Model Checking I
alias
Reactive Systems Verification

Luca Tesei

MSc in Computer Science, University of Camerino

Topics

- Definition of Linear Time properties. Examples.
- Satisfaction Relation of linear time properties. Examples.
- Trace Inclusion. Trace Equivalence.

Material

Reading:

Chapter 3 of the book, pages 99–106.

More:

The slides in the following pages are taken from the material of the course "Introduction to Model Checking" held by Prof. Dr. Ir. Joost-Pieter Katoen at Aachen University.

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

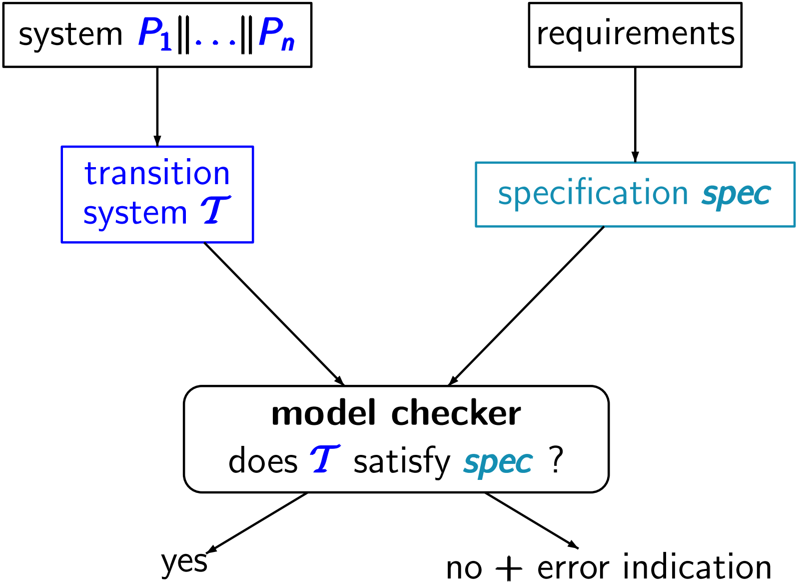
Regular Properties

Linear Temporal Logic

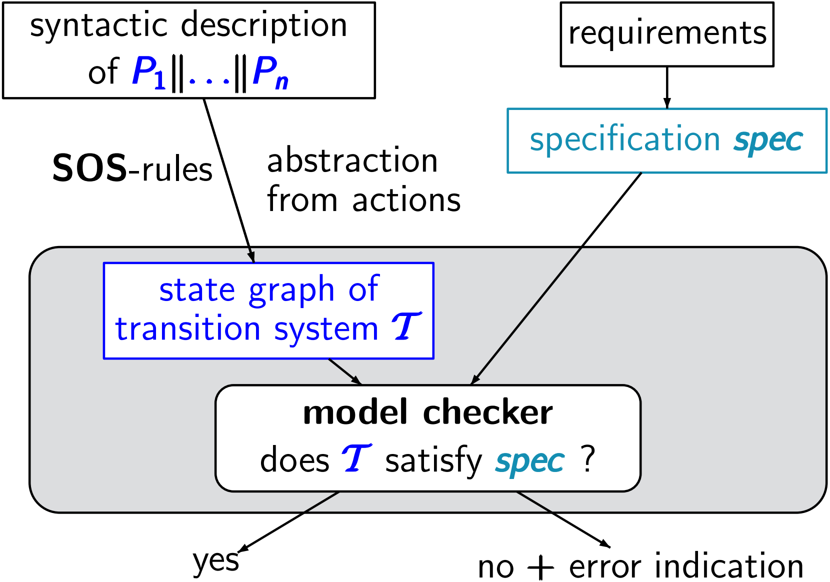
Computation-Tree Logic

Equivalences and Abstraction

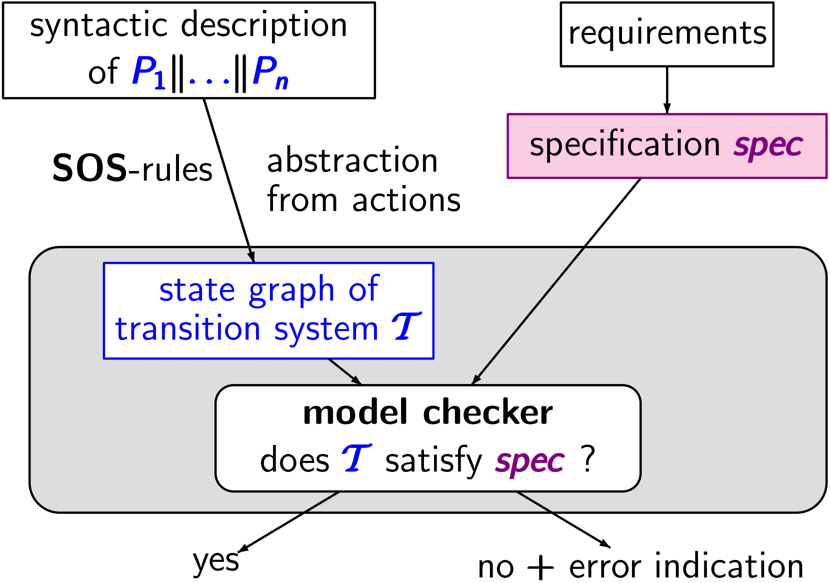
Model checking



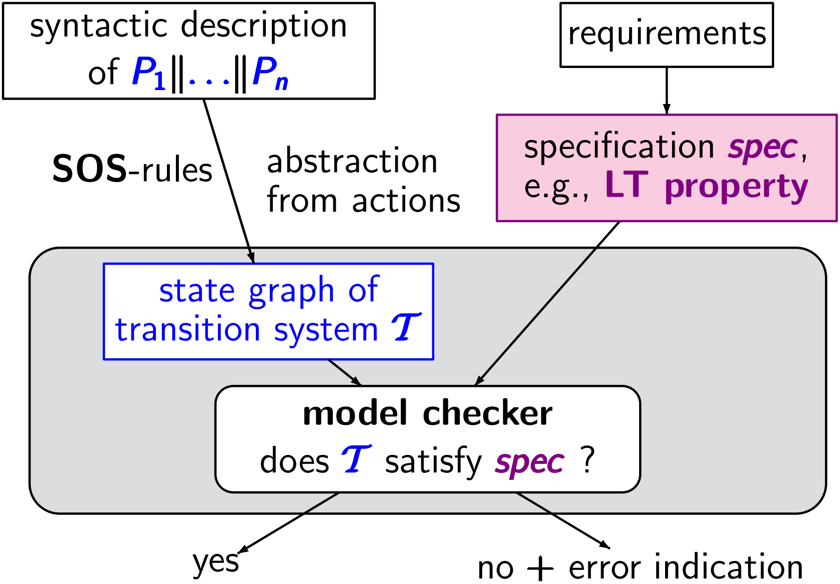
Model checking



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 $\Sigma = 2^{AP}$, 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., $E \subseteq \left(2^{AP}\right)^{\omega}$.

for TS over AP without terminal states

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

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

safety:

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

LT properties for mutual exclusion protocols

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

safety:

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

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

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

LT properties for mutual exclusion protocols

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

safety:

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

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

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

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

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

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

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

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

Satisfaction relation for LT properties

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

Satisfaction relation $\models$ for TS and states:

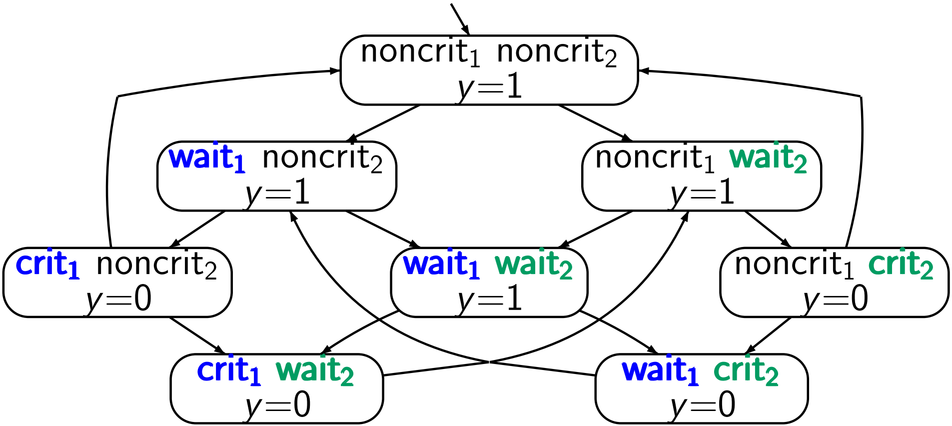
If $\mathcal{T}$ is a TS (without terminal states) over AP and E an LT property over AP then

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

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

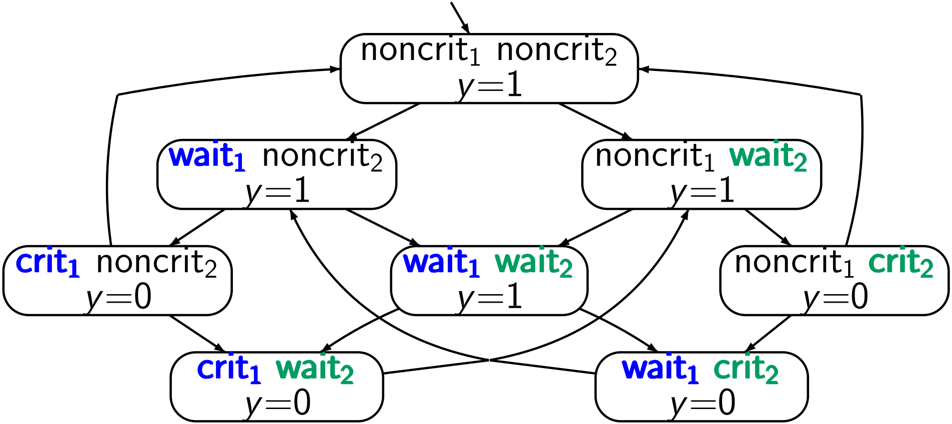
$$s \models E \quad \text{iff} \quad Traces(s) \subseteq E$$

Mutual exclusion with semaphore



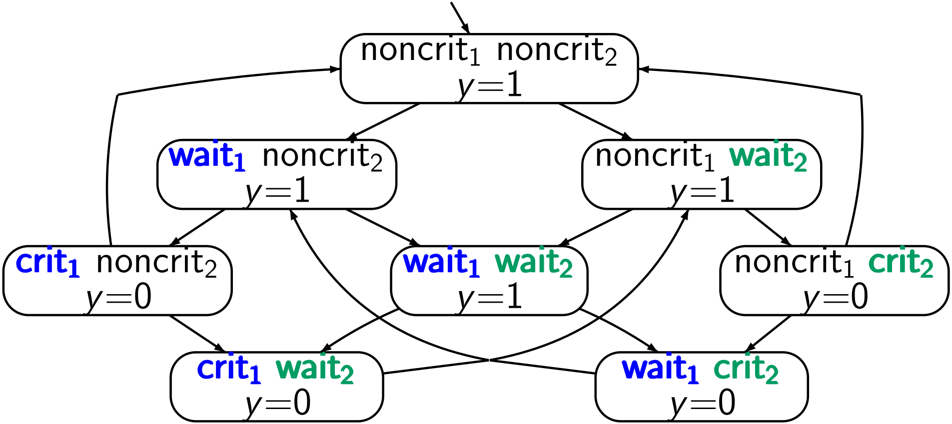
$\mathcal{T}_{Sem} \models MUTEX$

Mutual exclusion with semaphore



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

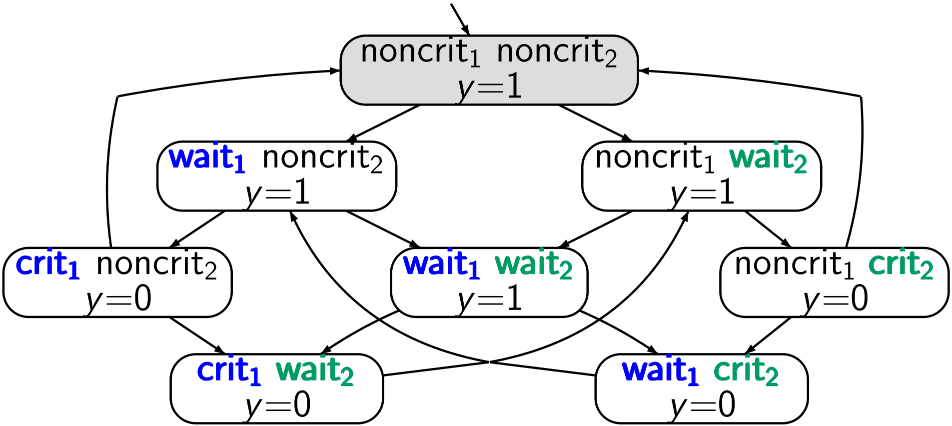
Mutual exclusion with semaphore



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

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

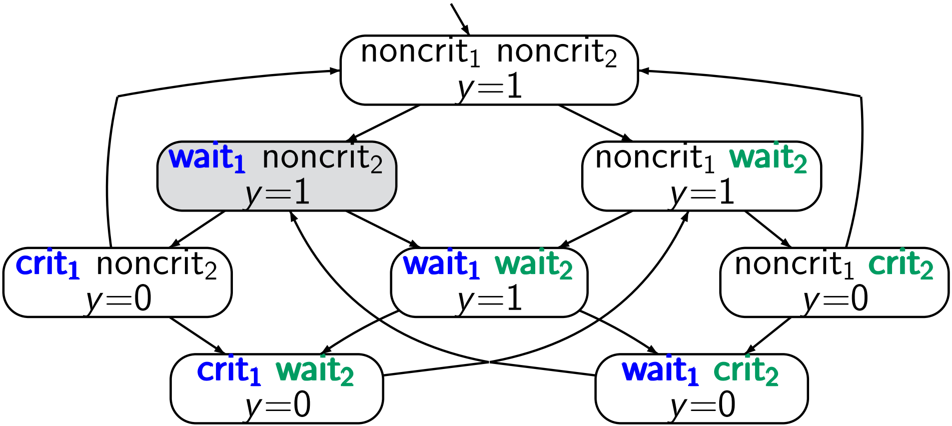
Mutual exclusion with semaphore



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

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

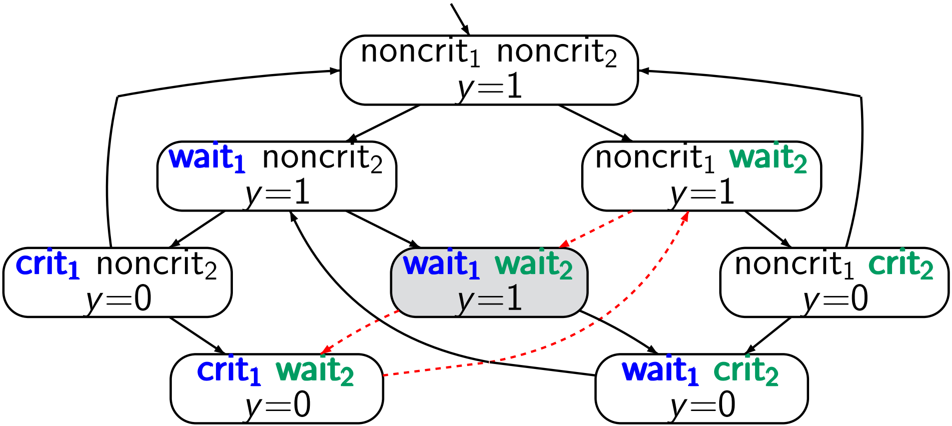
Mutual exclusion with semaphore



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

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

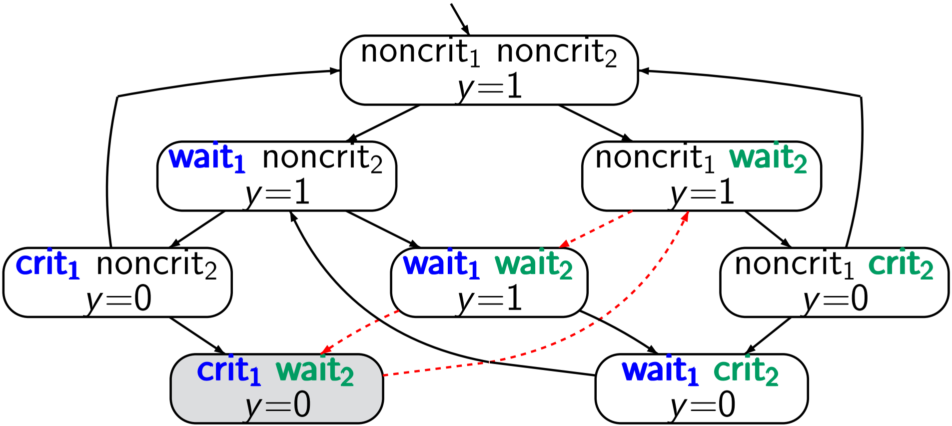
Mutual exclusion with semaphore



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

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

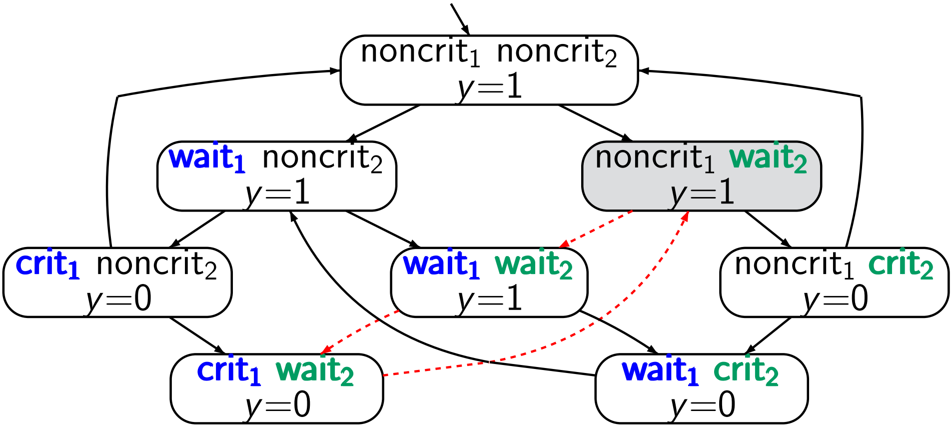
Mutual exclusion with semaphore



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

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

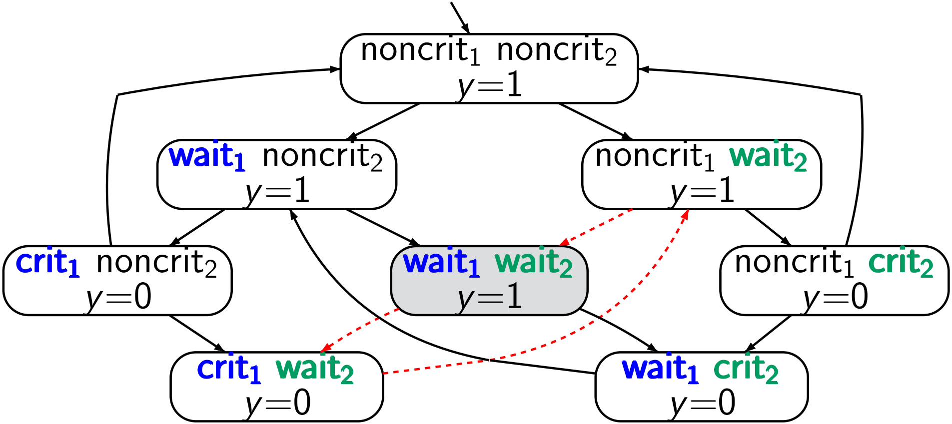
Mutual exclusion with semaphore



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

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

Mutual exclusion with semaphore



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

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

Peterson's mutual exclusion algorithm

Peterson's mutual exclusion algorithm

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

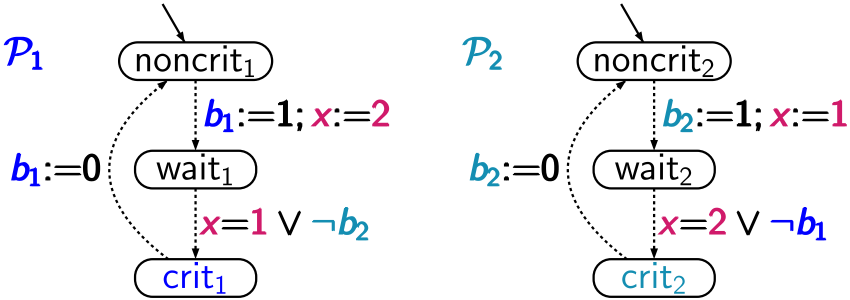
using three additional shared variables

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

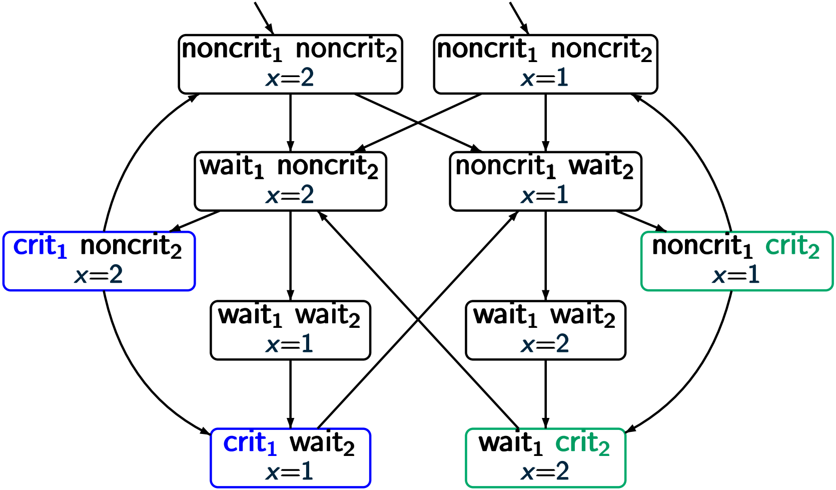
Peterson's mutual exclusion algorithm

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

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

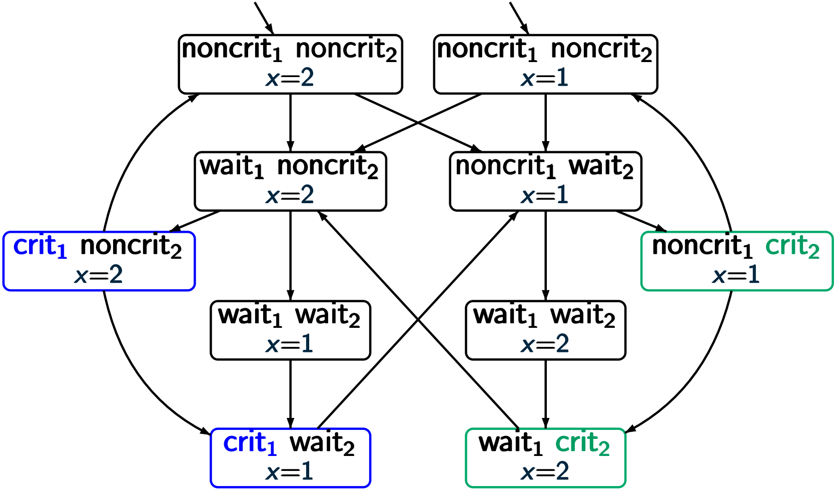


Peterson's mutual exclusion algorithm



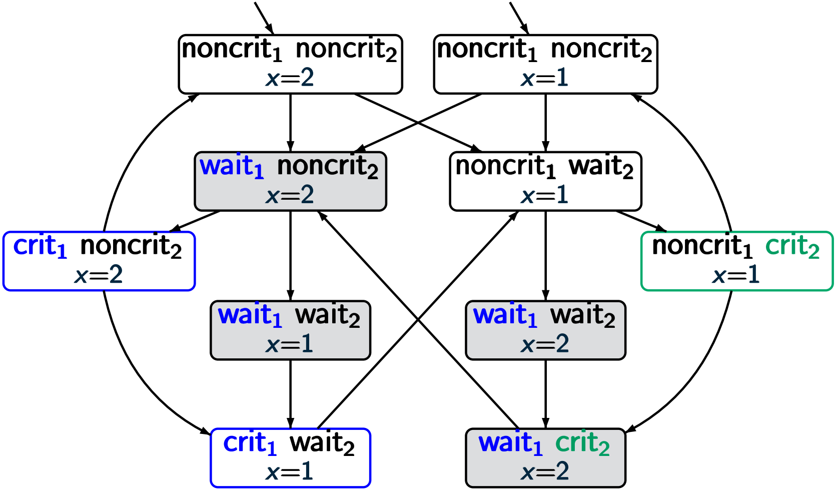
$\mathcal{T}_{Pet} \models$ *MUTEX*

Peterson's mutual exclusion algorithm



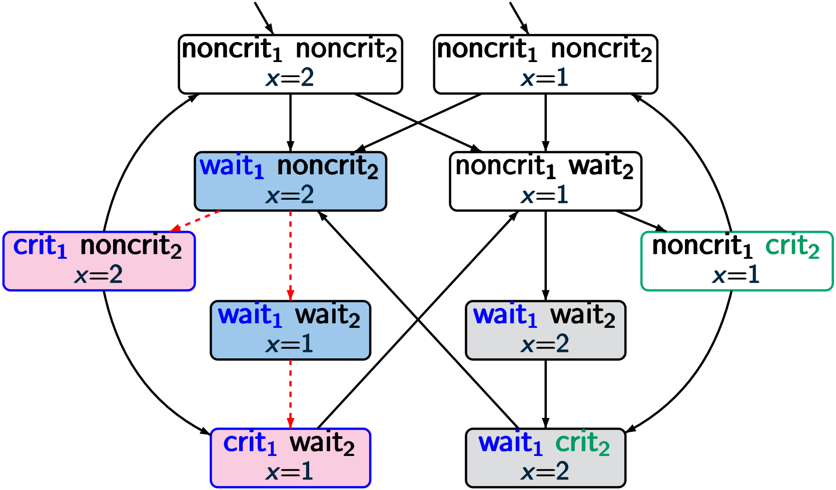
$\mathcal{T}_{Pet} \models \mathit{MUTEX}$ and $\mathcal{T}_{Pet} \models \mathit{LIVE}$

Peterson's mutual exclusion algorithm



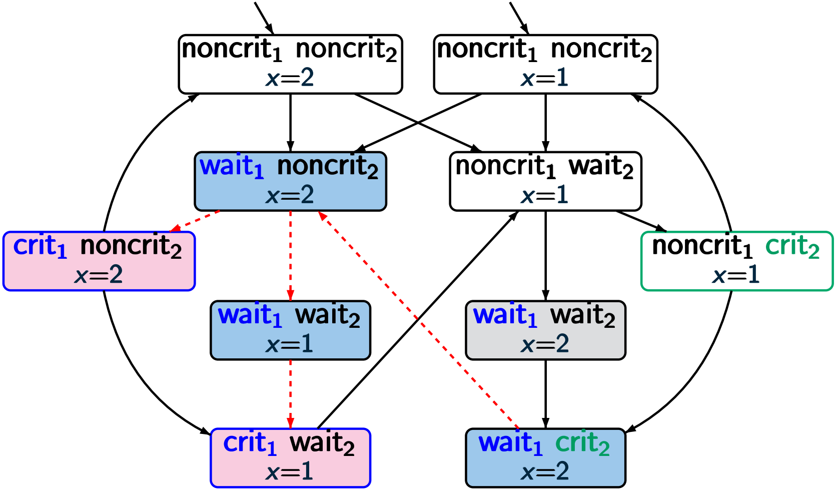
$\mathcal{T}_{Pet} \models \textit{MUTEX}$ and $\mathcal{T}_{Pet} \models \textit{LIVE}$

Peterson's mutual exclusion algorithm



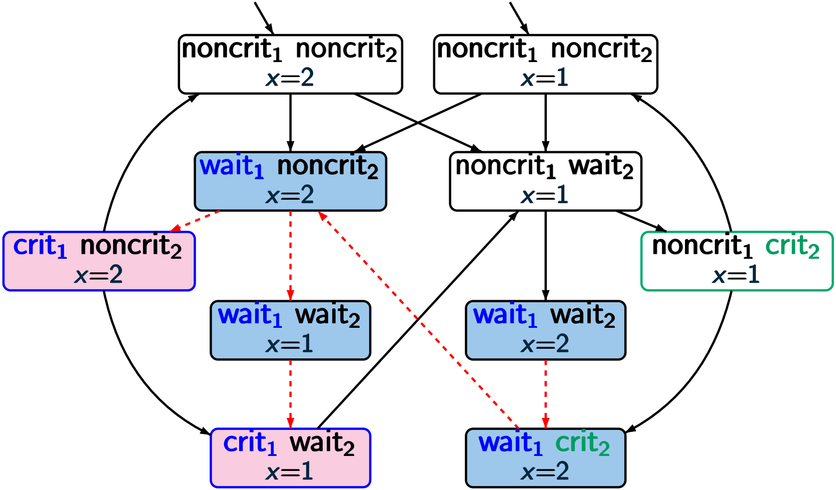
$\mathcal{T}_{Pet} \models \mathit{MUTEX}$ and $\mathcal{T}_{Pet} \models \mathit{LIVE}$

Peterson's mutual exclusion algorithm



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

Peterson's mutual exclusion algorithm



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

LT properties and trace inclusion

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

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

LT properties and trace inclusion

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

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

Consequence of these definitions:

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

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

LT properties and trace inclusion

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

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

Consequence of these definitions:

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

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

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

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 $\mathcal{T}_1 \models E$

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

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 $\mathcal{T}_1 \models E$

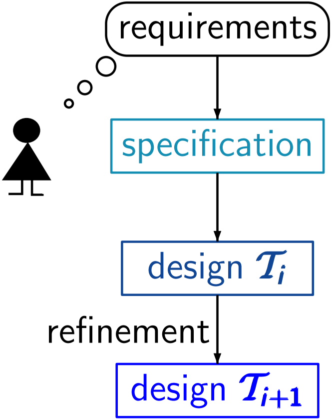
$(2) \Longrightarrow (1)$: consider $E = Traces(\mathcal{T}_2)$

Relevance of trace inclusion

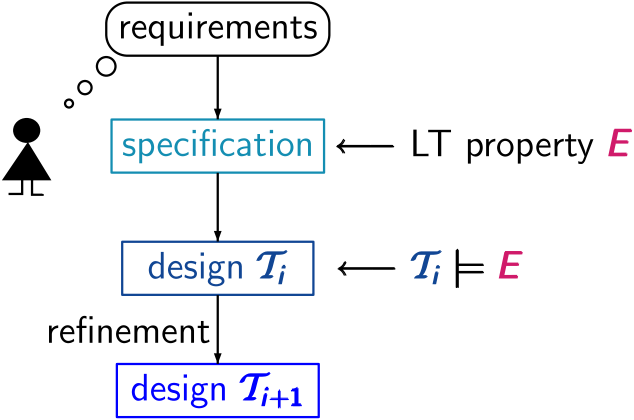
Trace inclusion appears naturally

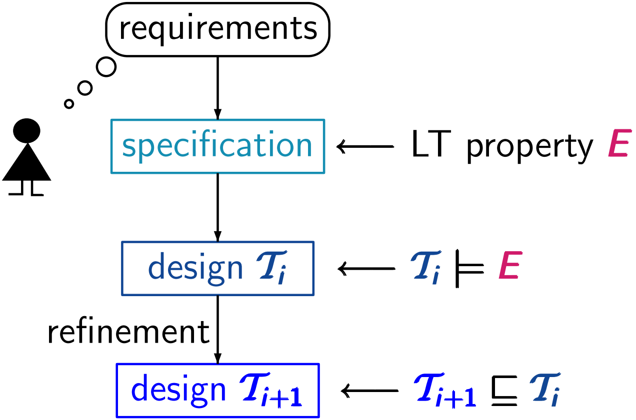
- as an implementation/refinement relation
- when resolving nondeterminism
- in the context of abstractions

Software design cycle



Software design cycle





implementation/refinement relation $\sqsubseteq$:

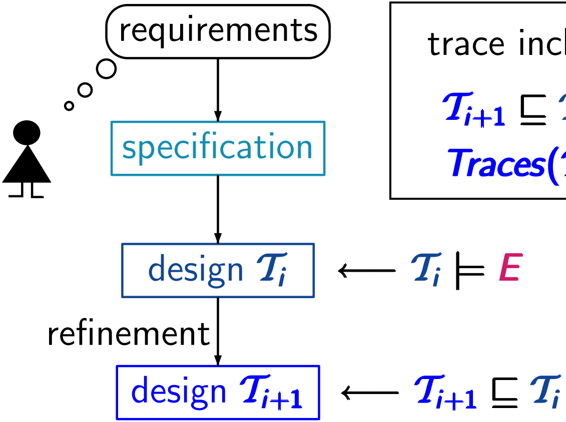
$\mathcal{T}_{i+1} \sqsubseteq \mathcal{T}_i$ iff "$\mathcal{T}_{i+1}$ correctly implements $\mathcal{T}_i$"

Trace inclusion as an implementation relation

trace inclusion

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

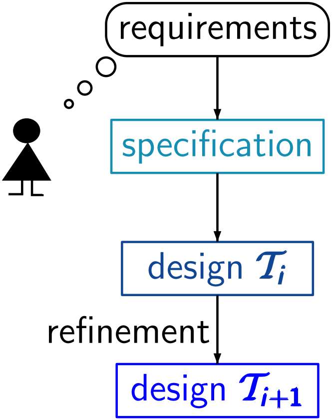
$\mathit{Traces}(\mathcal{T}_{i+1}) \subseteq \mathit{Traces}(\mathcal{T}_i)$



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

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

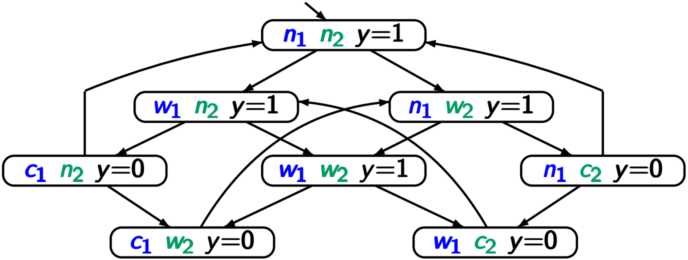
design $\mathcal{T}_i$ $\longleftarrow$ $\mathcal{T}_i \models E$

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

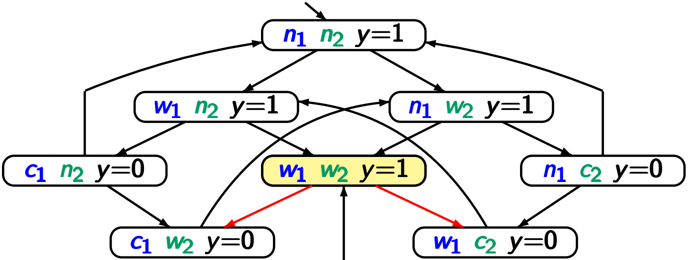
implementation/refinement relation $\sqsubseteq$:

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

Mutual exclusion with semaphore



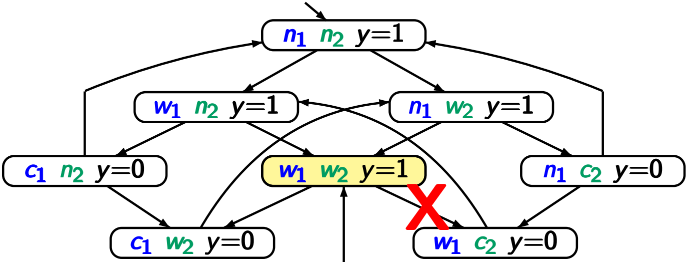
Mutual exclusion with semaphore



competition in state

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

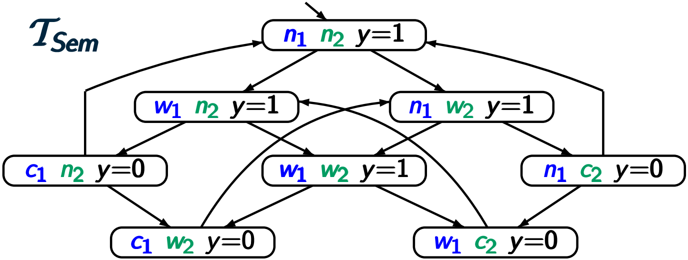
Mutual exclusion with semaphore

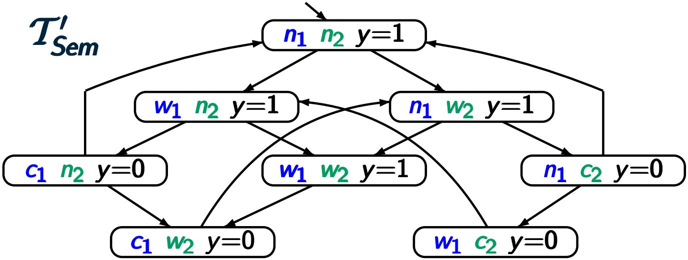


competition in state
$\langle wait_1 \; wait_2 \; y{=}1 \rangle$

resolve the nondeterminism by giving priority to process P_1

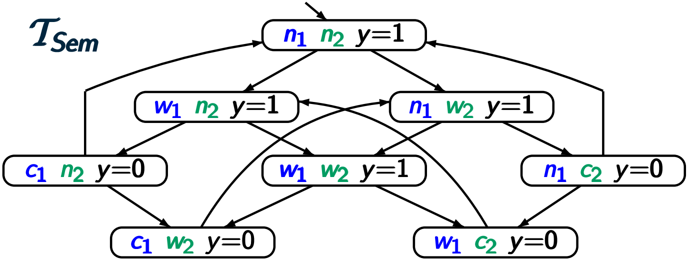
Mutual exclusion with semaphore



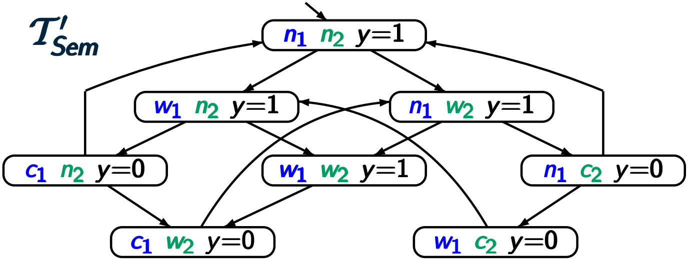


Mutual exclusion with semaphore

$\mathcal{T}_{Sem}$



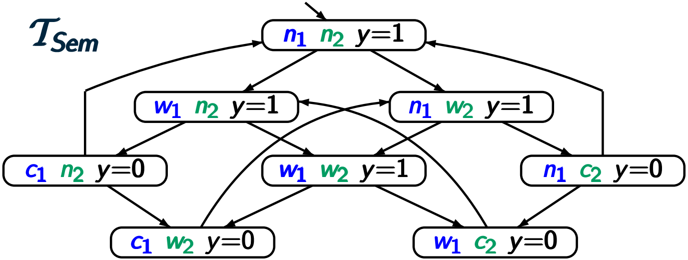
$\mathcal{T}'_{Sem}$



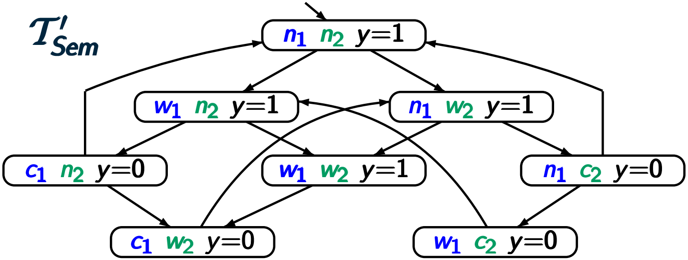
$$Paths(\mathcal{T}'_{Sem}) \subseteq Paths(\mathcal{T}_{Sem})$$

Mutual exclusion with semaphore

$\mathcal{T}_{Sem}$

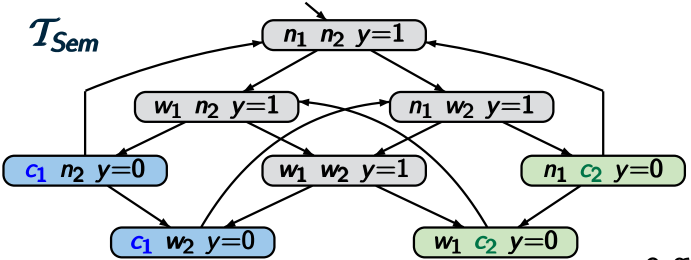


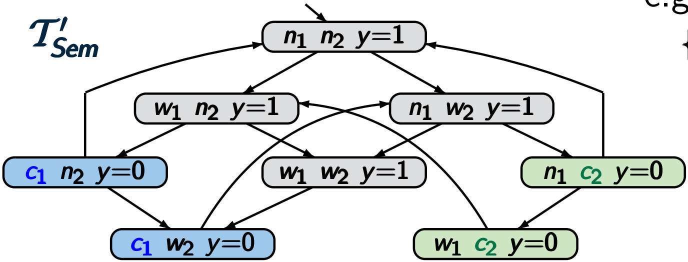
$\mathcal{T}'_{Sem}$



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

Mutual exclusion with semaphore





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

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

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
  ⋮
```

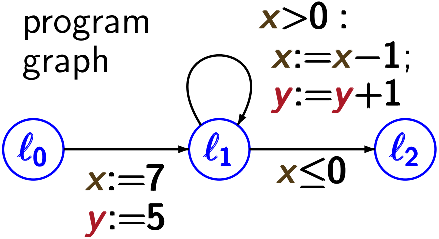
Trace inclusion and data abstraction


```
        ⋮
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      ⋮
```

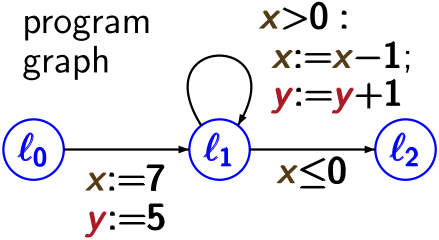


does $\ell_2 \wedge odd(y)$
never hold ?

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

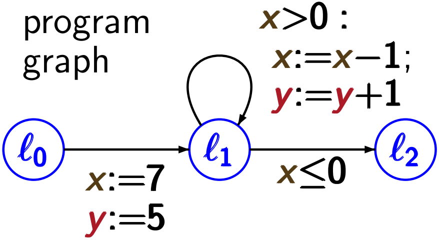


let $\mathcal{T}$ be the associated TS

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      ⋮
```



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

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

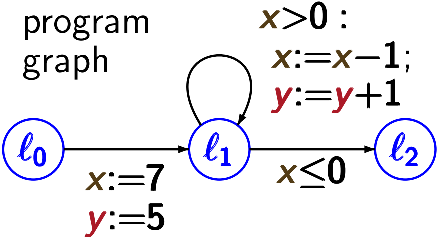
data abstraction w.r.t. the predicates

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

Trace inclusion and data abstraction

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

program graph



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

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

data abstraction w.r.t. the predicates

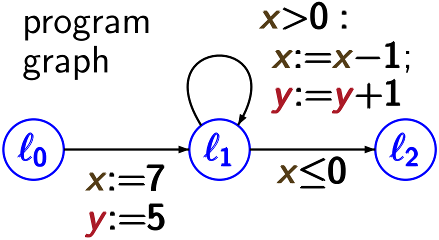
$x>0$, $x=0$, $x \equiv_2 y$ $\longleftarrow$ i.e., $x-y$ is even

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

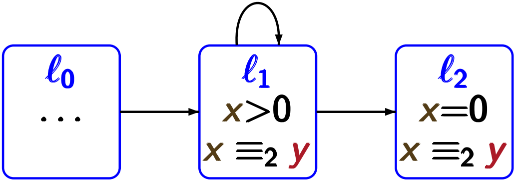
does $\ell_2 \wedge odd(y)$
never hold ?

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

program graph



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

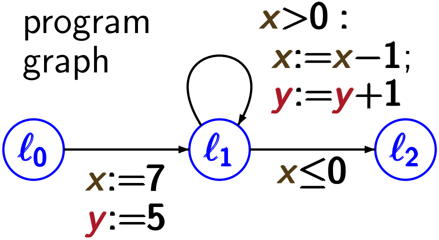


abstract transition system $\mathcal{T}'$

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

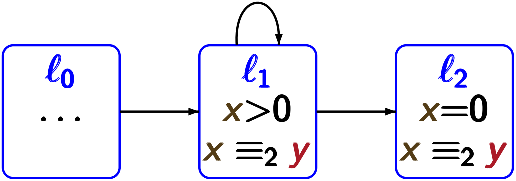


does $\ell_2 \wedge odd(y)$ never hold ?

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

data abstraction w.r.t. the predicates

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



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

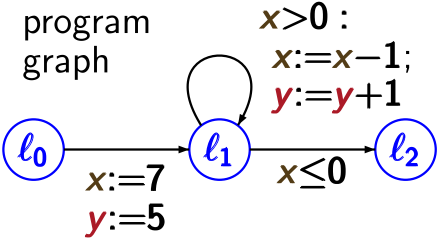
Trace inclusion and data abstraction

$\vdots$
ℓ_0 x:=7; y:=5;
ℓ_1 WHILE $x>0$ DO
  x:=$x-1$;
  y:=$y+1$
OD
ℓ_2 $\vdots$

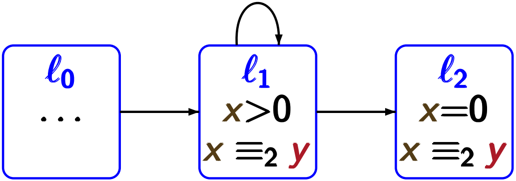
does $\ell_2 \wedge odd(y)$
never hold ?

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 $\ell_2 \wedge odd(y)$"

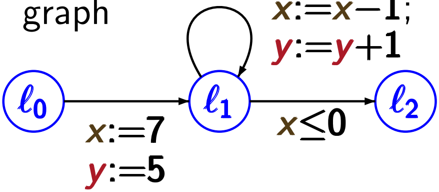
$Traces(\mathcal{T}) \subseteq Traces(\mathcal{T}')$

Trace inclusion and data abstraction

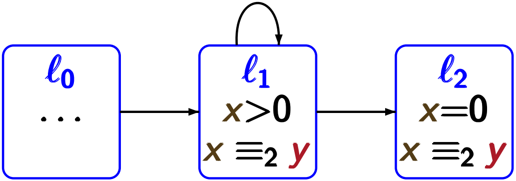

```
      ⋮
ℓ0   x:=7;  y:=5;
ℓ1   WHILE x>0 DO
           x:=x−1;
           y:=y+1
     OD
ℓ2    ⋮
```

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

program graph



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



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

Trace equivalence

Trace equivalence

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

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

Trace equivalence

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

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

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

Trace equivalence

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

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

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

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

LT properties and trace relations

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

The following statements are equivalent:

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

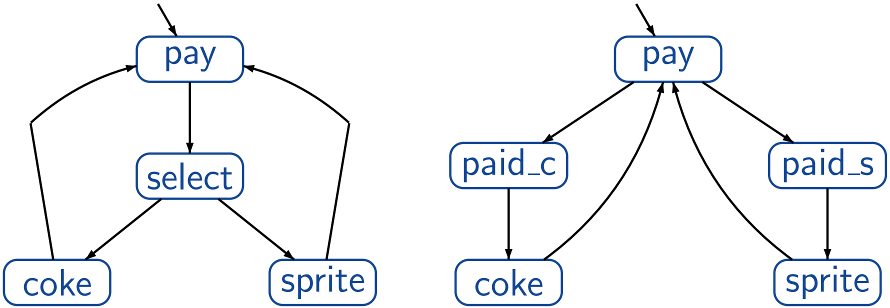
(2) for all LT-properties E: $\mathcal{T}_2 \models E \Longrightarrow \mathcal{T}_1 \models E$

The following statements are equivalent:

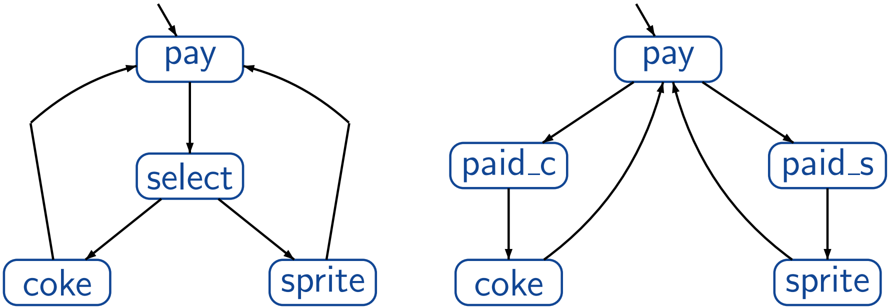
(1) $Traces(\mathcal{T}_1) = Traces(\mathcal{T}_2)$

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

Trace equivalent beverage machines

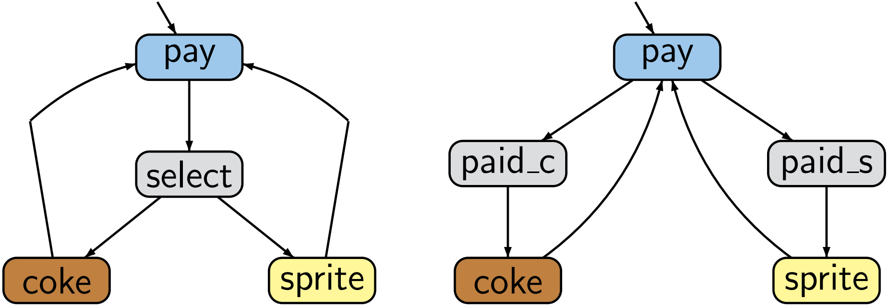


Trace equivalent beverage machines



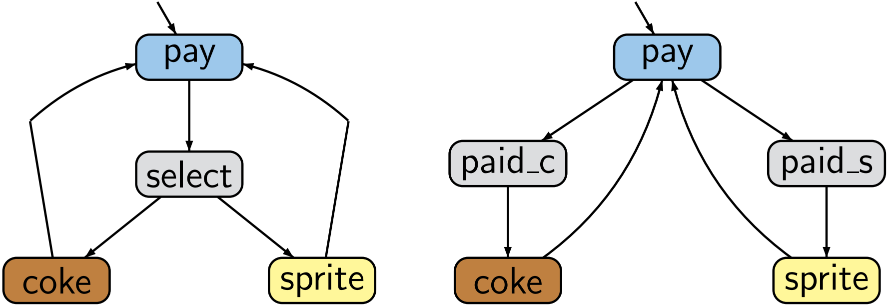
set of atomic propositions $AP = \{pay, coke, sprite\}$

Trace equivalent beverage machines



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

Trace equivalent beverage machines



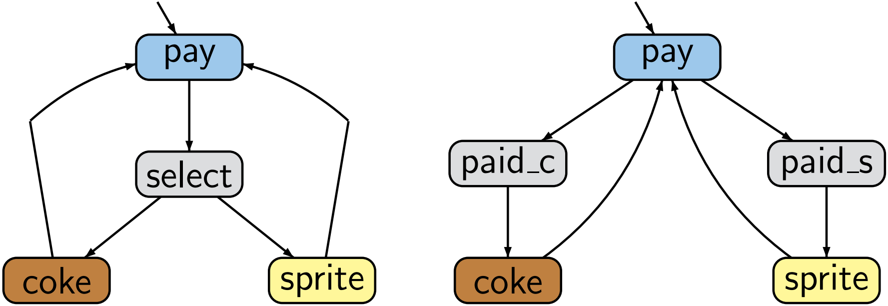
set of atomic propositions $AP = \{pay, coke, sprite\}$

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