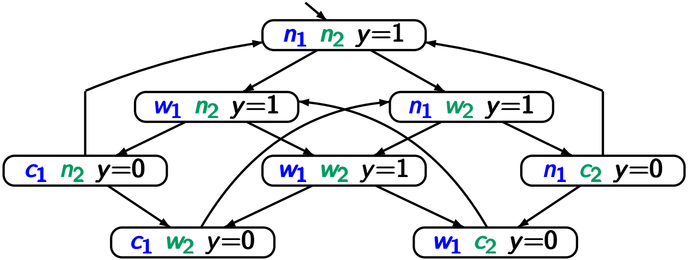
$\longleftarrow$ $\mathcal{T}_i \models E$

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

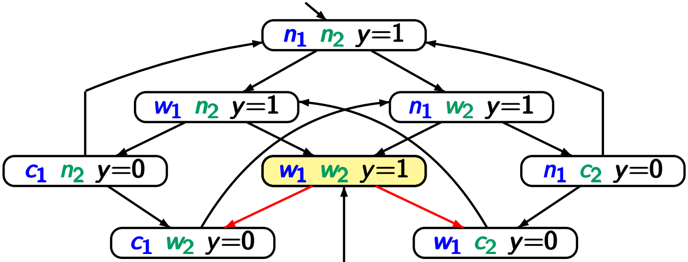
implementation/refinement relation $\sqsubseteq$:

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

Mutual exclusion with semaphore

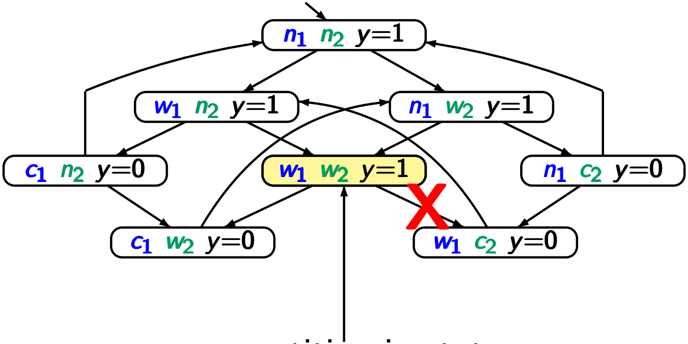


Mutual exclusion with semaphore



competition in state
$\langle wait_1 \ wait_2 \ y{=}1 \rangle$

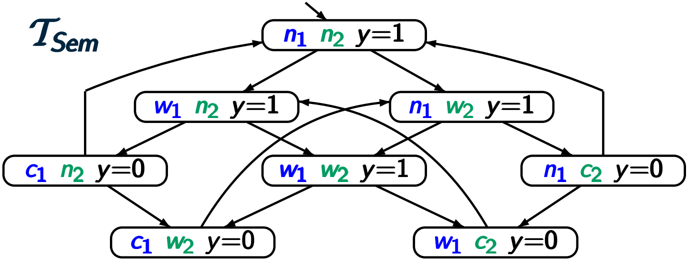
Mutual exclusion with semaphore

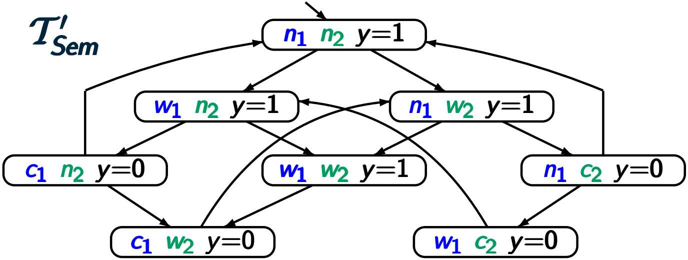


competition in state
$\langle wait_1\ wait_2\ y{=}1 \rangle$

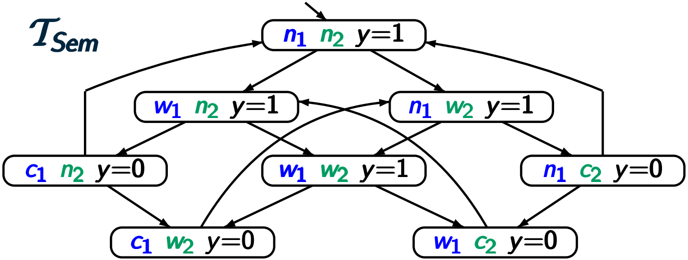
resolve the nondeterminism by giving priority to process P_1

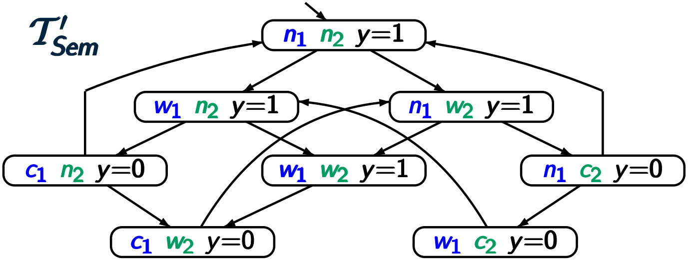
Mutual exclusion with semaphore





Mutual exclusion with semaphore

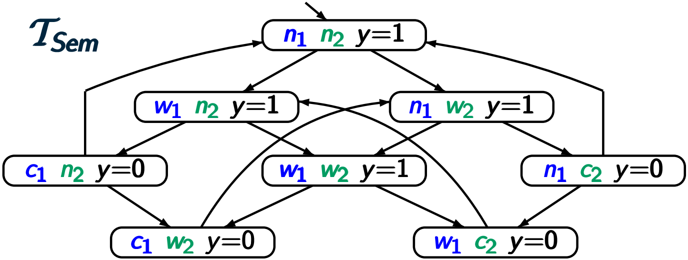




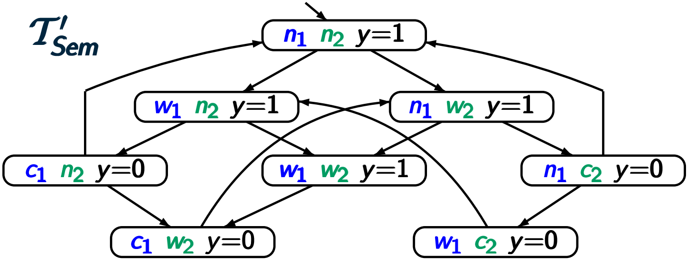
$$Paths(\mathcal{T}'_{Sem}) \subseteq Paths(\mathcal{T}_{Sem})$$

Mutual exclusion with semaphore

$\mathcal{T}_{Sem}$

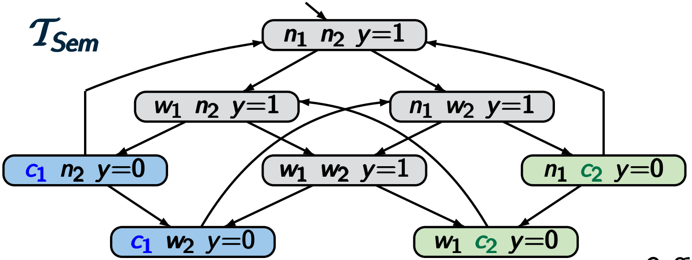


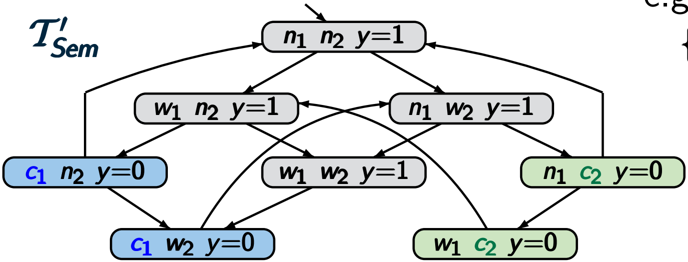
$\mathcal{T}'_{Sem}$



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

Mutual exclusion with semaphore





e.g., for $AP = \{\text{crit}_1, \text{crit}_2\}$

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

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

- 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

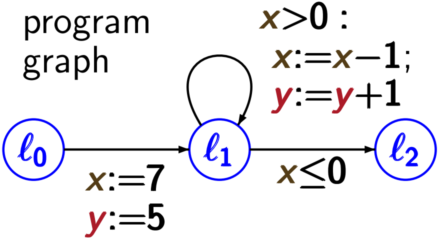

```
       ⋮
ℓ0   x:=7; y:=5;
ℓ1   WHILE x>0 DO
            x:=x−1;
            y:=y+1
     OD
ℓ2     ⋮
```

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      ⋮
```

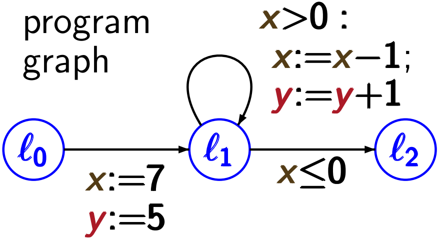
program graph



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

Trace inclusion and data abstraction

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



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

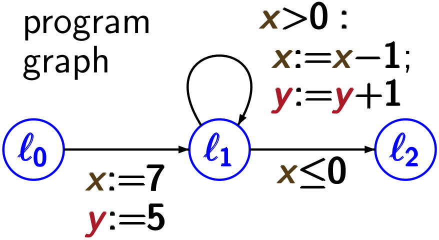
does $\ell_2 \wedge odd(y)$ never hold ?

$\longleftarrow \mathcal{T} \models$ "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 $\ell_2 \wedge odd(y)$" ?

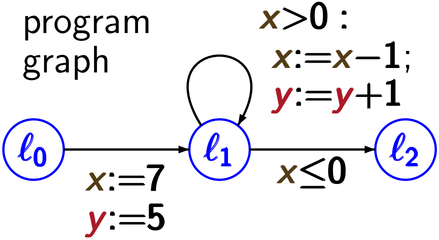
data abstraction w.r.t. the predicates

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

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

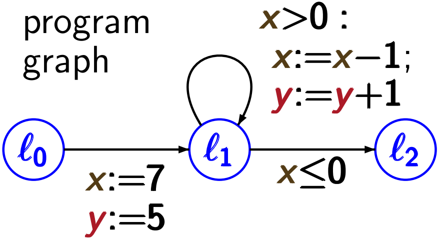
data abstraction w.r.t. the predicates

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

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

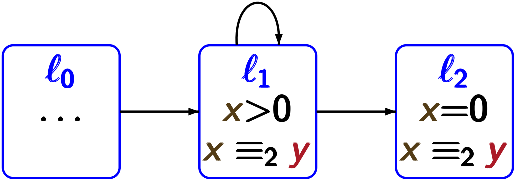


let $\mathcal{T}$ be the associated TS

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

data abstraction w.r.t. the predicates

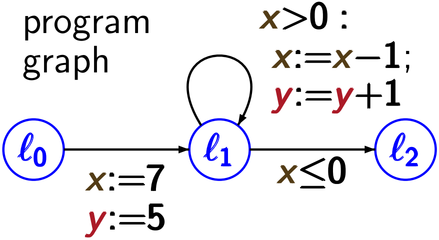
$x>0$, $x=0$, $x \equiv_2 y$



abstract transition system $\mathcal{T}'$

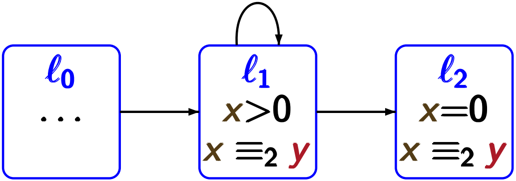
```
      ⋮
ℓ0   x:=7;  y:=5;
ℓ1   WHILE x>0 DO
           x:=x−1;
           y:=y+1
     OD
ℓ2    ⋮
```

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 $\ell_2 \wedge odd(y)$"

Trace inclusion and data abstraction

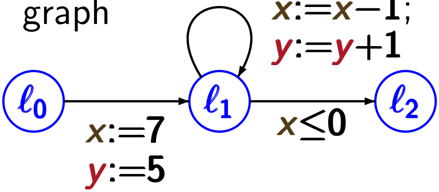

```
        ⋮
ℓ0   x:=7;  y:=5;
ℓ1   WHILE x>0 DO
            x:=x−1;
            y:=y+1
     OD
ℓ2      ⋮
```

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

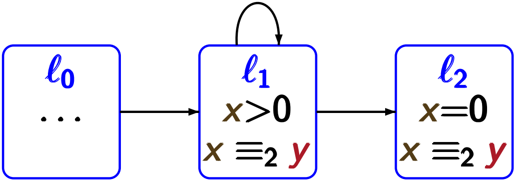
data abstraction w.r.t. the predicates

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

program graph



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



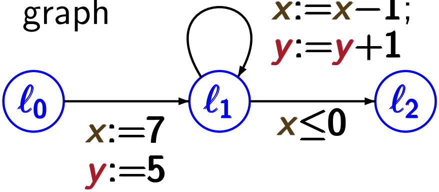
$\mathcal{T}' \models$ "never $\ell_2 \wedge odd(y)$"

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

Trace inclusion and data abstraction

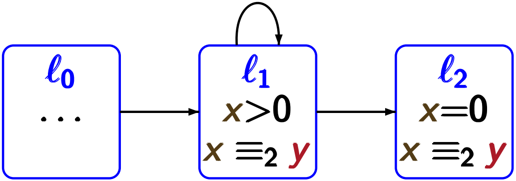

```
      ⋮
ℓ0   x:=7;  y:=5;
ℓ1   WHILE x>0 DO
           x:=x−1;
           y:=y+1
     OD
ℓ2    ⋮
```

program graph



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

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



$\mathcal{T} \models$ "never $\ell_2 \wedge odd(y)$" $\left\{ \begin{array}{l} \mathcal{T}' \models \text{"never } \ell_2 \wedge odd(y)\text{"} \\ Traces(\mathcal{T}) \subseteq Traces(\mathcal{T}') \end{array} \right.$

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

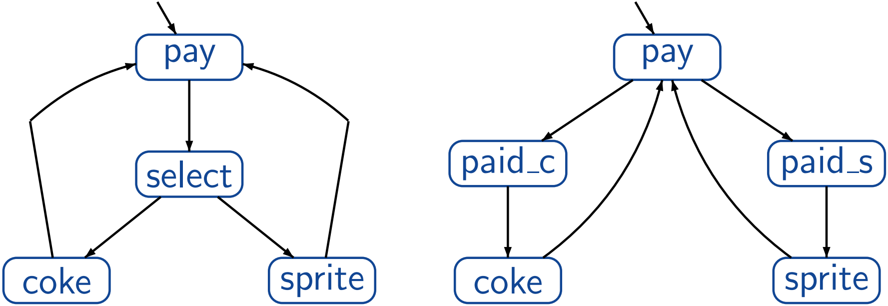
(2) for all LT-properties E: $\mathcal{T}_2 \models E \Longrightarrow \mathcal{T}_1 \models E$

The following statements are equivalent:

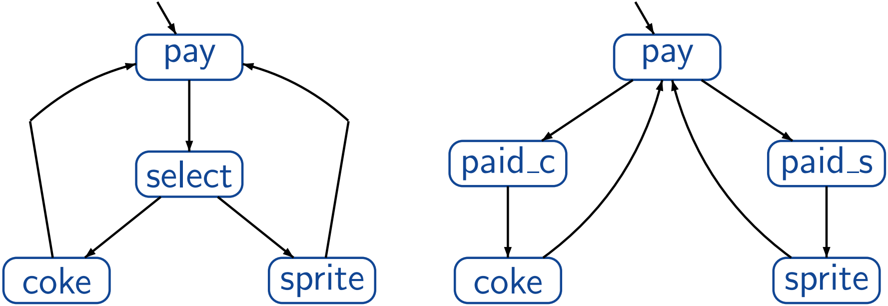
(1) $Traces(\mathcal{T}_1) = Traces(\mathcal{T}_2)$

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

Trace equivalent beverage machines

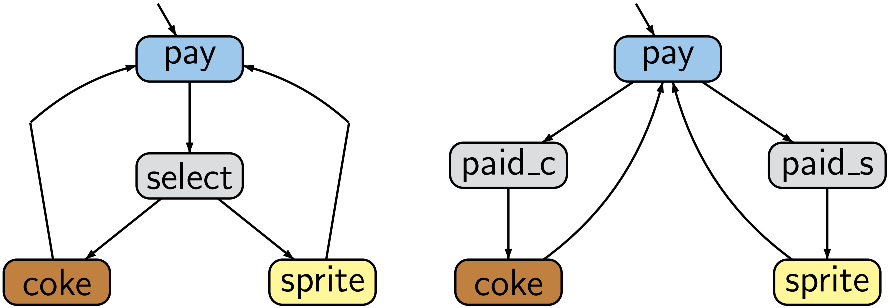


Trace equivalent beverage machines



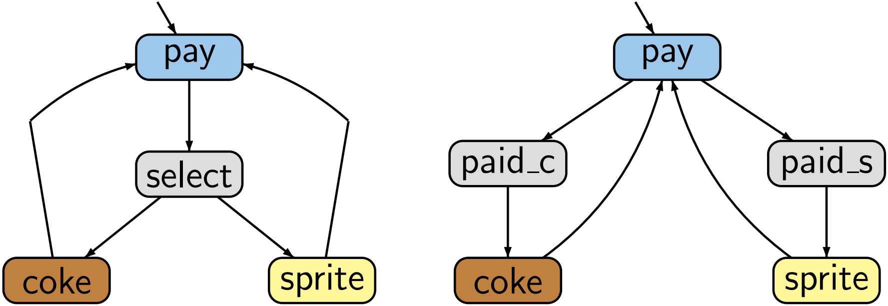
set of atomic propositions $AP = \{pay, coke, sprite\}$

Trace equivalent beverage machines



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

Trace equivalent beverage machines



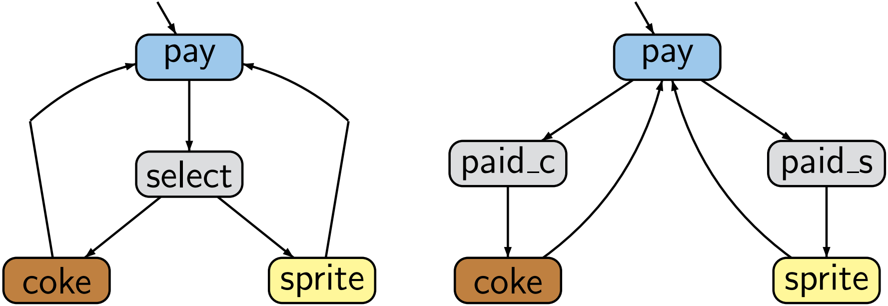
set of atomic propositions $AP = \{pay, coke, sprite\}$

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

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

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

Overview

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

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

↑

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

Propositional logic

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

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

semantics: interpretation over a subsets of AP

Propositional logic

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

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

semantics: Let $A \subseteq AP$

$A \models \mathit{true}$		
$A \models a$	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 $A \subseteq AP$

$$\begin{array}{lll} A \models \textit{true} & & \\ A \models a & \text{iff} & a \in A \\ A \models \Phi_1 \wedge \Phi_2 & \text{iff} & A \models \Phi_1 \text{ and } A \models \Phi_2 \\ A \models \neg\Phi & \text{iff} & A \not\models \Phi \end{array}$$

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

Propositional logic

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

↑ 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	$A \not\models \Phi$

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

Let E be an LT property over AP.

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

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

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):

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

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

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

here: $AP = \{crit_1, crit_2, \ldots\}$

Examples for invariants

mutual exclusion (safety):

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

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

deadlock freedom for 5 dining philosophers:

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

invariant condition:

$$\Phi = \neg wait_0 \vee \neg wait_1 \vee \neg wait_2 \vee \neg wait_3 \vee \neg wait_4$$

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

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.

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

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.

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

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 $\mathcal{T}$

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 $\mathcal{T}$

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.

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

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

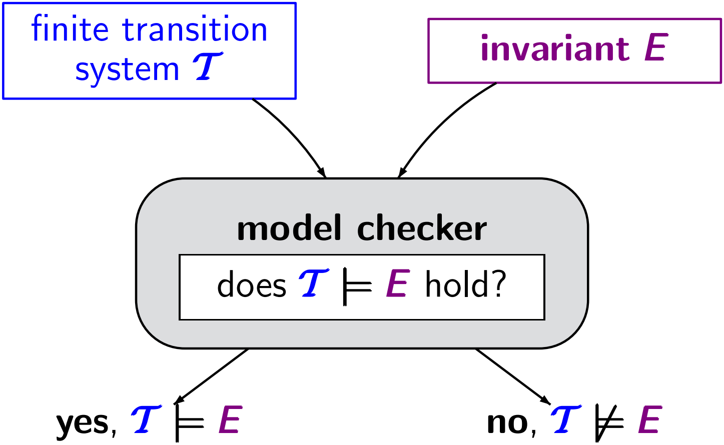
$\mathcal{T} \models E$ iff $trace(\pi) \in E$ for all $\pi \in Paths(\mathcal{T})$

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

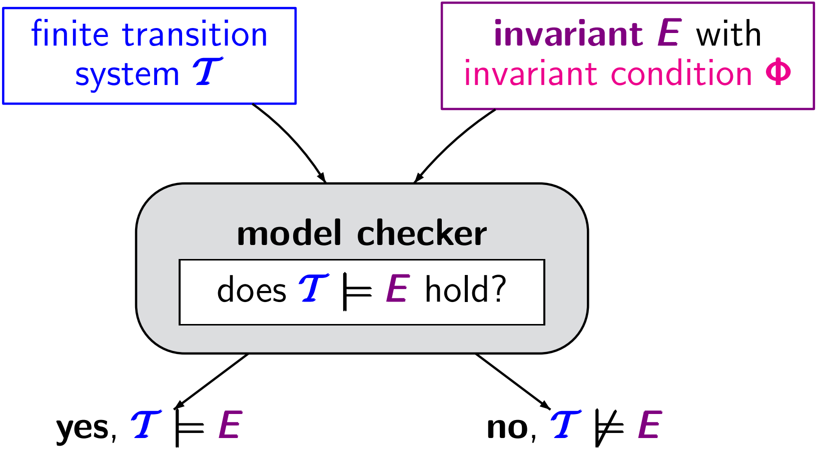
iff $s \models \Phi$ for all states $s \in Reach(\mathcal{T})$

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

Invariant checking

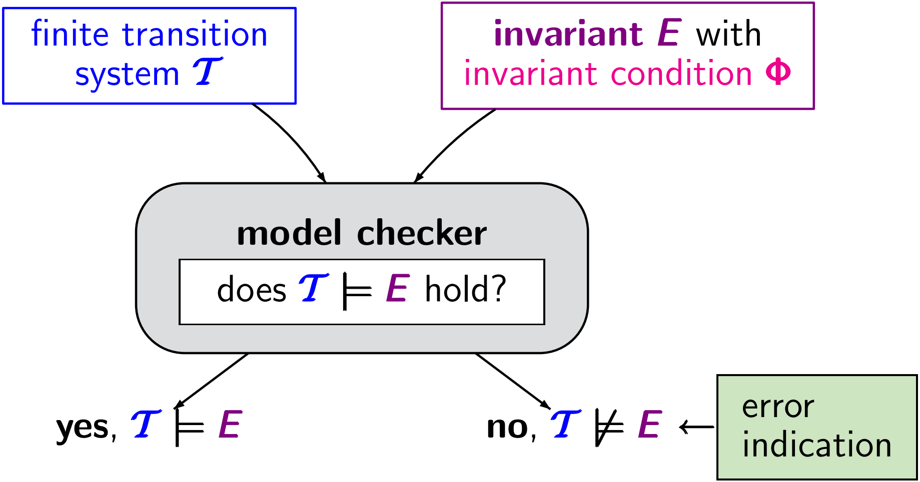


Invariant checking



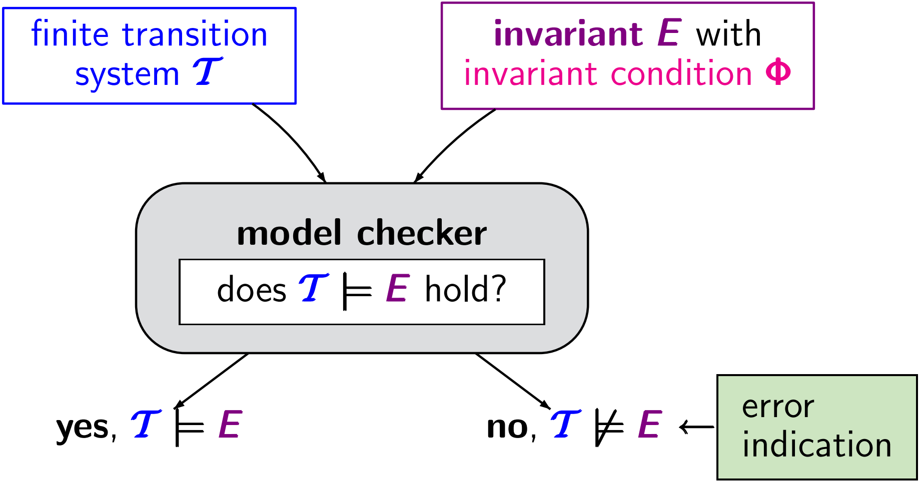
perform a graph analysis (**DFS** or **BFS**) to check whether $s \models \Phi$ for all $s \in Reach(\mathcal{T})$

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 $0 \le i < n$ 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 $t \not\models \Phi$

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

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

FOR ALL $s_0 \in S_0$ DO

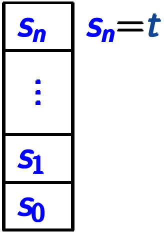
  IF $DFS(s_0, \Phi)$ THEN

    return "no" and $reverse(\pi)$

  FI

OD

return "yes"



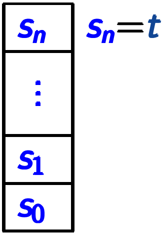
$DFS(s_0, \Phi)$ returns "true" iff depth-first search from state s_0 leads to some state t with $t \not\models \Phi$

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

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

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

FOR ALL $s_0 \in S_0$ DO
  IF $DFS(s_0, \Phi)$ THEN
    return "no" and $reverse(\pi)$
  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 $s \dots t$ 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 $DFS(s, \Phi)$

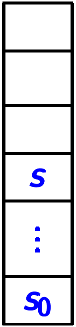
"searches" for a path fragment $s \ldots t$ with $t \not\models \Phi$

```
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 $s \dots t$ 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"
```

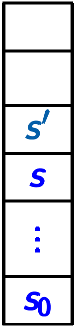


initial state

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"

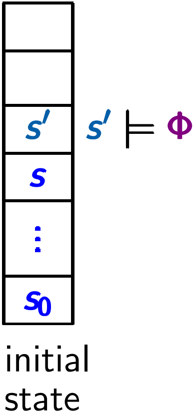


initial state

Recursive algorithm $DFS(s, \Phi)$

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

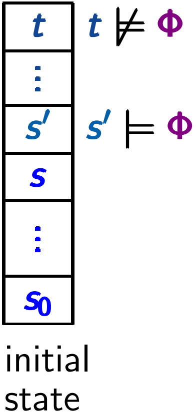
$Push(\pi, s)$;
IF $s \notin U$ THEN
    IF $s \not\models \Phi$ THEN return "true" FI
    IF $s \models \Phi$ THEN
        insert s in U;
        FOR ALL $s' \in Post(s)$ DO
            IF $DFS(s', \Phi)$ THEN
                return "true" FI
        OD
    FI
FI
$Pop(\pi)$; return "false"



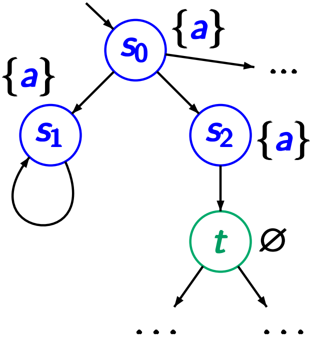
Recursive algorithm $DFS(s, \Phi)$

"searches" for a path fragment $s \ldots s' \ldots t$ 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"
```



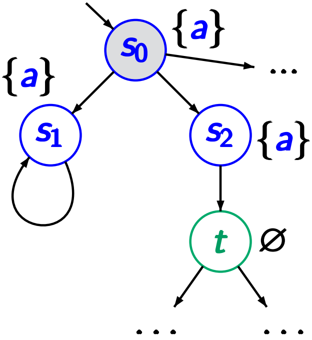
Example: invariant checking



invariant
condition a

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

Example: invariant checking



invariant condition a

$s_0, s_1, s_2 \models a$

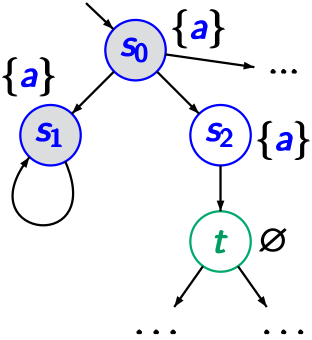
$t \not\models a$

$DFS(s_0, a)$

stack π

s_0

Example: invariant checking



invariant condition a

$s_0, s_1, s_2 \models a$

$t \not\models a$

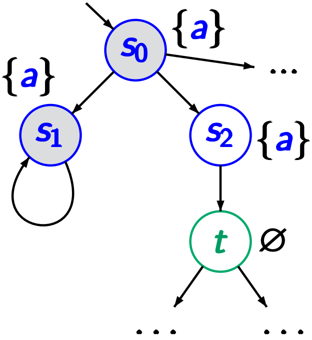
$DFS(s_0, a)$

$DFS(s_1, a)$

stack π

s_1
s_0

Example: invariant checking



invariant
condition a

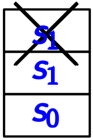
$s_0, s_1, s_2 \models a$
$t \not\models a$

$DFS(s_0, a)$

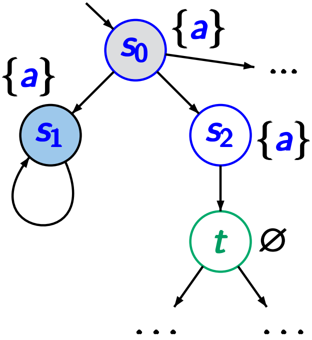
$DFS(s_1, a)$

$DFS(s_1, a)$

stack π



Example: invariant checking



invariant condition a

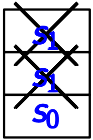
$$s_0, s_1, s_2 \models a$$
$$t \not\models a$$

$DFS(s_0, a)$

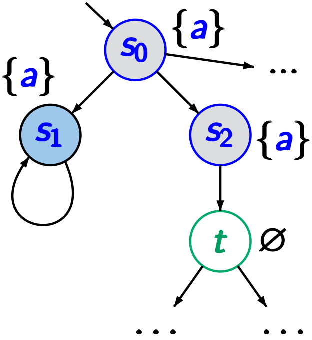
$DFS(s_1, a)$

$DFS(s_1, a)$

stack π



Example: invariant checking



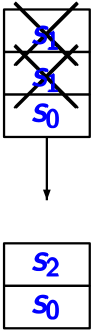
$DFS(s_0, a)$

$DFS(s_1, a)$

$DFS(s_1, a)$

$DFS(s_2, a)$

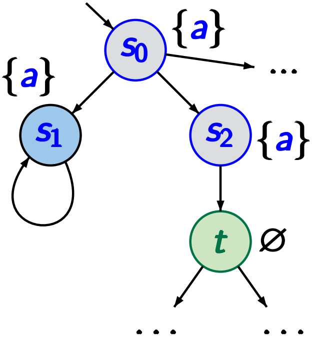
stack π



invariant condition a

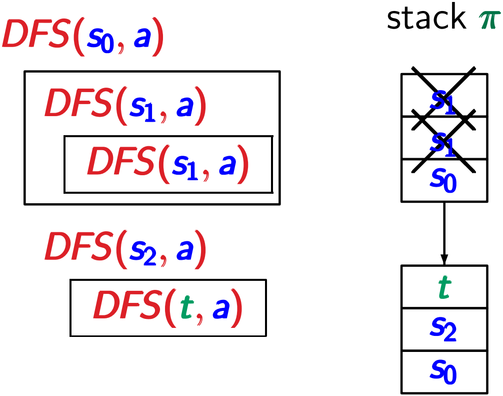
$$s_0, s_1, s_2 \models a$$
$$t \not\models a$$

Example: invariant checking



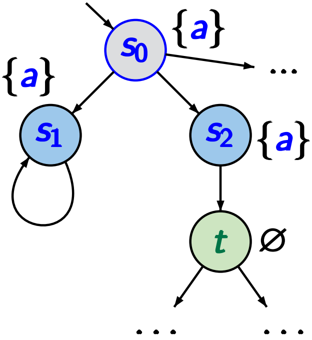
invariant condition a

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



Example: invariant checking

IS2.5-9



invariant condition a

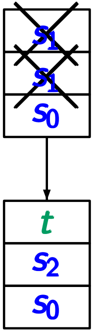
$s_0, s_1, s_2 \models a$

$t \not\models a$

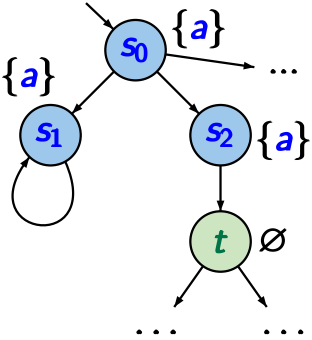
$DFS(s_0, a)$

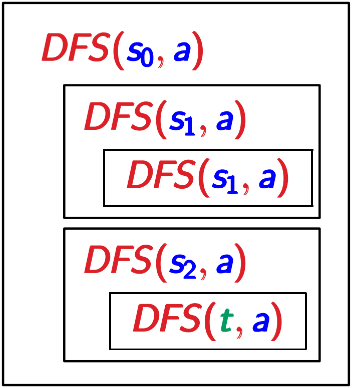
- $DFS(s_1, a)$
  - $DFS(s_1, a)$
- $DFS(s_2, a)$
  - $DFS(t, a)$

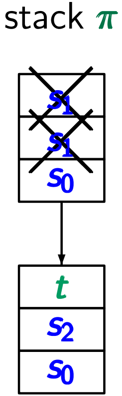
stack π



Example: invariant checking



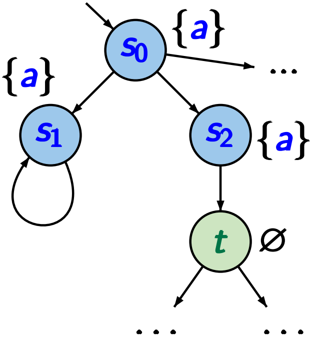


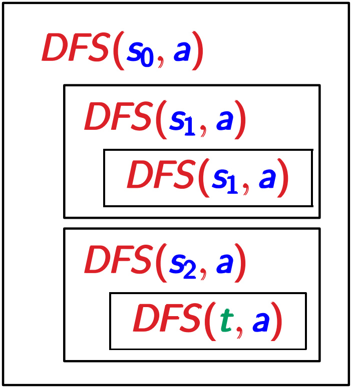


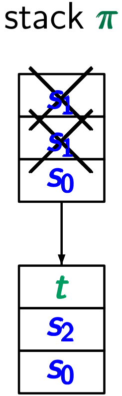
invariant condition a

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

Example: invariant checking





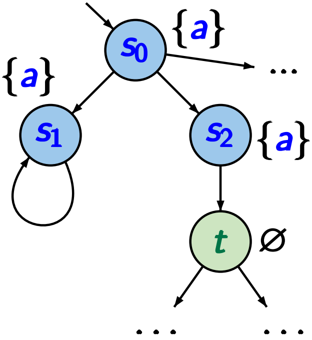


invariant condition a

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

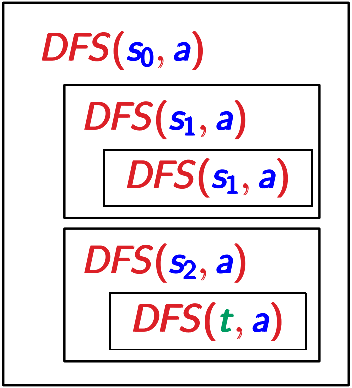
$s_0 \not\models$ "always a"

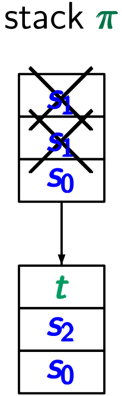
Example: invariant checking



invariant condition a

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





$s_0 \not\models$ "always a" $\longleftarrow$

error indication: $s_0\ s_2\ t$

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

Safety properties

state that "nothing bad will happen"

Safety properties

state that "nothing bad will happen"

invariants:

- mutual exclusion: *never* $crit_1 \wedge crit_2$
- deadlock freedom: *never* $\bigwedge_{0 \leq i < n} wait_i$

---

other safety properties:

- German traffic lights:
  every red phase is preceded by a yellow phase
- beverage machine:
  the total number of entered coins is never less than the total number of released drinks

Safety properties

state that "nothing bad will happen"

invariants: ⟵ "no **bad state** will be reached"

- mutual exclusion: *never* $crit_1 \wedge crit_2$
- deadlock freedom: *never* $\bigwedge_{0 \le i < n} wait_i$

other safety properties:

- German traffic lights:
  every red phase is preceded by a yellow phase
- beverage machine:
  the total number of entered coins is never less than the total number of released drinks

Safety properties

state that "nothing bad will happen"

invariants: ⟵ "no **bad state** will be reached"

- mutual exclusion: *never* $crit_1 \wedge crit_2$
- deadlock freedom: *never* $\bigwedge_{0 \leq i < n} wait_i$

other safety properties: ⟵ "no **bad prefix**"

- German traffic lights:
  every red phase is preceded by a yellow phase
- beverage machine:
  the total number of entered coins is never less than the total number of released drinks

Bad prefixes of safety properties

IS2.5-10B

- traffic lights:

every red phase is preceded by a yellow phase

↑

bad prefix: finite trace fragment where a red phase appears without being preceded by a yellow phase

e.g., $\ldots \{\bullet\}\{\bullet\}$

Bad prefixes of safety properties

- traffic lights:

  every red phase is preceded by a yellow phase

  ↑

  bad prefix: finite trace fragment where a red phase appears without being preceded by a yellow phase
  e.g., ... {●} {●}

- beverage machine:

  the total number of entered coins is never less than the total number of released drinks

  ↑

  bad prefix, e.g., {*pay*} {*drink*} {*drink*}

Definition of safety properties

Let E be a LT property over AP, i.e., $E \subseteq (2^{AP})^\omega$.

Definition of safety properties

Let E be a LT property over AP, i.e., $E \subseteq (2^{AP})^\omega$.

E is called a safety property if for all words

$$\sigma = A_0 A_1 A_2 \ldots \in \left(2^{AP}\right)^\omega \setminus E$$

there exists a finite prefix $A_0 A_1 \ldots A_n$ of σ such that *none* of the words $A_0 A_1 \ldots A_n B_{n+1} B_{n+2} B_{n+3} \ldots$ belongs to E, i.e.,

$$E \cap \left\{\sigma' \in (2^{AP})^\omega : A_0 \ldots A_n \text{ is a prefix of } \sigma'\right\} = \varnothing$$

Definition of safety properties

Let E be a LT property over AP, i.e., $E \subseteq (2^{AP})^\omega$.

E is called a safety property if for all words

$$\sigma = A_0 A_1 A_2 \ldots \in (2^{AP})^\omega \setminus E$$

there exists a finite prefix $A_0 A_1 \ldots A_n$ of σ such that *none* of the words $A_0 A_1 \ldots A_n B_{n+1} B_{n+2} B_{n+3} \ldots$ belongs to E, i.e.,

$$E \cap \{\sigma' \in (2^{AP})^\omega : A_0 \ldots A_n \text{ is a prefix of } \sigma'\} = \varnothing$$

Such words $A_0 A_1 \ldots A_n$ are called bad prefixes for E.

Definition of safety properties

Let E be a LT property over AP, i.e., $E \subseteq (2^{AP})^\omega$.

E is called a safety property if for all words

$$\sigma = A_0 A_1 A_2 \ldots \in (2^{AP})^\omega \setminus E$$

there exists a finite prefix $A_0 A_1 \ldots A_n$ of σ such that *none* of the words $A_0 A_1 \ldots A_n B_{n+1} B_{n+2} B_{n+3} \ldots$ belongs to E, i.e.,

$$E \cap \{\sigma' \in (2^{AP})^\omega : A_0 \ldots A_n \text{ is a prefix of } \sigma'\} = \varnothing$$

Such words $A_0 A_1 \ldots A_n$ are called bad prefixes for E.

$E =$ set of all infinite words that do *not* have a bad prefix

Let E be a LT property over AP, i.e., $E \subseteq (2^{AP})^\omega$.

E is called a safety property if for all words

$$\sigma = A_0 A_1 A_2 \ldots \in (2^{AP})^\omega \setminus E$$

there exists a finite prefix $A_0 A_1 \ldots A_n$ of σ such that *none* of the words $A_0 A_1 \ldots A_n B_{n+1} B_{n+2} B_{n+3} \ldots$ belongs to E, i.e.,

$$E \cap \{\sigma' \in (2^{AP})^\omega : A_0 \ldots A_n \text{ is a prefix of } \sigma'\} = \varnothing$$

Such words $A_0 A_1 \ldots A_n$ are called bad prefixes for E.

$BadPref_E \stackrel{\text{def}}{=}$ set of bad prefixes for E

Definition of safety properties

Let E be a LT property over AP, i.e., $E \subseteq (2^{AP})^\omega$.

E is called a safety property if for all words

$$\sigma = A_0 A_1 A_2 \ldots \in (2^{AP})^\omega \setminus E$$

there exists a finite prefix $A_0 A_1 \ldots A_n$ of σ such that *none* of the words $A_0 A_1 \ldots A_n B_{n+1} B_{n+2} B_{n+3} \ldots$ belongs to E, i.e.,

$$E \cap \{\sigma' \in (2^{AP})^\omega : A_0 \ldots A_n \text{ is a prefix of } \sigma'\} = \varnothing$$

Such words $A_0 A_1 \ldots A_n$ are called bad prefixes for E.

$$BadPref_E \stackrel{\text{def}}{=} \text{set of bad prefixes for } E \subseteq (2^{AP})^+$$

Definition of safety properties

Let E be a LT property over AP, i.e., $E \subseteq (2^{AP})^\omega$.

E is called a safety property if for all words

$$\sigma = A_0 A_1 A_2 \ldots \in (2^{AP})^\omega \setminus E$$

there exists a finite prefix $A_0 A_1 \ldots A_n$ of σ such that *none* of the words $A_0 A_1 \ldots A_n B_{n+1} B_{n+2} B_{n+3} \ldots$ belongs to E, i.e.,

$$E \cap \left\{\sigma' \in (2^{AP})^\omega : A_0 \ldots A_n \text{ is a prefix of } \sigma'\right\} = \varnothing$$

Such words $A_0 A_1 \ldots A_n$ are called bad prefixes for E.

$$BadPref_E \stackrel{\text{def}}{=} \text{set of bad prefixes for } E \subseteq \left(2^{AP}\right)^+$$

↑

briefly: $BadPref$

Definition of safety properties

Let E be a LT property over AP, i.e., $E \subseteq (2^{AP})^\omega$.

E is called a safety property if for all words

$$\sigma = A_0 A_1 A_2 \ldots \in (2^{AP})^\omega \setminus E$$

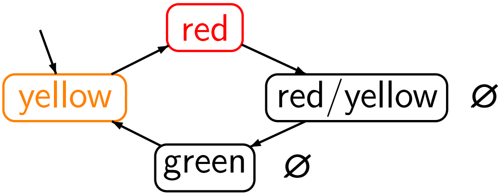
there exists a finite prefix $A_0 A_1 \ldots A_n$ of σ such that *none* of the words $A_0 A_1 \ldots A_n B_{n+1} B_{n+2} B_{n+3} \ldots$ belongs to E, i.e.,

$$E \cap \{\sigma' \in (2^{AP})^\omega : A_0 \ldots A_n \text{ is a prefix of } \sigma'\} = \varnothing$$

Such words $A_0 A_1 \ldots A_n$ are called bad prefixes for E.

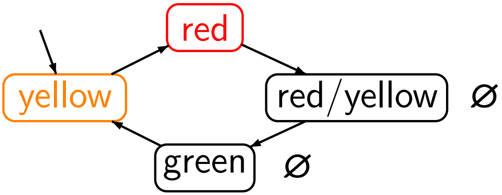
minimal bad prefixes: any word $A_0 \ldots A_i \ldots A_n \in BadPref$ s.t. no proper prefix $A_0 \ldots A_i$ is a bad prefix for E

Safety property for a traffic light



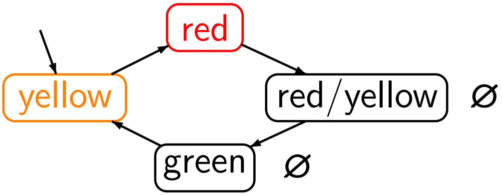
$AP = \{red, yellow\}$

Safety property for a traffic light



"every red phase is preceded by a yellow phase"

Safety property for a traffic light



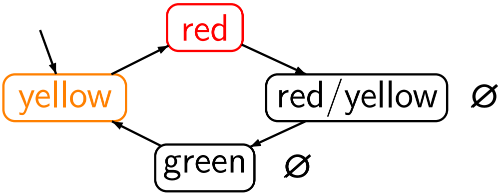
"every red phase is preceded by a yellow phase"

hence: $\mathcal{T} \models E$

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

$$red \in A_i \implies i \geq 1 \text{ and } yellow \in A_{i-1}$$

Safety property for a traffic light

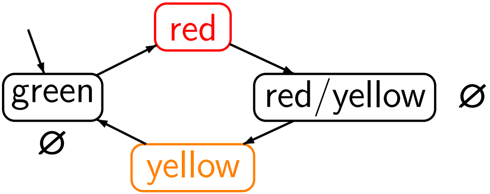


"every red phase is preceded by a yellow phase"

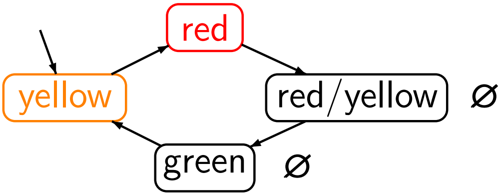
hence: $\mathcal{T} \models E$

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

$$red \in A_i \implies i \geq 1 \text{ and } yellow \in A_{i-1}$$



Safety property for a traffic light

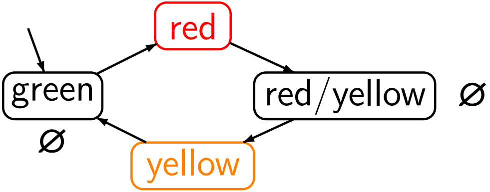


"every red phase is preceded by a yellow phase"

hence: $\mathcal{T} \models E$

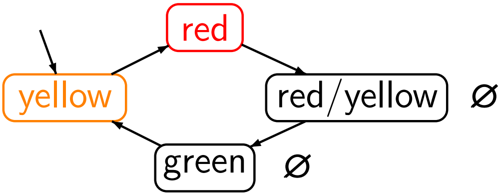
E = set of all infinite words $A_0\, A_1\, A_2 \ldots$ over 2^{AP} such that for all $i \in \mathbb{N}$:

$$red \in A_i \implies i \geq 1 \text{ and } yellow \in A_{i-1}$$



"there is a red phase that is not preceded by a yellow phase"

Safety property for a traffic light

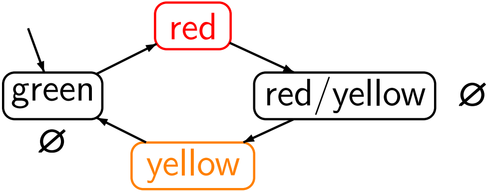


"every red phase is preceded by a yellow phase"

hence: $\mathcal{T} \models E$

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

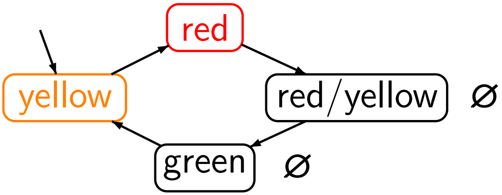
$$red \in A_i \implies i \geq 1 \text{ and } yellow \in A_{i-1}$$



"there is a red phase that is not preceded by a yellow phase"

hence: $\mathcal{T} \not\models E$

Safety property for a traffic light

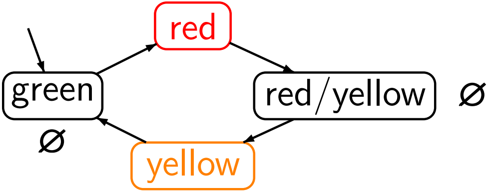


"every red phase is preceded by a yellow phase"

hence: $\mathcal{T} \models E$

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

$$\mathit{red} \in A_i \implies i \geq 1 \text{ and } \mathit{yellow} \in A_{i-1}$$

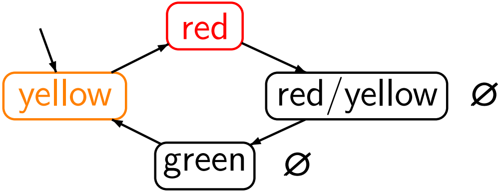


$\mathcal{T} \not\models E$

bad prefix, e.g.,

$\varnothing\,\{\mathit{red}\}\,\varnothing\,\{\mathit{yellow}\}$

Safety property for a traffic light

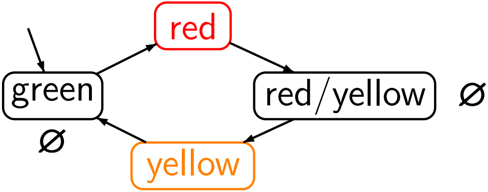


"every red phase is preceded by a yellow phase"

hence: $\mathcal{T} \models E$

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

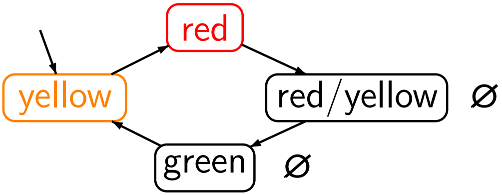
$$red \in A_i \implies i \geq 1 \text{ and } yellow \in A_{i-1}$$



$\mathcal{T} \not\models E$

minimal bad prefix: $\varnothing\,\{red\}$

Safety property for a traffic light



"every red phase is preceded by a yellow phase"

hence: $\mathcal{T} \models E$

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

$$red \in A_i \implies i \geq 1 \text{ and } yellow \in A_{i-1}$$

is a safety property over $AP = \{red, yellow\}$ with

$BadPref$ = set of all finite words $A_0\, A_1 \ldots A_n$ over 2^{AP} s.t. for some $i \in \{0, \ldots, n\}$:

$$red \in A_i \wedge (i{=}0 \vee yellow \notin A_{i-1})$$

Satisfaction of safety properties

IS2.5-11A

Let $E \subseteq (2^{AP})^\omega$ be a safety property, $\mathcal{T}$ a TS over AP.

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

$\mathit{Traces}(\mathcal{T}) \quad = \quad$ set of traces of $\mathcal{T}$

Satisfaction of safety properties

Let $E \subseteq (2^{AP})^\omega$ be a safety property, $\mathcal{T}$ a TS over AP.

$$
\begin{aligned}
\mathcal{T} \models E \quad & \text{iff} \quad Traces(\mathcal{T}) \subseteq E \\
& \text{iff} \quad Traces_{fin}(\mathcal{T}) \cap BadPref = \varnothing
\end{aligned}
$$

$BadPref$ = set of all bad prefixes of E

$Traces(\mathcal{T})$ = set of traces of $\mathcal{T}$

$Traces_{fin}(\mathcal{T})$ = set of finite traces of $\mathcal{T}$

$= \{ \, trace(\widehat{\pi}) : \widehat{\pi}$ is an initial, finite path fragment of $\mathcal{T} \}$

Satisfaction of safety properties

Let $E \subseteq (2^{AP})^\omega$ be a safety property, $\mathcal{T}$ a TS over AP.

$$\begin{aligned} \mathcal{T} \models E \quad & \text{iff} \quad Traces(\mathcal{T}) \subseteq E \\ & \text{iff} \quad Traces_{fin}(\mathcal{T}) \cap BadPref = \varnothing \\ & \text{iff} \quad Traces_{fin}(\mathcal{T}) \cap MinBadPref = \varnothing \end{aligned}$$

$BadPref$ = set of all bad prefixes of E

$MinBadPref$ = set of all minimal bad prefixes of E

$Traces(\mathcal{T})$ = set of traces of $\mathcal{T}$

$Traces_{fin}(\mathcal{T})$ = set of finite traces of $\mathcal{T}$

$= \{ \, trace(\widehat{\pi}) : \widehat{\pi}$ is an initial, finite path fragment of $\mathcal{T} \}$

Correct or wrong?

Every invariant is a safety property.

Correct or wrong?

Every invariant is a safety property.

correct.

Correct or wrong?

Every invariant is a safety property.

correct.

Let E be an invariant with invariant condition Φ.

Every invariant is a safety property.

correct.

Let E be an invariant with invariant condition Φ.

- bad prefixes for E: finite words $A_0 \ldots A_i \ldots A_n$ s.t.

$$A_i \not\models \Phi \text{ for some } i \in \{0, 1, \ldots, n\}$$

Every invariant is a safety property.

correct.

Let E be an invariant with invariant condition Φ.

- bad prefixes for E: finite words $A_0 \ldots A_i \ldots A_n$ s.t.

$$A_i \not\models \Phi \text{ for some } i \in \{0, 1, \ldots, n\}$$

- minimal bad prefixes for E:
  finite words $A_0\, A_1 \ldots A_{n-1}\, A_n$ such that

$$A_i \models \Phi \text{ for } i = 0, 1, \ldots, n{-}1, \text{ and } A_n \not\models \Phi$$

Correct or wrong?

$\varnothing$ is a safety property

Correct or wrong?

$\varnothing$ is a safety property

correct

Correct or wrong?

$\varnothing$ is a safety property

correct

- all finite words $A_0 \ldots A_n \in (2^{AP})^+$ are bad prefixes

Correct or wrong?

$\varnothing$ is a safety property

correct

- all finite words $A_0 \dots A_n \in (2^{AP})^+$ are bad prefixes
- $\varnothing$ is even an invariant (invariant condition ***false***)

$\varnothing$ is a safety property

correct

- all finite words $A_0 \ldots A_n \in (2^{AP})^+$ are bad prefixes
- $\varnothing$ is even an invariant (invariant condition ***false***)

$(2^{AP})^\omega$ is a safety property

$\varnothing$ is a safety property

correct

- all finite words $A_0 \ldots A_n \in (2^{AP})^+$ are bad prefixes
- $\varnothing$ is even an invariant (invariant condition ***false***)

$(2^{AP})^\omega$ is a safety property

correct

$\varnothing$ is a safety property

correct

- all finite words $A_0 \dots A_n \in (2^{AP})^+$ are bad prefixes
- $\varnothing$ is even an invariant (invariant condition ***false***)

$(2^{AP})^\omega$ is a safety property

correct

$$\text{“For all words} \in \underbrace{(2^{AP})^\omega \setminus (2^{AP})^\omega}_{=\varnothing} \dots\text{”}$$

Prefix closure

Prefix closure

For a given infinite word $\sigma = A_0 A_1 A_2 \ldots$, let

$$pref(\sigma) \stackrel{\text{def}}{=} \text{set of all nonempty, finite prefixes of } \sigma$$

$$= \{ A_0 A_1 \ldots A_n : n \geq 0 \}$$

Prefix closure

For a given infinite word $\sigma = A_0\, A_1\, A_2\, \ldots$, let

$$pref(\sigma) \overset{\text{def}}{=} \text{set of all nonempty, finite prefixes of } \sigma$$
$$= \{ A_0\, A_1 \ldots A_n : n \geq 0 \}$$

For $E \subseteq (2^{AP})^\omega$, let $pref(E) \overset{\text{def}}{=} \bigcup_{\sigma \in E} pref(\sigma)$

Prefix closure

For a given infinite word $\sigma = A_0\, A_1\, A_2\, \ldots$, let

$$pref(\sigma) \stackrel{\text{def}}{=} \text{set of all nonempty, finite prefixes of } \sigma$$
$$= \{ A_0\, A_1 \ldots A_n : n \geq 0 \}$$

For $E \subseteq (2^{AP})^\omega$, let $pref(E) \stackrel{\text{def}}{=} \bigcup_{\sigma \in E} pref(\sigma)$

Given an LT property E, the prefix closure of E is:

$$cl(E) \stackrel{\text{def}}{=} \{\sigma \in (2^{AP})^\omega : pref(\sigma) \subseteq pref(E)\}$$

Prefix closure and safety

For any infinite word $\sigma \in \left(2^{AP}\right)^{\omega}$, let

$$pref(\sigma) \;=\; \text{set of all nonempty, finite prefixes of } \sigma$$

For any LT property $E \subseteq \left(2^{AP}\right)^{\omega}$, let

$$pref(E) \;=\; \bigcup_{\sigma \in E} pref(\sigma) \text{ and}$$

$$cl(E) \;=\; \{\sigma \in (2^{AP})^{\omega} : pref(\sigma) \subseteq pref(E)\}$$

Prefix closure and safety

For any infinite word $\sigma \in \left(2^{AP}\right)^{\omega}$, let

$$pref(\sigma) \;=\; \text{set of all nonempty, finite prefixes of } \sigma$$

For any LT property $E \subseteq \left(2^{AP}\right)^{\omega}$, let

$$pref(E) \;=\; \bigcup_{\sigma \in E} pref(\sigma) \text{ and}$$

$$cl(E) \;=\; \left\{\sigma \in (2^{AP})^{\omega} : pref(\sigma) \subseteq pref(E)\right\}$$

Theorem:

E is a safety property iff $cl(E) = E$

Safety and finite trace inclusion

remind: LT properties and trace inclusion:

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

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

iff for all LT properties E: $\mathcal{T}_2 \models E \implies \mathcal{T}_1 \models E$

Safety and finite trace inclusion

remind: LT properties and trace inclusion:

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

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

iff for all LT properties E: $\mathcal{T}_2 \models E \implies \mathcal{T}_1 \models E$

safety properties and finite trace inclusion:

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

$$Traces_{fin}(\mathcal{T}_1) \subseteq Traces_{fin}(\mathcal{T}_2)$$

iff for all safety properties E: $\mathcal{T}_2 \models E \implies \mathcal{T}_1 \models E$

Safety and finite trace inclusion

$$\mathit{Traces}_{\mathit{fin}}(\mathcal{T}_1) \subseteq \mathit{Traces}_{\mathit{fin}}(\mathcal{T}_2)$$

iff for all safety properties E: $\mathcal{T}_2 \models E \implies \mathcal{T}_1 \models E$

Safety and finite trace inclusion

$$Traces_{fin}(\mathcal{T}_1) \subseteq Traces_{fin}(\mathcal{T}_2)$$

iff for all safety properties E: $\mathcal{T}_2 \models E \implies \mathcal{T}_1 \models E$

Proof "$\Longrightarrow$": obvious, as for safety property E:

$$\mathcal{T} \models E \quad \text{iff} \quad Traces_{fin}(\mathcal{T}) \cap BadPref = \varnothing$$

Safety and finite trace inclusion

$$Traces_{fin}(\mathcal{T}_1) \subseteq Traces_{fin}(\mathcal{T}_2)$$

iff for all safety properties E: $\mathcal{T}_2 \models E \implies \mathcal{T}_1 \models E$

Proof "$\Longrightarrow$": obvious, as for safety property E:

$$\mathcal{T} \models E \quad \text{iff} \quad Traces_{fin}(\mathcal{T}) \cap BadPref = \varnothing$$

Hence:

If $\mathcal{T}_2 \models E$ and $Traces_{fin}(\mathcal{T}_1) \subseteq Traces_{fin}(\mathcal{T}_2)$ then:

Safety and finite trace inclusion

$$Traces_{fin}(\mathcal{T}_1) \subseteq Traces_{fin}(\mathcal{T}_2)$$

iff for all safety properties E: $\mathcal{T}_2 \models E \implies \mathcal{T}_1 \models E$

Proof "$\Longrightarrow$": obvious, as for safety property E:

$$\mathcal{T} \models E \quad \text{iff} \quad Traces_{fin}(\mathcal{T}) \cap BadPref = \varnothing$$

Hence:

If $\mathcal{T}_2 \models E$ and $Traces_{fin}(\mathcal{T}_1) \subseteq Traces_{fin}(\mathcal{T}_2)$ then:

$$\begin{aligned} & Traces_{fin}(\mathcal{T}_1) \cap BadPref \\ \subseteq\ & Traces_{fin}(\mathcal{T}_2) \cap BadPref = \varnothing \end{aligned}$$

Safety and finite trace inclusion

$$Traces_{fin}(\mathcal{T}_1) \subseteq Traces_{fin}(\mathcal{T}_2)$$

iff for all safety properties E: $\mathcal{T}_2 \models E \implies \mathcal{T}_1 \models E$

Proof "$\Longrightarrow$": obvious, as for safety property E:

$$\mathcal{T} \models E \quad \text{iff} \quad Traces_{fin}(\mathcal{T}) \cap BadPref = \varnothing$$

Hence:

If $\mathcal{T}_2 \models E$ and $Traces_{fin}(\mathcal{T}_1) \subseteq Traces_{fin}(\mathcal{T}_2)$ then:

$$\begin{aligned} & Traces_{fin}(\mathcal{T}_1) \cap BadPref \\ \subseteq \; & Traces_{fin}(\mathcal{T}_2) \cap BadPref = \varnothing \end{aligned}$$

and therefore $\mathcal{T}_1 \models E$

Safety and finite trace inclusion

$$Traces_{fin}(\mathcal{T}_1) \subseteq Traces_{fin}(\mathcal{T}_2)$$

iff for all safety properties E: $\mathcal{T}_2 \models E \implies \mathcal{T}_1 \models E$

Proof "$\Longleftarrow$": consider the LT property

$$E = cl(Traces(\mathcal{T}_2))$$

Safety and finite trace inclusion

$$Traces_{fin}(\mathcal{T}_1) \subseteq Traces_{fin}(\mathcal{T}_2)$$

iff for all safety properties E: $\mathcal{T}_2 \models E \implies \mathcal{T}_1 \models E$

Proof "$\Longleftarrow$": consider the LT property

$$E = cl(Traces(\mathcal{T}_2)) = \{\sigma : pref(\sigma) \subseteq Traces_{fin}(\mathcal{T}_2)\}$$

Safety and finite trace inclusion

$$Traces_{fin}(\mathcal{T}_1) \subseteq Traces_{fin}(\mathcal{T}_2)$$

iff for all safety properties E: $\mathcal{T}_2 \models E \implies \mathcal{T}_1 \models E$

Proof "$\Longleftarrow$": consider the LT property

$$E = cl(Traces(\mathcal{T}_2)) = \{\sigma : pref(\sigma) \subseteq Traces_{fin}(\mathcal{T}_2)\}$$

$\uparrow$

for each transition system $\mathcal{T}$:

$$pref(Traces(\mathcal{T})) = Traces_{fin}(\mathcal{T})$$

$$Traces_{fin}(\mathcal{T}_1) \subseteq Traces_{fin}(\mathcal{T}_2)$$

iff for all safety properties E: $\mathcal{T}_2 \models E \implies \mathcal{T}_1 \models E$

Proof "$\Longleftarrow$": consider the LT property

$$E = cl(Traces(\mathcal{T}_2)) = \{\sigma : pref(\sigma) \subseteq Traces_{fin}(\mathcal{T}_2)\}$$

Then, E is a safety property

Safety and finite trace inclusion

$$\mathit{Traces}_{fin}(\mathcal{T}_1) \subseteq \mathit{Traces}_{fin}(\mathcal{T}_2)$$

iff for all safety properties E: $\mathcal{T}_2 \models E \implies \mathcal{T}_1 \models E$

Proof "$\Longleftarrow$": consider the LT property

$$E = cl(\mathit{Traces}(\mathcal{T}_2)) = \{\sigma : \mathit{pref}(\sigma) \subseteq \mathit{Traces}_{fin}(\mathcal{T}_2)\}$$

Then, E is a safety property

↑

as $cl(E) = E$

Safety and finite trace inclusion

$$Traces_{fin}(\mathcal{T}_1) \subseteq Traces_{fin}(\mathcal{T}_2)$$

iff for all safety properties E: $\mathcal{T}_2 \models E \implies \mathcal{T}_1 \models E$

Proof "$\Longleftarrow$": consider the LT property

$$E = cl(Traces(\mathcal{T}_2)) = \{\sigma : pref(\sigma) \subseteq Traces_{fin}(\mathcal{T}_2)\}$$

Then, E is a safety property

$\uparrow$

as $cl(E) = E$

set of bad prefixes: $(2^{AP})^+ \setminus Traces_{fin}(\mathcal{T}_2)$

Safety and finite trace inclusion

$$Traces_{fin}(\mathcal{T}_1) \subseteq Traces_{fin}(\mathcal{T}_2)$$

iff for all safety properties E: $\mathcal{T}_2 \models E \implies \mathcal{T}_1 \models E$

Proof "$\Longleftarrow$": consider the LT property

$$E = cl(Traces(\mathcal{T}_2)) = \{\sigma : pref(\sigma) \subseteq Traces_{fin}(\mathcal{T}_2)\}$$

Then, E is a safety property and $\mathcal{T}_2 \models E$.

Safety and finite trace inclusion

$$Traces_{fin}(\mathcal{T}_1) \subseteq Traces_{fin}(\mathcal{T}_2)$$

iff for all safety properties E: $\mathcal{T}_2 \models E \implies \mathcal{T}_1 \models E$

Proof "$\Longleftarrow$": consider the LT property

$$E = cl(Traces(\mathcal{T}_2)) = \{\sigma : pref(\sigma) \subseteq Traces_{fin}(\mathcal{T}_2)\}$$

Then, E is a safety property and $\mathcal{T}_2 \models E$.

By assumption: $\mathcal{T}_1 \models E$

Safety and finite trace inclusion

$$Traces_{fin}(\mathcal{T}_1) \subseteq Traces_{fin}(\mathcal{T}_2)$$

iff for all safety properties E: $\mathcal{T}_2 \models E \implies \mathcal{T}_1 \models E$

Proof "$\Longleftarrow$": consider the LT property

$$E = cl(Traces(\mathcal{T}_2)) = \{\sigma : pref(\sigma) \subseteq Traces_{fin}(\mathcal{T}_2)\}$$

Then, E is a safety property and $\mathcal{T}_2 \models E$.

By assumption: $\mathcal{T}_1 \models E$ and therefore $Traces(\mathcal{T}_1) \subseteq E$.

Safety and finite trace inclusion

$$Traces_{fin}(\mathcal{T}_1) \subseteq Traces_{fin}(\mathcal{T}_2)$$

iff for all safety properties E: $\mathcal{T}_2 \models E \implies \mathcal{T}_1 \models E$

Proof "$\Longleftarrow$": consider the LT property

$$E = cl(Traces(\mathcal{T}_2)) = \{\sigma : pref(\sigma) \subseteq Traces_{fin}(\mathcal{T}_2)\}$$

Then, E is a safety property and $\mathcal{T}_2 \models E$.

By assumption: $\mathcal{T}_1 \models E$ and therefore $Traces(\mathcal{T}_1) \subseteq E$.

Hence: $Traces_{fin}(\mathcal{T}_1) = pref(Traces(\mathcal{T}_1))$

Safety and finite trace inclusion

$$Traces_{fin}(\mathcal{T}_1) \subseteq Traces_{fin}(\mathcal{T}_2)$$

iff for all safety properties E: $\mathcal{T}_2 \models E \implies \mathcal{T}_1 \models E$

Proof "$\Longleftarrow$": consider the LT property

$$E = cl(Traces(\mathcal{T}_2)) = \{\sigma : pref(\sigma) \subseteq Traces_{fin}(\mathcal{T}_2)\}$$

Then, E is a safety property and $\mathcal{T}_2 \models E$.

By assumption: $\mathcal{T}_1 \models E$ and therefore $Traces(\mathcal{T}_1) \subseteq E$.

Hence: $Traces_{fin}(\mathcal{T}_1) = pref(Traces(\mathcal{T}_1))$

$$\subseteq pref(E)$$

Safety and finite trace inclusion

$$Traces_{fin}(\mathcal{T}_1) \subseteq Traces_{fin}(\mathcal{T}_2)$$

iff for all safety properties E: $\mathcal{T}_2 \models E \implies \mathcal{T}_1 \models E$

Proof "$\Longleftarrow$": consider the LT property

$$E = cl(Traces(\mathcal{T}_2)) = \{\sigma : pref(\sigma) \subseteq Traces_{fin}(\mathcal{T}_2)\}$$

Then, E is a safety property and $\mathcal{T}_2 \models E$.

By assumption: $\mathcal{T}_1 \models E$ and therefore $Traces(\mathcal{T}_1) \subseteq E$.

Hence: $Traces_{fin}(\mathcal{T}_1) = pref(Traces(\mathcal{T}_1))$

$$\subseteq pref(E) = pref(cl(Traces(\mathcal{T}_2)))$$

Safety and finite trace inclusion

$$Traces_{fin}(\mathcal{T}_1) \subseteq Traces_{fin}(\mathcal{T}_2)$$

iff for all safety properties E: $\mathcal{T}_2 \models E \implies \mathcal{T}_1 \models E$

Proof "$\Longleftarrow$": consider the LT property

$$E = cl(Traces(\mathcal{T}_2)) = \{\sigma : pref(\sigma) \subseteq Traces_{fin}(\mathcal{T}_2)\}$$

Then, E is a safety property and $\mathcal{T}_2 \models E$.

By assumption: $\mathcal{T}_1 \models E$ and therefore $Traces(\mathcal{T}_1) \subseteq E$.

Hence:
$$\begin{aligned} Traces_{fin}(\mathcal{T}_1) &= pref(Traces(\mathcal{T}_1)) \\ &\subseteq pref(E) = pref(cl(Traces(\mathcal{T}_2))) \\ &= Traces_{fin}(\mathcal{T}_2) \end{aligned}$$

Safety and finite trace equivalence

safety properties and finite trace inclusion:

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

$$Traces_{fin}(\mathcal{T}_1) \subseteq Traces_{fin}(\mathcal{T}_2)$$

iff for all safety properties E: $\mathcal{T}_2 \models E \implies \mathcal{T}_1 \models E$

Safety and finite trace equivalence

safety properties and finite trace inclusion:

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

$$Traces_{fin}(\mathcal{T}_1) \subseteq Traces_{fin}(\mathcal{T}_2)$$

iff for all safety properties E: $\mathcal{T}_2 \models E \implies \mathcal{T}_1 \models E$

safety properties and finite trace equivalence:

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

$$Traces_{fin}(\mathcal{T}_1) = Traces_{fin}(\mathcal{T}_2)$$

iff $\mathcal{T}_1$ and $\mathcal{T}_2$ satisfy the same safety properties

Summary: trace relations and properties

trace inclusion

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

for all LT properties E: $\mathcal{T}' \models E \Longrightarrow \mathcal{T} \models E$

finite trace inclusion

$Traces_{fin}(\mathcal{T}) \subseteq Traces_{fin}(\mathcal{T}')$ iff

for all safety properties E: $\mathcal{T}' \models E \Longrightarrow \mathcal{T} \models E$

Summary: trace relations and properties

trace equivalence

$Traces(\mathcal{T}) = Traces(\mathcal{T}')$ iff

$\mathcal{T}$ and $\mathcal{T}'$ satisfy the same LT properties

finite trace equivalence

$Traces_{fin}(\mathcal{T}) = Traces_{fin}(\mathcal{T}')$ iff

$\mathcal{T}$ and $\mathcal{T}'$ satisfy the same safety properties

correct or wrong?

If $\mathit{Traces}(\mathcal{T}) \subseteq \mathit{Traces}(\mathcal{T}')$
then $\mathit{Traces}_{\mathit{fin}}(\mathcal{T}) \subseteq \mathit{Traces}_{\mathit{fin}}(\mathcal{T}')$.

> If $\mathit{Traces}(\mathcal{T}) \subseteq \mathit{Traces}(\mathcal{T}')$
> then $\mathit{Traces}_{\mathit{fin}}(\mathcal{T}) \subseteq \mathit{Traces}_{\mathit{fin}}(\mathcal{T}')$.

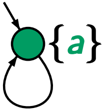
correct, since

$$\begin{aligned} \mathit{Traces}_{\mathit{fin}}(\mathcal{T}) &= \text{set of all finite nonempty prefixes of words in } \mathit{Traces}(\mathcal{T}) \\ &= \mathit{pref}(\mathit{Traces}(\mathcal{T})) \end{aligned}$$

If $Traces(\mathcal{T}) \subseteq Traces(\mathcal{T}')$
then $Traces_{fin}(\mathcal{T}) \subseteq Traces_{fin}(\mathcal{T}')$.

correct, since

$$\begin{aligned} Traces_{fin}(\mathcal{T}) &= \text{set of all finite nonempty prefixes of words in } Traces(\mathcal{T}) \\ &= pref(Traces(\mathcal{T})) \end{aligned}$$



$$\begin{aligned} Traces(\mathcal{T}) &= \{ \{a\}^\omega \} \\ Traces_{fin}(\mathcal{T}) &= \{ \{a\}^n : n \geq 1 \} \end{aligned}$$

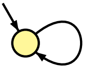
is trace equivalence the same as
finite trace equivalence **?**

is trace equivalence the same as
finite trace equivalence **?**

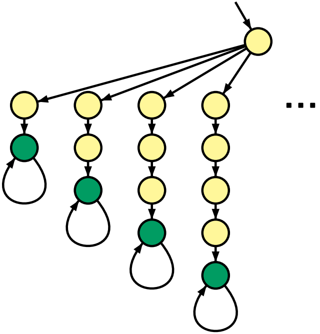
answer: **no**

Finite trace relations versus trace relations

$\mathcal{T}$



$\mathcal{T}'$

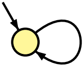


$\bigcirc \ \widehat{=} \ \varnothing \qquad \bullet \ \widehat{=} \ \{b\}$

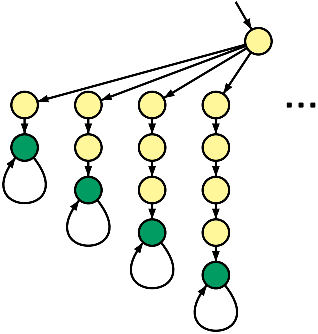
set of propositions

$AP = \{b\}$

Finite trace relations versus trace relations



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



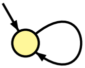
 $\widehat{=} \varnothing$  $\widehat{=} \{b\}$

set of propositions

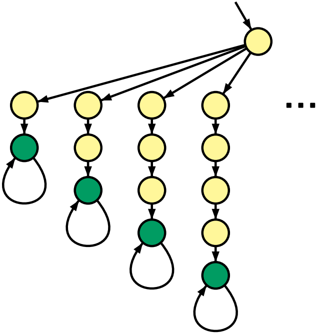
$AP = \{b\}$

Finite trace relations versus trace relations

$\mathcal{T}$



$\mathcal{T}'$

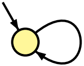


$$Traces(\mathcal{T}) = \{\varnothing^\omega\}$$
$$Traces_{fin}(\mathcal{T}) = \{\varnothing^n : n \geq 0\}$$

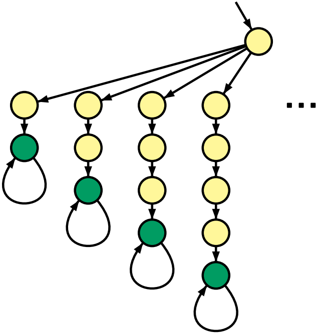
set of propositions
$AP = \{b\}$

Finite trace relations versus trace relations

$\mathcal{T}$



$\mathcal{T}'$



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

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

$$Traces(\mathcal{T}') = \{\varnothing^n \{b\}^\omega : n \geq 2\}$$

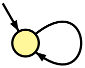
 $\widehat{=} \varnothing$  $\widehat{=} \{b\}$

set of propositions

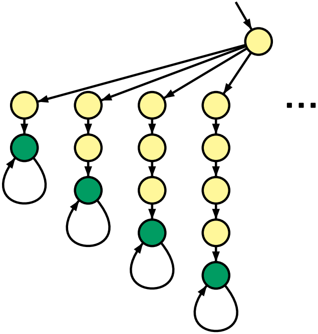
$AP = \{b\}$

Finite trace relations versus trace relations

$\mathcal{T}$



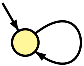
$\mathcal{T}'$



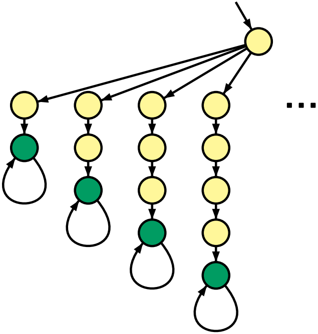
$$
\begin{aligned}
Traces(\mathcal{T}) &= \{\varnothing^\omega\} \\
Traces_{fin}(\mathcal{T}) &= \{\varnothing^n : n \geq 0\} \\
Traces(\mathcal{T}') &= \{\varnothing^n \{b\}^\omega : n \geq 2\} \\
Traces_{fin}(\mathcal{T}') &= \{\varnothing^n : n \geq 0\} \ \cup \\
&\quad \{\varnothing^n \{b\}^m : n \geq 2 \wedge m \geq 1\}
\end{aligned}
$$

Finite trace relations versus trace relations

$\mathcal{T}$



$\mathcal{T}'$



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

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

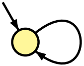
$$Traces(\mathcal{T}') = \{\varnothing^n\{b\}^\omega : n \geq 2\}$$

$$Traces_{fin}(\mathcal{T}') = \{\varnothing^n : n \geq 0\} \cup \{\varnothing^n\{b\}^m : n \geq 2 \wedge m \geq 1\}$$

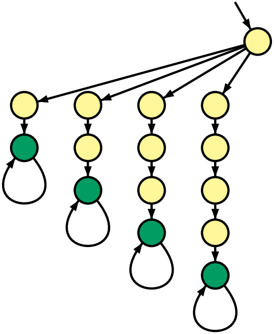
$Traces(\mathcal{T}) \not\subseteq Traces(\mathcal{T}')$, but
$Traces_{fin}(\mathcal{T}) \subseteq Traces_{fin}(\mathcal{T}')$

Finite trace relations versus trace relations

$\mathcal{T}$



$\mathcal{T}'$



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

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

$$Traces(\mathcal{T}') = \{\varnothing^n\{b\}^\omega : n \geq 2\}$$

$$Traces_{fin}(\mathcal{T}') = \{\varnothing^n : n \geq 0\} \cup \{\varnothing^n\{b\}^m : n \geq 2 \wedge m \geq 1\}$$

$Traces(\mathcal{T}) \not\subseteq Traces(\mathcal{T}')$, but
$Traces_{fin}(\mathcal{T}) \subseteq Traces_{fin}(\mathcal{T}')$

LT property
$E \mathrel{\hat{=}}$ "eventually b"

$\mathcal{T} \not\models E$, $\quad \mathcal{T}' \models E$

Finite trace and trace inclusion

Suppose that $\mathcal{T}$ and $\mathcal{T}'$ are TS over AP such that

(1) $\mathcal{T}$ has no terminal states,

(2) $\mathcal{T}'$ is finite.

Finite trace and trace inclusion

Suppose that $\mathcal{T}$ and $\mathcal{T}'$ are TS over AP such that

(1) $\mathcal{T}$ has no terminal states,
i.e., all paths of $\mathcal{T}$ are infinite

(2) $\mathcal{T}'$ is finite.

Finite trace and trace inclusion

Suppose that $\mathcal{T}$ and $\mathcal{T}'$ are TS over ***AP*** such that

(1) $\mathcal{T}$ has no terminal states,
i.e., all paths of $\mathcal{T}$ are infinite

(2) $\mathcal{T}'$ is finite.

Then:

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

$$\text{iff} \quad Traces_{fin}(\mathcal{T}) \subseteq Traces_{fin}(\mathcal{T}')$$

Finite trace and trace inclusion

Suppose that $\mathcal{T}$ and $\mathcal{T}'$ are TS over AP such that

(1) $\mathcal{T}$ has no terminal states,
i.e., all paths of $\mathcal{T}$ are infinite

(2) $\mathcal{T}'$ is finite.

Then:
$$Traces(\mathcal{T}) \subseteq Traces(\mathcal{T}')$$
$$\text{iff} \quad Traces_{fin}(\mathcal{T}) \subseteq Traces_{fin}(\mathcal{T}')$$

"$\Longrightarrow$": holds for all transition systems,
no matter whether (1) and (2) hold

Finite trace and trace inclusion

Suppose that $\mathcal{T}$ and $\mathcal{T}'$ are TS over AP such that

(1) $\mathcal{T}$ has no terminal states,
i.e., all paths of $\mathcal{T}$ are infinite

(2) $\mathcal{T}'$ is finite.

Then:
$$Traces(\mathcal{T}) \subseteq Traces(\mathcal{T}')$$
$$\text{iff} \quad Traces_{fin}(\mathcal{T}) \subseteq Traces_{fin}(\mathcal{T}')$$

"$\Longrightarrow$": holds for all transition systems

"$\Longleftarrow$": suppose that (1) and (2) hold and that

(3) $Traces_{fin}(\mathcal{T}) \subseteq Traces_{fin}(\mathcal{T}')$

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

Finite trace and trace inclusion

Suppose that $\mathcal{T}$ and $\mathcal{T}'$ are TS over AP such that

(1) $\mathcal{T}$ has no terminal states

(2) $\mathcal{T}'$ is finite

(3) $Traces_{fin}(\mathcal{T}) \subseteq Traces_{fin}(\mathcal{T}')$

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

Proof:

Finite trace and trace inclusion

Suppose that $\mathcal{T}$ and $\mathcal{T}'$ are TS over AP such that

(1) $\mathcal{T}$ has no terminal states

(2) $\mathcal{T}'$ is finite

(3) $Traces_{fin}(\mathcal{T}) \subseteq Traces_{fin}(\mathcal{T}')$

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

Proof: Pick some path $\pi = s_0\, s_1\, s_2 \ldots$ in $\mathcal{T}$ and show that there exists a path

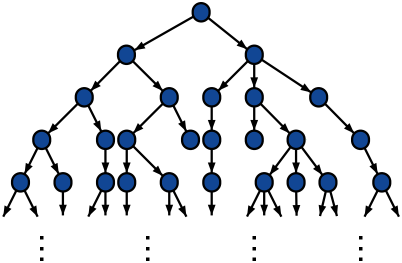
$$\pi' = t_0\, t_1\, t_2 \ldots \text{ in } \mathcal{T}'$$

such that $trace(\pi) = trace(\pi')$

Tracesfin versus traces

finite TS $\mathcal{T}'$

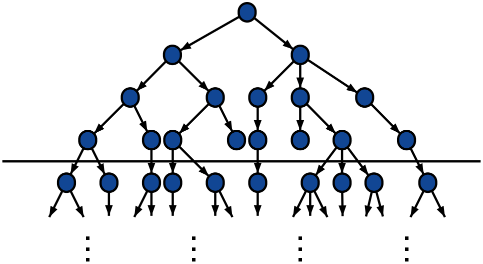
paths from state t_0
(unfolded into a tree)



Tracesfin versus traces

finite TS $\mathcal{T}'$

paths from state t_0
(unfolded into a tree)

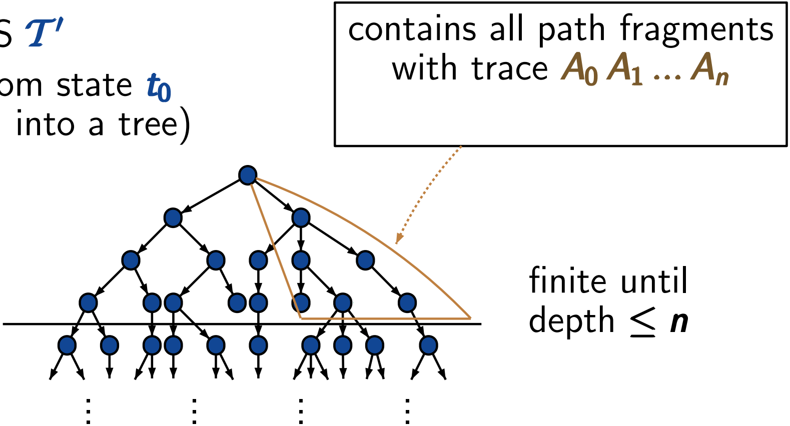


finite until
depth $\leq n$

Tracesfin versus traces

finite TS $\mathcal{T}'$

paths from state t_0
(unfolded into a tree)

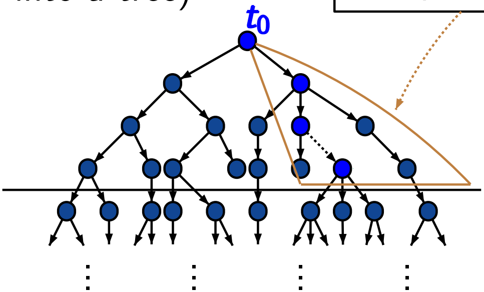


Tracesfin versus traces

finite TS $\mathcal{T}'$

paths from state t_0
(unfolded into a tree)

contains all path fragments with trace $A_0\,A_1 \ldots A_n$
in particular: $t_0\,t_1 \ldots t_n$

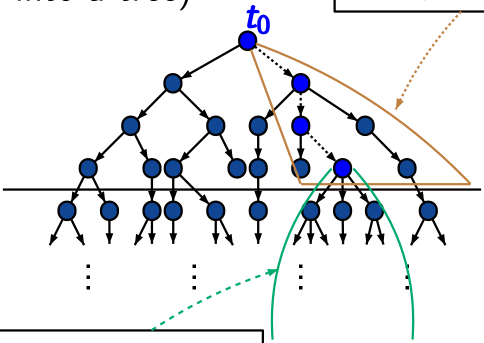


finite until depth $\leq n$

finite TS $\mathcal{T}'$

paths from state t_0
(unfolded into a tree)

contains all path fragments
with trace $A_0\, A_1 \ldots A_n$
in particular: $t_0\, t_1 \ldots t_n$

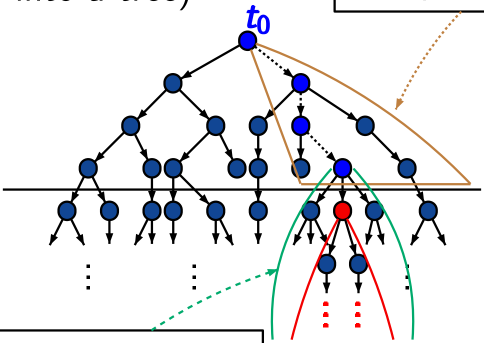


finite until
depth $\leq n$

contains infinitely
many path fragments
$t_n\, s^m_{n+1} \ldots s^m_m$

finite TS $\mathcal{T}'$

paths from state t_0
(unfolded into a tree)



contains all path fragments with trace $A_0\, A_1 \ldots A_n$
in particular: $t_0\, t_1 \ldots t_n$

contains infinitely many path fragments
$t_n\, s_{n+1}^m \ldots s_m^m$

there exists $t_{n+1} \in Post(t_n)$
s.t. $t_{n+1} = s_{n+1}^m$ for
infinitely many m

Finite trace and trace inclusion

Suppose that $\mathcal{T}$ and $\mathcal{T}'$ are TS over AP such that

(1) $\mathcal{T}$ has no terminal states

(2) $\mathcal{T}'$ is finite ⟵ image-finiteness is sufficient

(3) $Traces_{fin}(\mathcal{T}) \subseteq Traces_{fin}(\mathcal{T}')$

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

Finite trace and trace inclusion

Suppose that $\mathcal{T}$ and $\mathcal{T}'$ are TS over AP such that

(1) $\mathcal{T}$ has no terminal states

(2) $\mathcal{T}'$ is finite ⟵ image-finiteness is sufficient

(3) $Traces_{fin}(\mathcal{T}) \subseteq Traces_{fin}(\mathcal{T}')$

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

image-finiteness of $\mathcal{T}' = (S', Act, \rightarrow, S'_0, AP, L')$:

Finite trace and trace inclusion

Suppose that $\mathcal{T}$ and $\mathcal{T}'$ are TS over AP such that

(1) $\mathcal{T}$ has no terminal states

(2) $\mathcal{T}'$ is finite ⟵ image-finiteness is sufficient

(3) $Traces_{fin}(\mathcal{T}) \subseteq Traces_{fin}(\mathcal{T}')$

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

image-finiteness of $\mathcal{T}' = (S', Act, \rightarrow, S'_0, AP, L')$:

- for each $A \in 2^{AP}$ and state $s \in S'$:

$$\{t \in Post(s) : L'(t) = A\} \text{ is finite}$$

Finite trace and trace inclusion

Suppose that $\mathcal{T}$ and $\mathcal{T}'$ are TS over AP such that

(1) $\mathcal{T}$ has no terminal states

(2) $\mathcal{T}'$ is finite ⟵ image-finiteness is sufficient

(3) $Traces_{fin}(\mathcal{T}) \subseteq Traces_{fin}(\mathcal{T}')$

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

image-finiteness of $\mathcal{T}' = (S', Act, \rightarrow, S'_0, AP, L')$:

- for each $A \in 2^{AP}$ and state $s \in S'$:

  $\{t \in Post(s) : L'(t) = A\}$ is finite

- for each $A \in 2^{AP}$: $\{s_0 \in S'_0 : L'(s_0) = A\}$ is finite

Trace equivalence vs. finite trace equivalence

Whenever $\mathit{Traces}(\mathcal{T}) = \mathit{Traces}(\mathcal{T}')$ then
$\mathit{Traces}_{\mathit{fin}}(\mathcal{T}) = \mathit{Traces}_{\mathit{fin}}(\mathcal{T}')$

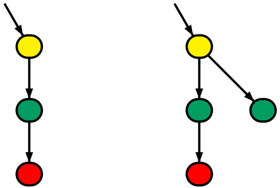
Whenever $\mathit{Traces}(\mathcal{T}) = \mathit{Traces}(\mathcal{T}')$ then
$\mathit{Traces}_{\mathit{fin}}(\mathcal{T}) = \mathit{Traces}_{\mathit{fin}}(\mathcal{T}')$

while the reverse direction does not hold in general
(even not for finite transition systems)

Trace equivalence vs. finite trace equivalence

Whenever $\mathit{Traces}(\mathcal{T}) = \mathit{Traces}(\mathcal{T}')$ then
$\mathit{Traces}_{\mathit{fin}}(\mathcal{T}) = \mathit{Traces}_{\mathit{fin}}(\mathcal{T}')$

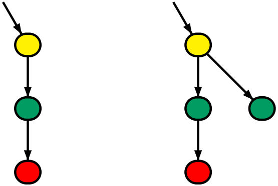
while the reverse direction does not hold in general (even not for finite transition systems)



Trace equivalence vs. finite trace equivalence

Whenever $Traces(\mathcal{T}) = Traces(\mathcal{T}')$ then
$Traces_{fin}(\mathcal{T}) = Traces_{fin}(\mathcal{T}')$

while the reverse direction does not hold in general (even not for finite transition systems)



finite trace equivalent,

but *not* trace equivalent

Trace equivalence vs. finite trace equivalence

Whenever $\mathit{Traces}(\mathcal{T}) = \mathit{Traces}(\mathcal{T}')$ then
$\mathit{Traces}_{\mathit{fin}}(\mathcal{T}) = \mathit{Traces}_{\mathit{fin}}(\mathcal{T}')$

The reverse implication holds under additional assumptions, e.g.,

- if $\mathcal{T}$ and $\mathcal{T}'$ are finite and have no terminal states
- or, if $\mathcal{T}$ and $\mathcal{T}'$ are AP-deterministic

Overview

"liveness: something good will happen."

"event ***a*** will occur eventually"

e.g., termination for sequential programs

"event ***a*** will occur infinitely many times"

e.g., starvation freedom for dining philosophers

"whenever event ***b*** occurs then event ***a***
will occur sometimes in the future"

e.g., every waiting process enters eventually
its critical section

which property type?

- Each philosopher thinks infinitely often.

which property type?

- Each philosopher thinks infinitely often.

liveness

which property type? LF2.6-2

- Each philosopher thinks infinitely often.

  liveness

- Two philosophers next to each other never eat at the same time.

which property type?

- Each philosopher thinks infinitely often.

  liveness

- Two philosophers next to each other never eat at the same time.

  invariant

which property type?

- Each philosopher thinks infinitely often.

liveness

- Two philosophers next to each other never eat at the same time.

invariant

- Whenever a philosopher eats then he has been thinking at some time before.

which property type? LF2.6-2

- Each philosopher thinks infinitely often.

  liveness

- Two philosophers next to each other never eat at the same time.

  invariant

- Whenever a philosopher eats then he has been thinking at some time before.

  safety

which property type?

- Each philosopher thinks infinitely often.

liveness

- Two philosophers next to each other never eat at the same time.

invariant

- Whenever a philosopher eats then he has been thinking at some time before.

safety

- Whenever a philosopher eats then he will think some time afterwards.

which property type? LF2.6-2

- Each philosopher thinks infinitely often.

  liveness

- Two philosophers next to each other never eat at the same time.

  invariant

- Whenever a philosopher eats then he has been thinking at some time before.

  safety

- Whenever a philosopher eats then he will think some time afterwards.

  liveness

- Each philosopher thinks infinitely often.

  liveness

- Two philosophers next to each other never eat at the same time.

  invariant

- Whenever a philosopher eats then he has been thinking at some time before.

  safety

- Whenever a philosopher eats then he will think some time afterwards.

  liveness

- Between two eating phases of philosopher i lies at least one eating phase of philosopher $i+1$.

which property type?

- Each philosopher thinks infinitely often.

  liveness

- Two philosophers next to each other never eat at the same time.

  invariant

- Whenever a philosopher eats then he has been thinking at some time before.

  safety

- Whenever a philosopher eats then he will think some time afterwards.

  liveness

- Between two eating phases of philosopher $\boldsymbol{i}$ lies at least one eating phase of philosopher $\boldsymbol{i+1}$.

  safety

Liveness

many different formal definitions of liveness
have been suggested in the literature

many different formal definitions of liveness have been suggested in the literature

here: one just example for a formal definition of liveness

Definition of liveness properties

LF2.6-DEF-LIVENESS

Definition of liveness properties

Let E be an LT property over AP, i.e., $E \subseteq \left(2^{AP}\right)^{\omega}$.

E is called a liveness property if each finite word over AP can be extended to an infinite word in E

Definition of liveness properties

Let E be an LT property over AP, i.e., $E \subseteq \left(2^{AP}\right)^{\omega}$.

> E is called a liveness property if each finite word over AP can be extended to an infinite word in E, i.e., if
>
> $$pref(E) = \left(2^{AP}\right)^{+}$$

recall: $pref(E) =$ set of all finite, nonempty prefixes of words in E

Definition of liveness properties

Let E be an LT property over AP, i.e., $E \subseteq (2^{AP})^\omega$.

E is called a liveness property if each finite word over AP can be extended to an infinite word in E, i.e., if

$$pref(E) = (2^{AP})^+$$

Examples:

- each process will eventually enter its critical section
- each process will enter its critical section infinitely often
- whenever a process has requested its critical section then it will eventually enter its critical section

Examples for liveness properties

An LT property E over AP is called a liveness property if $pref(E) = \left(2^{AP}\right)^+$

Examples for $AP = \{crit_i : i = 1, \ldots, n\}$:

Examples for liveness properties

An LT property E over AP is called a liveness property if $pref(E) = (2^{AP})^+$

Examples for $AP = \{crit_i : i = 1, \ldots, n\}$:

- each process will eventually enter its critical section

Examples for liveness properties

An LT property E over AP is called a liveness property if $pref(E) = (2^{AP})^+$

Examples for $AP = \{crit_i : i = 1, \ldots, n\}$:

- each process will eventually enter its critical section

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

$\forall i \in \{1, \ldots, n\}\ \exists k \geq 0.\ crit_i \in A_k$

Examples for liveness properties

An LT property E over AP is called a liveness property if $pref(E) = (2^{AP})^+$

Examples for $AP = \{crit_i : i = 1, \ldots, n\}$:

- each process will eventually enter its critical section
- each process will enter its critical section infinitely often

Examples for liveness properties

An LT property E over AP is called a liveness property if $pref(E) = (2^{AP})^+$

Examples for $AP = \{crit_i : i = 1, \ldots, n\}$:

- each process will eventually enter its critical section
- each process will enter its critical section infinitely often

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

$$\forall i \in \{1, \ldots, n\} \ \overset{\infty}{\exists} k \geq 0. \ crit_i \in A_k$$

Examples for liveness properties

An LT property E over AP is called a liveness property if $pref(E) = (2^{AP})^+$

Examples for $AP = \{wait_i, crit_i : i = 1, \ldots, n\}$:

- each process will eventually enter its critical section
- each process will enter its crit. section inf. often
- whenever a process is waiting then it will eventually enter its critical section

Examples for liveness properties

An LT property E over AP is called a liveness property if $pref(E) = (2^{AP})^+$

Examples for $AP = \{wait_i, crit_i : i = 1, \ldots, n\}$:

- each process will eventually enter its critical section
- each process will enter its crit. section inf. often
- whenever a process is waiting then it will eventually enter its critical section

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

$$\forall i \in \{1, \ldots, n\} \; \forall j \geq 0. \; wait_i \in A_j \longrightarrow \exists k > j. \; crit_i \in A_k$$

Recall: safety properties, prefix closure

Let E be an LT-property, i.e., $E \subseteq \left(2^{AP}\right)^{\omega}$

Recall: safety properties, prefix closure

Let E be an LT-property, i.e., $E \subseteq \left(2^{AP}\right)^{\omega}$

E is a safety property

iff $\forall \sigma \in \left(2^{AP}\right)^{\omega} \setminus E \;\; \exists A_0\, A_1 \ldots A_n \in \mathit{pref}(\sigma)$ s.t.

$$\left\{\sigma' \in E : A_0\, A_1 \ldots A_n \in \mathit{pref}(\sigma')\right\} = \varnothing$$

Recall: safety properties, prefix closure

Let E be an LT-property, i.e., $E \subseteq (2^{AP})^{\omega}$

E is a safety property

iff $\forall \sigma \in (2^{AP})^{\omega} \setminus E \;\; \exists A_0 A_1 \ldots A_n \in \mathit{pref}(\sigma)$ s.t.

$$\{\sigma' \in E : A_0 A_1 \ldots A_n \in \mathit{pref}(\sigma')\} = \varnothing$$

remind:

$\mathit{pref}(\sigma)$ = set of all finite, nonempty prefixes of σ

$$\mathit{pref}(E) = \bigcup_{\sigma \in E} \mathit{pref}(\sigma)$$

Recall: safety properties, prefix closure

Let E be an LT-property, i.e., $E \subseteq \left(2^{AP}\right)^{\omega}$

E is a safety property

iff $\forall \sigma \in \left(2^{AP}\right)^{\omega} \setminus E \;\; \exists A_0 A_1 \ldots A_n \in pref(\sigma)$ s.t.

$$\left\{\sigma' \in E : A_0 A_1 \ldots A_n \in pref(\sigma')\right\} = \varnothing$$

iff $cl(E) = E$

remind: $cl(E) = \left\{\sigma \in \left(2^{AP}\right)^{\omega} : pref(\sigma) \subseteq pref(E)\right\}$

$pref(\sigma)$ = set of all finite, nonempty prefixes of σ

$$pref(E) = \bigcup_{\sigma \in E} pref(\sigma)$$

Decomposition theorem

Decomposition theorem

For each LT-property E, there exists a safety property $SAFE$ and a liveness property $LIVE$ s.t.

$$E = SAFE \cap LIVE$$

Decomposition theorem

For each LT-property E, there exists a safety property $SAFE$ and a liveness property $LIVE$ s.t.

$$E = SAFE \cap LIVE$$

Proof:

Decomposition theorem

For each LT-property E, there exists a safety property $SAFE$ and a liveness property $LIVE$ s.t.

$$E = SAFE \cap LIVE$$

Proof: Let $SAFE \stackrel{\text{def}}{=} cl(E)$

Decomposition theorem

For each LT-property E, there exists a safety property $SAFE$ and a liveness property $LIVE$ s.t.

$$E = SAFE \cap LIVE$$

Proof: Let $SAFE \stackrel{\text{def}}{=} cl(E)$

---

remind: $cl(E) = \{\sigma \in (2^{AP})^\omega : pref(\sigma) \subseteq pref(E)\}$

$pref(\sigma)$ = set of all finite, nonempty prefixes of σ

$$pref(E) = \bigcup_{\sigma \in E} pref(\sigma)$$

Decomposition theorem

For each LT-property *E*, there exists a safety property *SAFE* and a liveness property *LIVE* s.t.

$$E = SAFE \cap LIVE$$

Proof: Let

$$SAFE \stackrel{\text{def}}{=} cl(E)$$

$$LIVE \stackrel{\text{def}}{=} E \cup \left(\left(2^{AP}\right)^{\omega} \setminus cl(E) \right)$$

---

remind: $cl(E) = \left\{\sigma \in \left(2^{AP}\right)^{\omega} : pref(\sigma) \subseteq pref(E)\right\}$

$pref(\sigma)$ = set of all finite, nonempty prefixes of σ

$$pref(E) = \bigcup_{\sigma \in E} pref(\sigma)$$

Decomposition theorem

For each LT-property E, there exists a safety property $SAFE$ and a liveness property $LIVE$ s.t.

$$E \;=\; SAFE \cap LIVE$$

Proof: Let $SAFE \stackrel{\text{def}}{=} cl(E)$

$$LIVE \stackrel{\text{def}}{=} E \cup \left(\left(2^{AP}\right)^{\omega} \setminus cl(E) \right)$$

Show that:

- $E = SAFE \cap LIVE$
- $SAFE$ is a safety property
- $LIVE$ is a liveness property

Decomposition theorem

For each LT-property E, there exists a safety property $SAFE$ and a liveness property $LIVE$ s.t.

$$E = SAFE \cap LIVE$$

Proof: Let

$$SAFE \stackrel{\text{def}}{=} cl(E)$$

$$LIVE \stackrel{\text{def}}{=} E \cup \left(\left(2^{AP}\right)^{\omega} \setminus cl(E) \right)$$

Show that:

- $E = SAFE \cap LIVE$ $\surd$
- $SAFE$ is a safety property
- $LIVE$ is a liveness property

For each LT-property E, there exists a safety property $SAFE$ and a liveness property $LIVE$ s.t.

$$E = SAFE \cap LIVE$$

Proof: Let

$$SAFE \stackrel{\text{def}}{=} cl(E)$$

$$LIVE \stackrel{\text{def}}{=} E \cup \left(\left(2^{AP}\right)^{\omega} \setminus cl(E) \right)$$

Show that:

- $E = SAFE \cap LIVE$ ✓
- $SAFE$ is a safety property as $cl(SAFE) = SAFE$
- $LIVE$ is a liveness property

For each LT-property E, there exists a safety property $SAFE$ and a liveness property $LIVE$ s.t.

$$E = SAFE \cap LIVE$$

Proof: Let $SAFE \stackrel{\text{def}}{=} cl(E)$

$$LIVE \stackrel{\text{def}}{=} E \cup \left((2^{AP})^{\omega} \setminus cl(E) \right)$$

Show that:

- $E = SAFE \cap LIVE$ $\surd$
- $SAFE$ is a safety property as $cl(SAFE) = SAFE$
- $LIVE$ is a liveness property, i.e., $pref(LIVE) = (2^{AP})^{+}$

Being safe and live

Which LT properties are both
a safety and a liveness property?

Being safe and live

Which LT properties are both a safety and a liveness property?

answer: The set $\left(2^{AP}\right)^\omega$ is the only LT property which is a safety property and a liveness property

Which LT properties are both a safety and a liveness property?

answer: The set $(2^{AP})^{\omega}$ is the only LT property which is a safety property and a liveness property

- $(2^{AP})^{\omega}$ is a safety and a liveness property: $\surd$

Which LT properties are both a safety and a liveness property?

answer: The set $(2^{AP})^{\omega}$ is the only LT property which is a safety property and a liveness property

- $(2^{AP})^{\omega}$ is a safety and a liveness property: $\checkmark$
- If E is a liveness property then

$$pref(E) = (2^{AP})^{+}$$

Which LT properties are both
a safety and a liveness property?

answer: The set $(2^{AP})^\omega$ is the only LT property which is a safety property and a liveness property

- $(2^{AP})^\omega$ is a safety and a liveness property: $\surd$
- If E is a liveness property then

$$pref(E) = (2^{AP})^+$$

$$\Longrightarrow \quad cl(E) = (2^{AP})^\omega$$

Being safe and live

Which LT properties are both
a safety and a liveness property?

answer: The set $(2^{AP})^{\omega}$ is the only LT property which is a safety property and a liveness property

- $(2^{AP})^{\omega}$ is a safety and a liveness property: $\surd$
- If E is a liveness property then

$$\begin{aligned} & pref(E) = (2^{AP})^{+} \\ \Longrightarrow \quad & cl(E) = (2^{AP})^{\omega} \end{aligned}$$

If E is a safety property too, then $cl(E) = E$.

Being safe and live

> Which LT properties are both a safety and a liveness property?

answer: The set $(2^{AP})^{\omega}$ is the only LT property which is a safety property and a liveness property

- $(2^{AP})^{\omega}$ is a safety and a liveness property: ✓
- If E is a liveness property then

$$pref(E) = (2^{AP})^{+}$$

$$\Longrightarrow \quad cl(E) = (2^{AP})^{\omega}$$

If E is a safety property too, then $cl(E) = E$.
Hence $E = cl(E) = (2^{AP})^{\omega}$.