Overview: Model Checking

1. Introduction
2. Modelling parallel systems
3. Linear Time Properties
4. Regular Properties
5. Linear Temporal Logic
6. Computation Tree Logic
7. Equivalences and Abstraction
8. **Partial Order Reduction**
9. Timed Automata
10. Probabilistic Systems

A few historical notes

1977 temporal logics (**LTL**) as specification formalism
for parallel systems [Pnueli]

1981 model checking
for **CTL** [Clarke/Emerson], [Queile/Sifakis]
for **LTL** [Lichtenstein/Pnueli], [Vardi/Wolper]
for **CTL*** [Emerson/Lei]
...

A few historical notes

1977 temporal logics (**LTL**) as specification formalism
for parallel systems [Pnueli]

1981 model checking

for **CTL** [Clarke/Emerson], [Queile/Sifakis]
for **LTL** [Lichtenstein/Pnueli], [Vardi/Wolper]
for **CTL*** [Emerson/Lei]
...

.. state explosion problem ...

A few historical notes

1977 temporal logics (**LTL**) as specification formalism
for parallel systems [Pnueli]

1981 model checking

... to attack the state explosion problem ...

1987 symbolic model checking with BDDs [McMillan]
1990 partial order reduction
[Godefroid], [Valmari], [Peled]
1992 net unfoldings
abstraction-refinement
symmetry reduction

A few historical notes

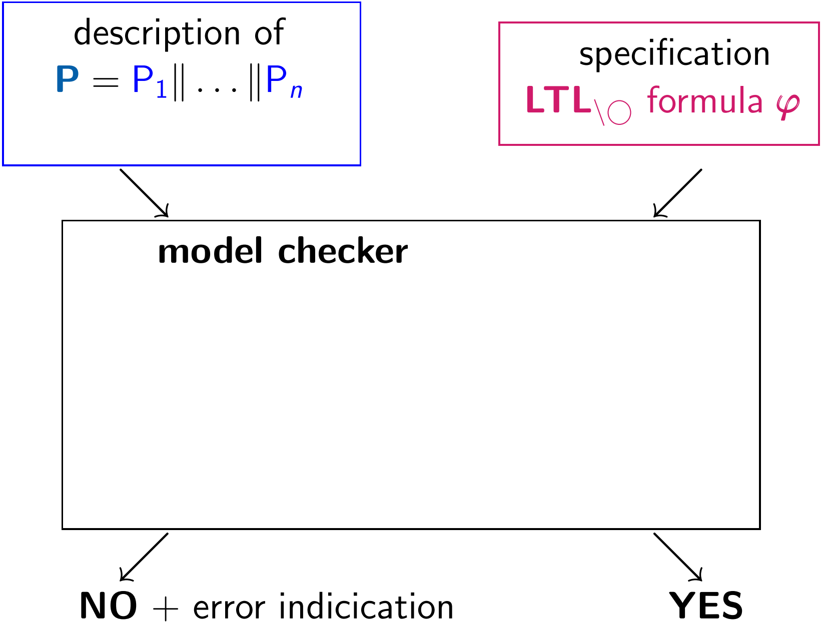
1977 *temporal logics* (LTL) as specification formalism
for parallel systems [Pnueli]

1981 *model checking*

... to attack the *state explosion problem* ...

1987 symbolic model checking with BDDs [McMillan]
1990 **partial order reduction** for **$\mathrm{LTL}_{\setminus\bigcirc}$**
[Godefroid], [Valmari], [Peled]
1992 net unfoldings
abstraction-refinement
symmetry reduction

Partial order reduction



Partial order reduction

description of
$\mathbf{P} = \mathsf{P}_1 \| \ldots \| \mathsf{P}_n$
e.g., by PROMELA

specification
$\mathbf{LTL}_{\setminus \bigcirc}$ formula φ

model checker e.g., SPIN

NO + error indicication

YES

Partial order reduction

description of
$\mathbf{P} = \mathrm{P}_1 \| \ldots \| \mathrm{P}_n$
e.g., by PROMELA

specification
$\mathbf{LTL}_{\setminus \bigcirc}$ formula φ

model checker e.g., SPIN

NBA $\mathcal{A}$ for $\neg\varphi$

NO + error indicication

YES

Partial order reduction

description of
$\mathbf{P} = \mathsf{P}_1 \| \ldots \| \mathsf{P}_n$
e.g., by PROMELA

specification
$\mathbf{LTL}_{\setminus\bigcirc}$ formula φ

model checker e.g., SPIN

NBA $\mathcal{A}$ for $\neg\varphi$

on-the-fly-construction
of a fragment $\mathcal{T}_{\text{red}}$ of $\mathcal{T}_{\mathbf{P}} \times \mathcal{A}$

NO + error indicication

YES

Partial order reduction

description of
$\mathbf{P} = P_1 \| \ldots \| P_n$
e.g., by PROMELA

specification
$\mathbf{LTL}_{\setminus\bigcirc}$ formula φ

model checker e.g., SPIN

NBA $\mathcal{A}$ for $\neg\varphi$

on-the-fly-construction
of a fragment $\mathcal{T}_{\text{red}}$ of $\mathcal{T}_{\mathbf{P}} \times \mathcal{A}$
with integrated persistence checking
$\mathcal{T}_{\text{red}} \models \Diamond\Box\neg\mathbf{F}$?

NO + error indicication

YES

Basic idea of partial order reduction

- for asynchronous systems

Basic idea of partial order reduction

- for asynchronous systems
- analyze representatives of path equivalence classes

Basic idea of partial order reduction

- for asynchronous systems
- analyze representatives of path equivalence classes that represent the same the same behavior up to the interleaving order

Basic idea of partial order reduction

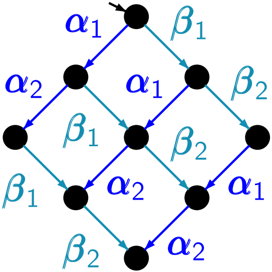
- for asynchronous systems
- analyze representatives of path equivalence classes that represent the same the same behavior up to the interleaving order

$\mathcal{T} = \mathcal{T}_1 \,|||\, \mathcal{T}_2$

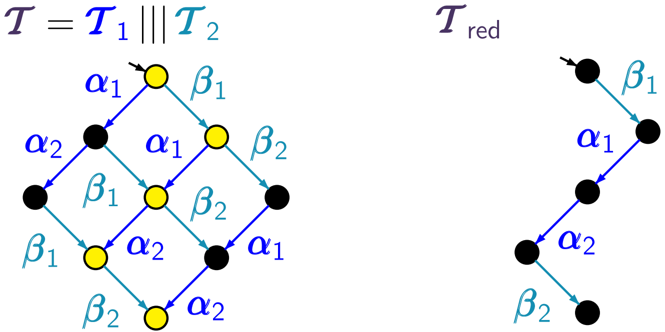
Basic idea of partial order reduction

- for asynchronous systems
- analyze representatives of path equivalence classes that represent the same the same behavior up to the interleaving order

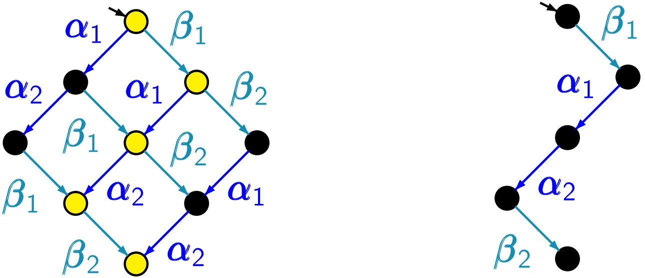
$\mathcal{T} = \mathcal{T}_1 \,|||\, \mathcal{T}_2$



- for asynchronous systems
- analyze representatives of path equivalence classes that represent the same the same behavior up to the interleaving order

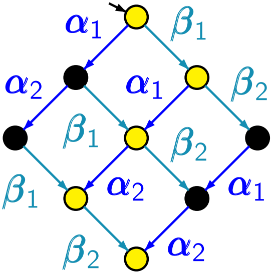


$\mathcal{T} = \mathcal{T}_1 \,|||\, \mathcal{T}_2$ $\qquad\qquad$ $\mathcal{T}_{red}$

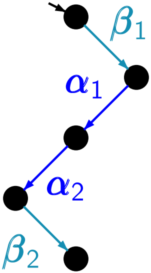


Partial order reduction for $\mathsf{LTL}_{\setminus\bigcirc}$ specifications

$\mathcal{T} = \mathcal{T}_1 \,|||\, \mathcal{T}_2$



$\mathcal{T}_{\mathsf{red}}$

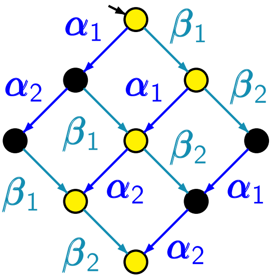


requirement: for all $\mathsf{LTL}_{\setminus\bigcirc}$ formulas φ:

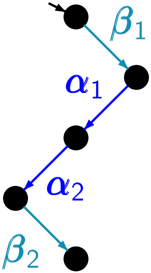
$$\mathcal{T} \models \varphi \quad \text{iff} \quad \mathcal{T}_{\mathsf{red}} \models \varphi$$

Partial order reduction for $\mathsf{LTL}_{\setminus\bigcirc}$ specifications

$\mathcal{T} = \mathcal{T}_1 \,|||\, \mathcal{T}_2$



$\mathcal{T}_{\text{red}}$



requirement: for all $\mathsf{LTL}_{\setminus\bigcirc}$ formulas φ:

$$\mathcal{T} \models \varphi \quad \text{iff} \quad \mathcal{T}_{\text{red}} \models \varphi$$

hence: ensure that the reduction yields $\mathcal{T} \triangleq \mathcal{T}_{\text{red}}$

The ample set method [Peled '93]

given: syntactical representation of processes of TS $\mathcal{T}$

goal: on-the-fly construction of a fragment $\mathcal{T}_{red}$

The ample set method [Peled '93]

given: syntactical representation of processes of TS $\mathcal{T}$

goal: on-the-fly construction of a fragment $\mathcal{T}_{\text{red}}$
by selecting action-sets $\text{ample}(s) \subseteq \text{Act}(s)$

The ample set method [Peled '93]

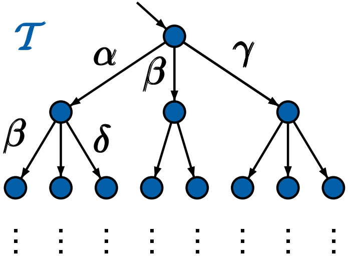
given: syntactical representation of processes of TS $\mathcal{T}$

goal: on-the-fly construction of a fragment $\mathcal{T}_{red}$
by selecting action-sets $ample(s) \subseteq Act(s)$
and expanding only the α-successors of s
where $\alpha \in ample(s)$

The ample set method [Peled '93]

given: syntactical representation of processes of TS $\mathcal{T}$

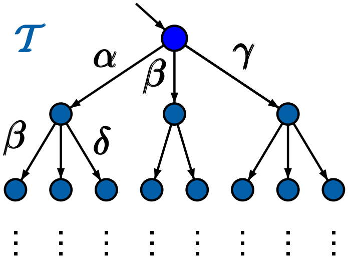
goal: on-the-fly construction of a fragment $\mathcal{T}_{red}$
by selecting action-sets ample(s) ⊆ Act(s)
and expanding only the α-successors of s
where $\alpha \in$ ample(s)



The ample set method [Peled '93]

given: syntactical representation of processes of TS $\mathcal{T}$

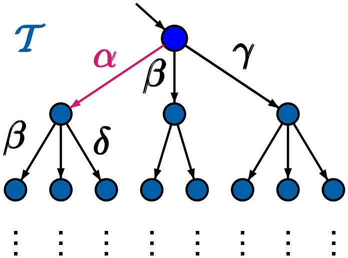
goal: on-the-fly construction of a fragment $\mathcal{T}_{red}$
by selecting action-sets ample(s) $\subseteq$ Act(s)
and expanding only the α-successors of s
where $\alpha \in$ ample(s)



The ample set method [Peled '93]

given: syntactical representation of processes of TS $\mathcal{T}$

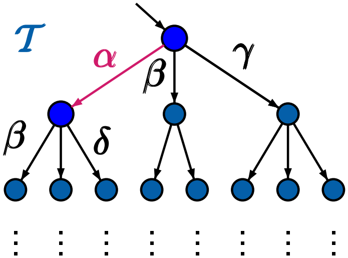
goal: on-the-fly construction of a fragment $\mathcal{T}_{red}$
by selecting action-sets $ample(s) \subseteq Act(s)$
and expanding only the α-successors of s
where $\alpha \in ample(s)$



The ample set method [Peled '93]

given: syntactical representation of processes of TS $\mathcal{T}$

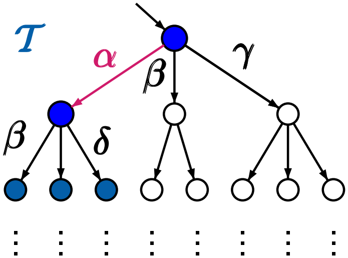
goal: on-the-fly construction of a fragment $\mathcal{T}_{red}$
by selecting action-sets $ample(s) \subseteq Act(s)$
and expanding only the α-successors of s
where $\alpha \in ample(s)$



The ample set method [Peled '93]

given: syntactical representation of processes of TS $\mathcal{T}$

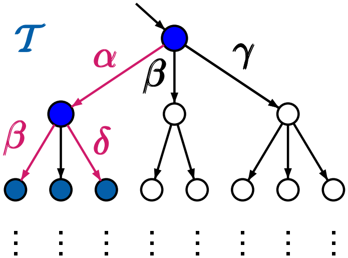
goal: on-the-fly construction of a fragment $\mathcal{T}_{red}$
by selecting action-sets $ample(s) \subseteq Act(s)$
and expanding only the α-successors of s
where $\alpha \in ample(s)$



The ample set method [Peled '93]

given: syntactical representation of processes of TS $\mathcal{T}$

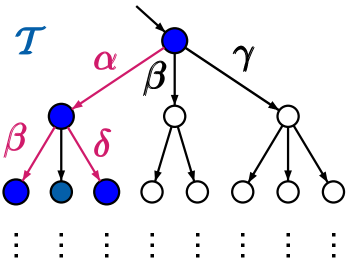
goal: on-the-fly construction of a fragment $\mathcal{T}_{red}$
by selecting action-sets $ample(s) \subseteq Act(s)$
and expanding only the α-successors of s
where $\alpha \in ample(s)$



The ample set method [Peled '93]

given: syntactical representation of processes of TS $\mathcal{T}$

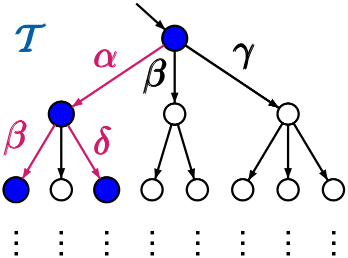
goal: on-the-fly construction of a fragment $\mathcal{T}_{red}$
by selecting action-sets $ample(s) \subseteq Act(s)$
and expanding only the α-successors of s
where $\alpha \in ample(s)$



The ample set method [Peled '93]

given: syntactical representation of processes of TS $\mathcal{T}$

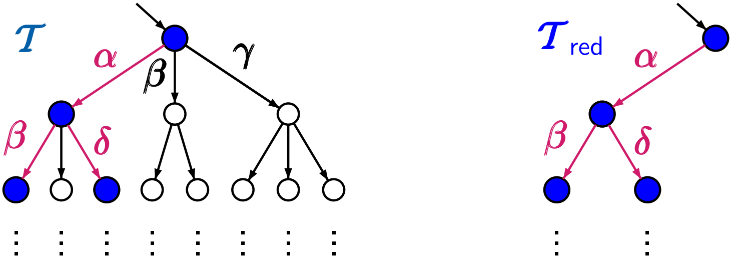
goal: on-the-fly construction of a fragment $\mathcal{T}_{red}$
by selecting action-sets $ample(s) \subseteq Act(s)$
and expanding only the α-successors of s
where $\alpha \in ample(s)$



The ample set method [Peled '93]

given: syntactical representation of processes of TS $\mathcal{T}$

goal: on-the-fly construction of a fragment $\mathcal{T}_{red}$
by selecting action-sets $ample(s) \subseteq Act(s)$
and expanding only the α-successors of s
where $\alpha \in ample(s)$



The ample set method [Peled '93]

given: syntactical representation of processes of TS $\mathcal{T}$

goal: on-the-fly construction of a fragment $\mathcal{T}_{red}$
by selecting action-sets $ample(s) \subseteq Act(s)$
and expanding only the α-successors of s
where $\alpha \in ample(s)$

requirements:

The ample set method [Peled '93]

given: syntactical representation of processes of TS $\mathcal{T}$

goal: on-the-fly construction of a fragment $\mathcal{T}_{red}$
by selecting action-sets $ample(s) \subseteq Act(s)$
and expanding only the α-successors of s
where $\alpha \in ample(s)$

requirements:

- stutter trace equivalence: $\mathcal{T} \triangleq \mathcal{T}_{red}$

The ample set method [Peled '93]

given: syntactical representation of processes of TS $\mathcal{T}$

goal: on-the-fly construction of a fragment $\mathcal{T}_{red}$
by selecting action-sets $ample(s) \subseteq Act(s)$
and expanding only the α-successors of s
where $\alpha \in ample(s)$

requirements:

- stutter trace equivalence: $\mathcal{T} \triangleq \mathcal{T}_{red}$
  hence: $\mathcal{T}, \mathcal{T}_{red}$ are $\mathbf{LTL}_{\setminus \bigcirc}$ equivalent

The ample set method [Peled '93]

given: syntactical representation of processes of TS $\mathcal{T}$

goal: on-the-fly construction of a fragment $\mathcal{T}_{red}$
by selecting action-sets $ample(s) \subseteq Act(s)$
and expanding only the α-successors of s
where $\alpha \in ample(s)$

requirements:

- stutter trace equivalence: $\mathcal{T} \triangleq \mathcal{T}_{red}$
  hence: $\mathcal{T}$, $\mathcal{T}_{red}$ are $\mathbf{LTL}_{\setminus \bigcirc}$ equivalent
- $\mathcal{T}_{red}$ is smaller than $\mathcal{T}$

The ample set method [Peled '93]

given: syntactical representation of processes of TS $\mathcal{T}$

goal: on-the-fly construction of a fragment $\mathcal{T}_{\text{red}}$
by selecting action-sets $\text{ample}(s) \subseteq \text{Act}(s)$
and expanding only the α-successors of s
where $\alpha \in \text{ample}(s)$

requirements:

- stutter trace equivalence: $\mathcal{T} \triangleq \mathcal{T}_{\text{red}}$
  hence: $\mathcal{T}$, $\mathcal{T}_{\text{red}}$ are $\mathbf{LTL}_{\setminus\bigcirc}$ equivalent
- $\mathcal{T}_{\text{red}}$ is smaller than $\mathcal{T}$
- efficient construction of $\mathcal{T}_{\text{red}}$ is possible

The reduced transition system $\mathcal{T}_{red}$

is a fragment of $\mathcal{T}$ that results from $\mathcal{T}$ by

- a DFS-based on-the-fly analysis and
- choosing ample sets $ample(s) \subseteq Act(s)$ for each expanded state,
- expanding only the α-successors of s where $\alpha \in ample(s)$

The reduced transition system $\mathcal{T}_{\text{red}}$

is a fragment of $\mathcal{T}$ that results from $\mathcal{T}$ by

- a DFS-based on-the-fly analysis and
- choosing ample sets $\text{ample}(s) \subseteq Act(s)$ for each expanded state,
- expanding only the α-successors of s where $\alpha \in \text{ample}(s)$

transition relation $\Rightarrow$ of $\mathcal{T}_{\text{red}}$ is given by:

$$\frac{s \xrightarrow{\alpha} s' \ \wedge \ \alpha \in \text{ample}(s)}{s \overset{\alpha}{\Longrightarrow} s'}$$

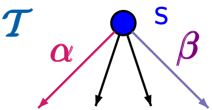
The reduced transition system $\mathcal{T}_{\text{red}}$

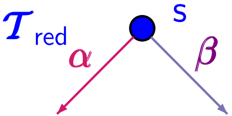
is a fragment of $\mathcal{T}$ that results from $\mathcal{T}$ by

... choosing ample sets $ample(s) \subseteq Act(s)$

transition relation $\Rightarrow$ of $\mathcal{T}_{\text{red}}$ is given by:

$$\frac{s \xrightarrow{\alpha} s' \wedge \alpha \in ample(s)}{s \overset{\alpha}{\Longrightarrow} s'}$$





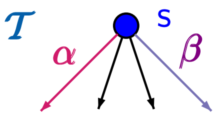
$ample(s) = \{\alpha, \beta\}$

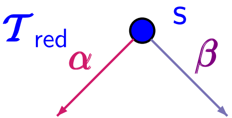
The reduced transition system $\mathcal{T}_{red}$

is a fragment of $\mathcal{T}$ that results from $\mathcal{T}$ by
... choosing ample sets $ample(s) \subseteq Act(s)$

transition relation $\Rightarrow$ of $\mathcal{T}_{red}$ is given by:

$$\frac{s \xrightarrow{\alpha} s' \wedge \alpha \in ample(s)}{s \overset{\alpha}{\Longrightarrow} s'}$$



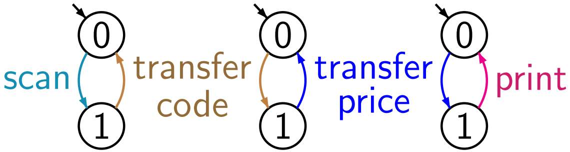


$ample(s) = \{\alpha, \beta\}$

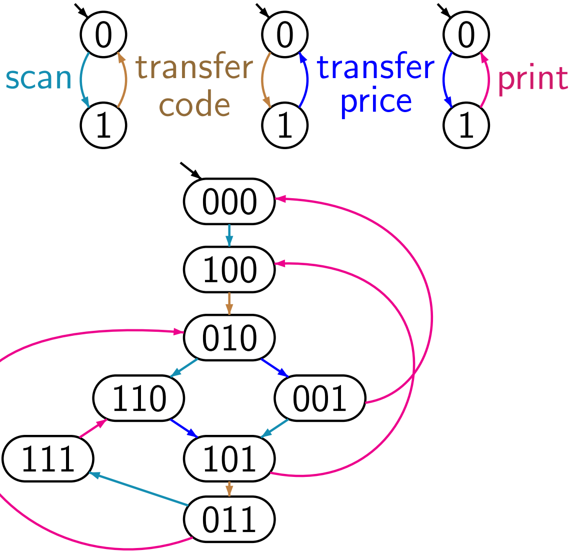
state space S_{red} of $\mathcal{T}_{red}$: all states that are reachable
from the initial states in $\mathcal{T}$ via $\Rightarrow$

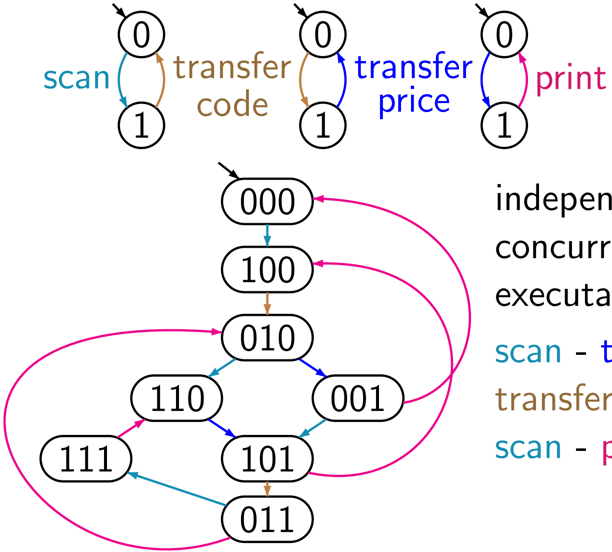
Booking system

$\mathcal{T}$ = SCL ∥ BP ∥ Printer



Booking system





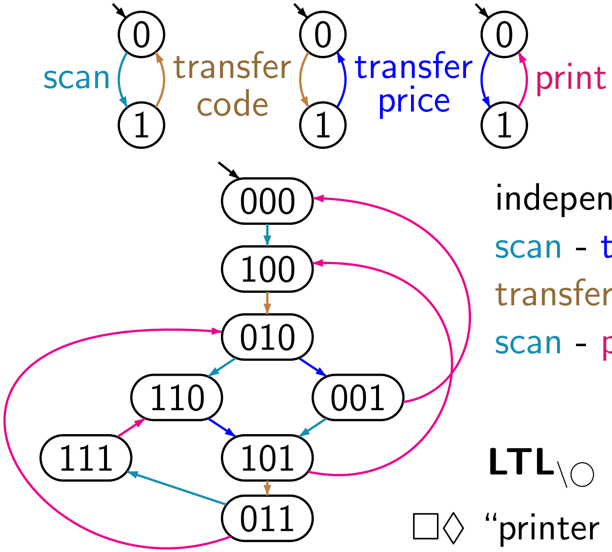
independent, i.e., concurrently executable, actions:

scan - transfer price

transfer code - print

scan - print

Booking system



independent actions:

scan - transf. price

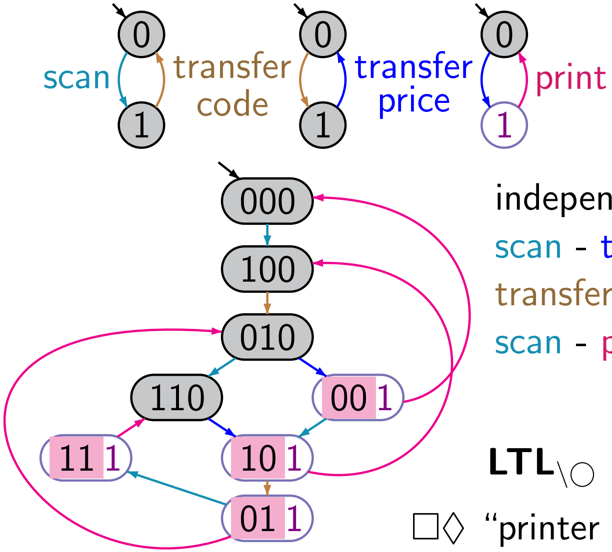
transfer code - print

scan - print

LTL$_{\setminus\bigcirc}$ property:

$\Box\Diamond$ "printer is in state 1"

Booking system



independent actions:

scan - transf. price

transfer code - print

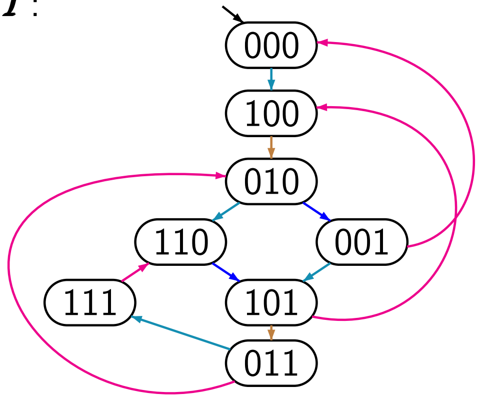
scan - print

LTL$_{\setminus\bigcirc}$ property:

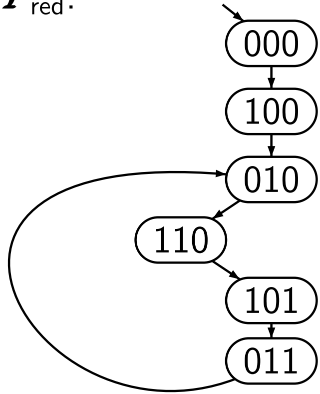
$\Box\Diamond$ "printer is in state 1"

Booking system

$\mathcal{T}$:

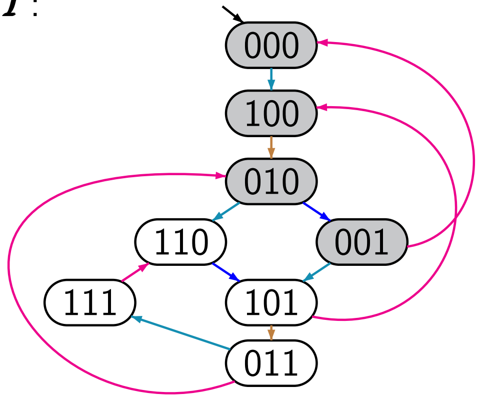


$\mathcal{T}_{red}$:



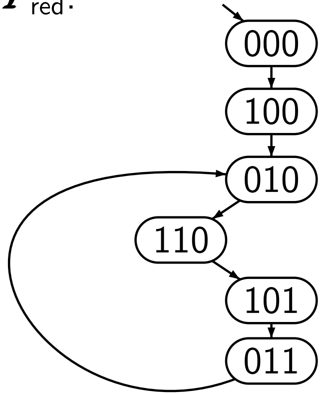
Booking system

$\mathcal{T}$:



scan
code
price
print
scan
code
...

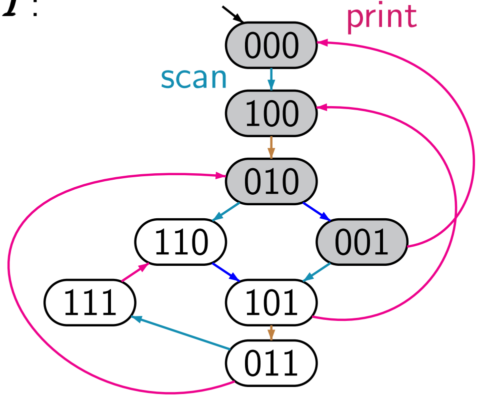
$\mathcal{T}_{red}$:



scan
code
scan
price
code
print
...

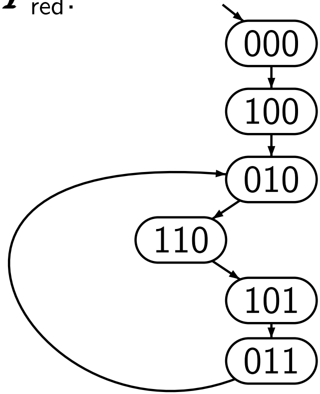
Booking system

$\mathcal{T}$:



scan
code
price
print
scan
code
...

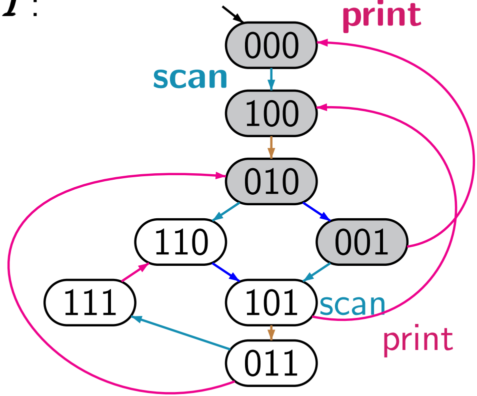
$\mathcal{T}_{red}$:



scan
code
scan
price
code
print
...

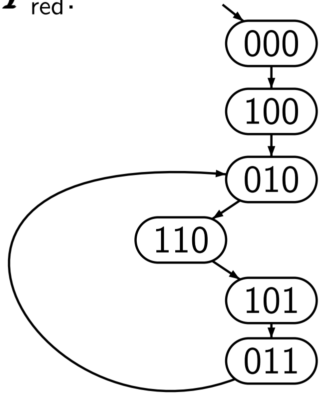
Booking system

$\mathcal{T}$:



scan
code
price
print
scan
code
...

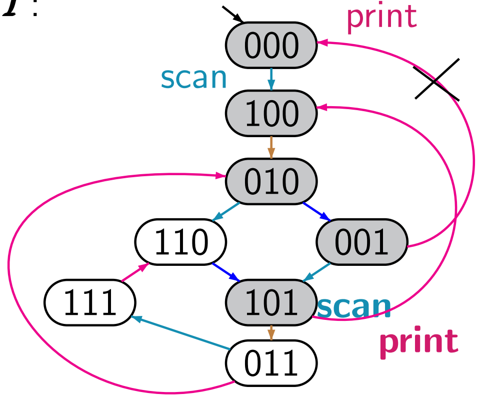
$\mathcal{T}_{red}$:



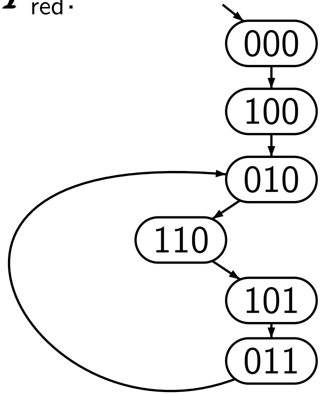
scan
code
scan
price
code
print
...

Booking system

$\mathcal{T}$:



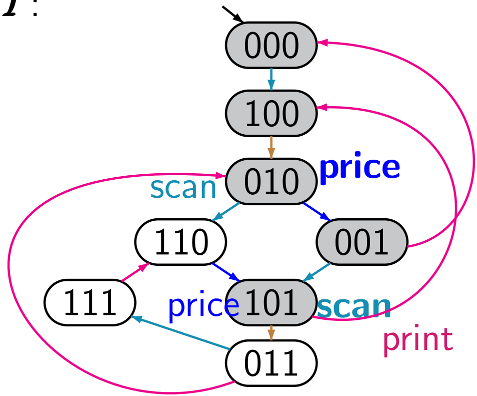
$\mathcal{T}_{red}$:



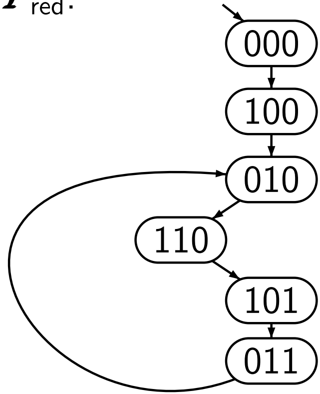
scan		scan	scan
code		code	code
price		price	scan
print	$\rightsquigarrow$	**scan**	price
scan		**print**	code
code		code	print
...		...	...

Booking system

$\mathcal{T}$:



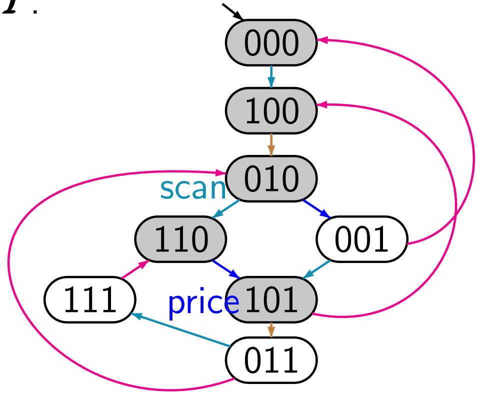
$\mathcal{T}_{red}$:



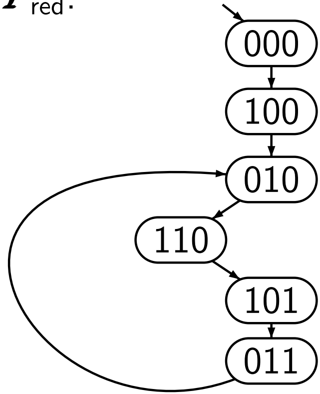
scan		scan	scan
code		code	code
price		**price**	scan
print	$\rightsquigarrow$	**scan**	price
scan		print	code
code		code	print
...		...	...

Booking system

$\mathcal{T}$:



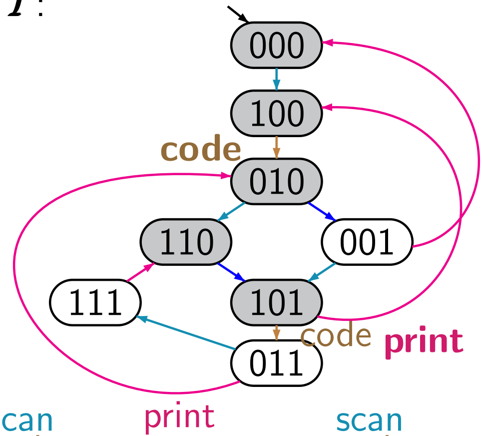
$\mathcal{T}_{red}$:



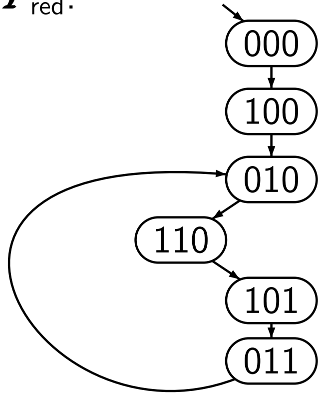
scan		scan	scan
code		code	code
price		**scan**	scan
print	⇝	**price**	price
scan		print	code
code		code	print
...		...	...

Booking system

$\mathcal{T}$:



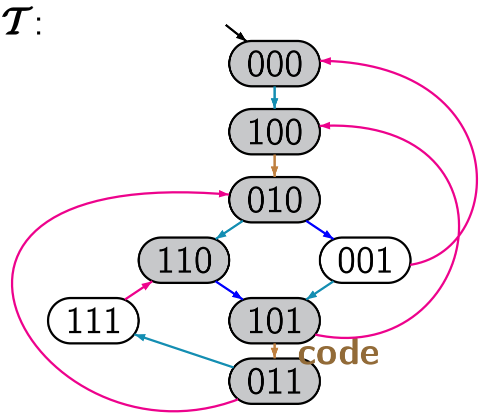
$\mathcal{T}_{\text{red}}$:



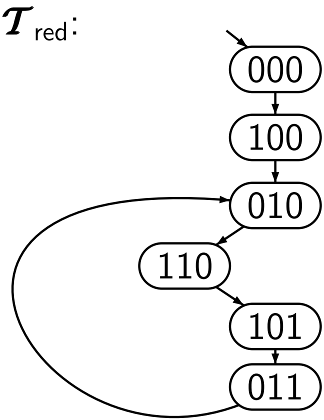
scan		scan	scan
code		code	code
price		scan	scan
print	$\rightsquigarrow$	price	price
scan		**print**	code
code		**code**	print
...		...	...

Booking system

$\mathcal{T}$:



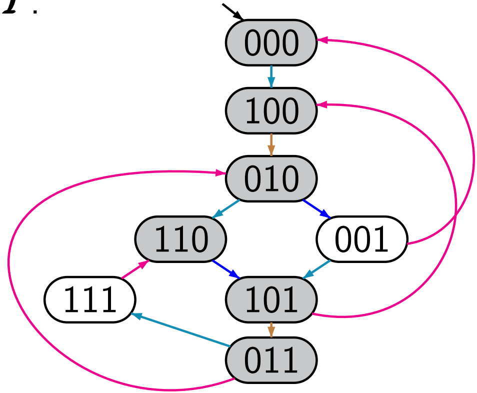
$\mathcal{T}_{\text{red}}$:



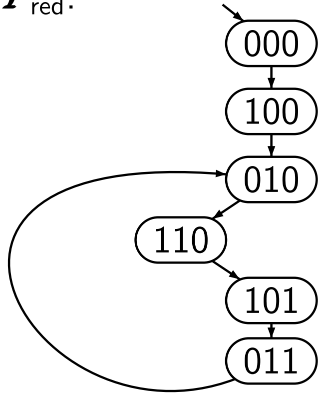
scan		scan	scan
code		code	code
price		scan	scan
print	$\rightsquigarrow$	price	price
scan		**code**	code
code		**print**	print
...		...	...

Booking system

$\mathcal{T}$:



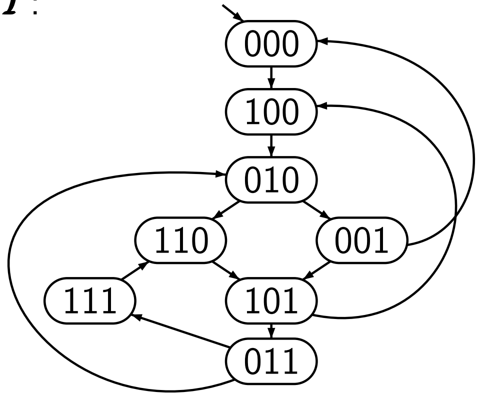
$\mathcal{T}_{red}$:



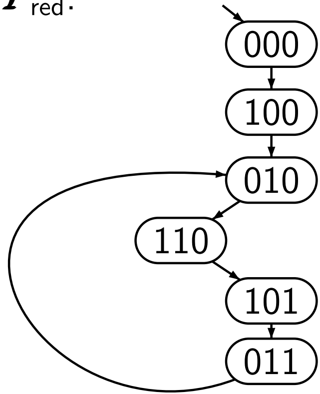
scan		scan		scan
code		code		code
price		scan		scan
print	$\rightsquigarrow$	price	$=$	price
scan		code		code
code		print		print
...		...		...

AP = { "printer in state 1" }

$\mathcal{T}$:

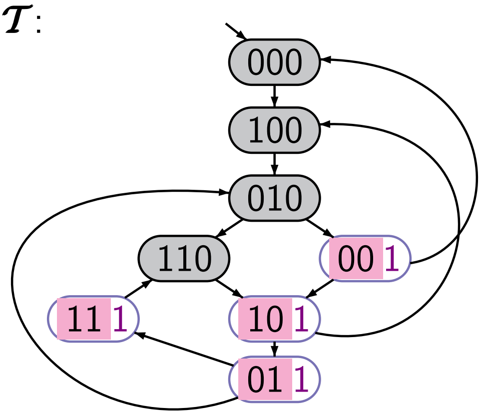


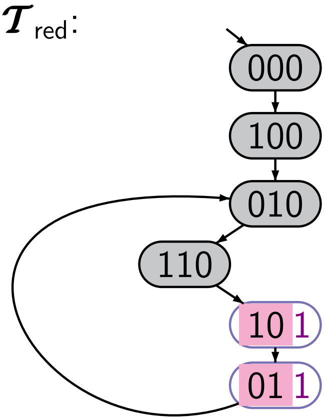
$\mathcal{T}_{red}$:



scan		scan		scan
code		code		code
price		price		scan
print	⇝	scan	⇝	price
scan		print		print
code		code		code

AP = { "printer in state 1" }

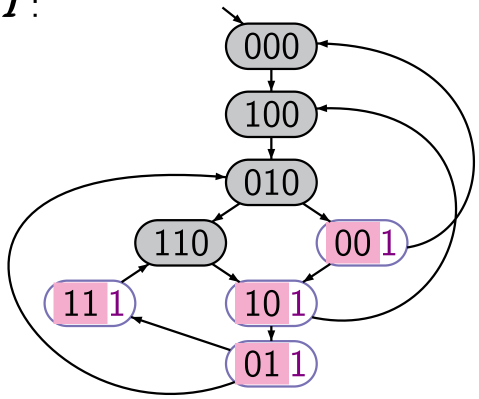




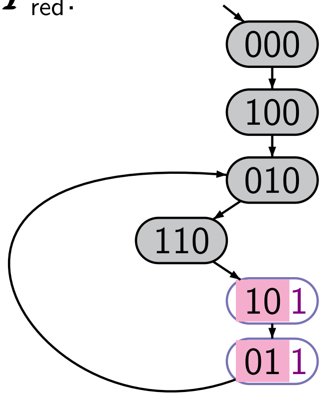
scan	$\emptyset$		scan	$\emptyset$		scan	$\emptyset$
code	$\emptyset$		code	$\emptyset$		code	$\emptyset$
price	$\emptyset$		price	$\emptyset$		scan	$\emptyset$
print	$\{1\}$	$\rightsquigarrow$	scan	$\{1\}$	$\rightsquigarrow$	price	$\emptyset$
scan	$\emptyset$		print	$\{1\}$		print	$\{1\}$
code	$\emptyset$		code	$\emptyset$		code	$\emptyset$

... stuttering equivalent ..

$\mathcal{T}$:



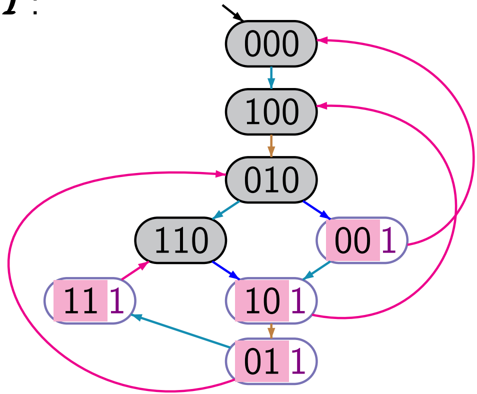
$\mathcal{T}_{red}$:



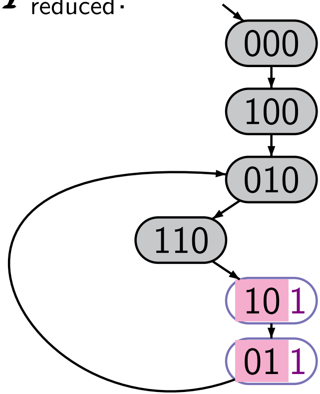
scan	$\emptyset$		scan	$\emptyset$		scan	$\emptyset$
code	$\emptyset$		code	$\emptyset$		code	$\emptyset$
price	$\emptyset$		price	$\emptyset$		scan	$\emptyset$
print	$\{1\}$	$\rightsquigarrow$	scan	$\{1\}$	$\rightsquigarrow$	price	$\emptyset$
scan	$\emptyset$		print	$\{1\}$		print	$\{1\}$
code	$\emptyset$		code	$\emptyset$		code	$\emptyset$

Booking system

$\mathcal{T}$:

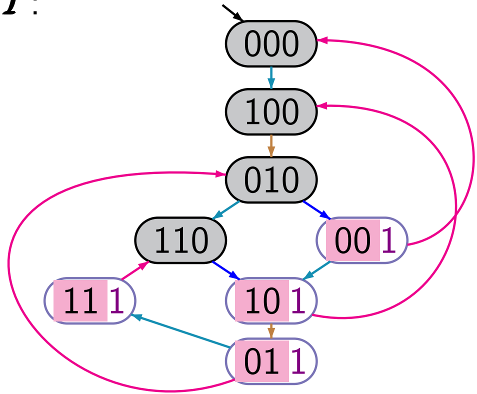


$\mathcal{T}_{\text{reduced}}$:

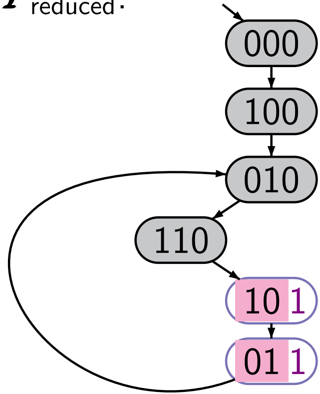


Booking system

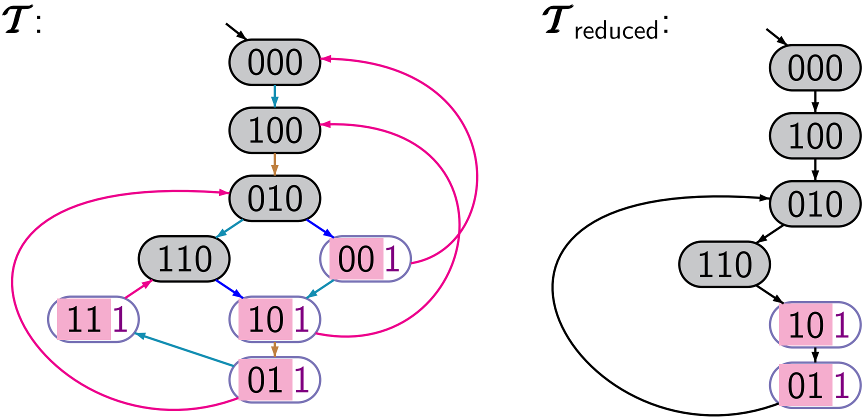
$\mathcal{T}$:



$\mathcal{T}_{\text{reduced}}$:



$\mathcal{T} \triangleq \mathcal{T}_{\text{red}}$



$\mathcal{T} \triangleq \mathcal{T}_{\text{red}}$ hence: $\mathcal{T}_{\text{red}} \models \varphi$ implies $\mathcal{T} \models \varphi$ where

$\varphi = \Box\Diamond$ "printer is in control state 1"

Action-determinism

LTL3.4-11A

Action-determinism

Let $\mathcal{T} = (\mathbf{S}, \mathit{Act}, \rightarrow, \mathbf{S}_0, \mathsf{AP}, \mathsf{L})$ be a transition system.

Action-determinism

Let $\mathcal{T} = (\mathbf{S}, Act, \rightarrow, \mathbf{S}_0, \mathrm{AP}, \mathrm{L})$ be a transition system.

For state $\mathbf{s}$:

$$Act(\mathbf{s}) \;=\; \{\, \alpha \in Act \;:\; \exists \mathbf{t} \in \mathbf{S} \text{ s.t. } \mathbf{s} \xrightarrow{\alpha} \mathbf{t} \,\}$$

Action-determinism

Let $\mathcal{T} = (\mathbf{S}, Act, \rightarrow, \mathbf{S}_0, \mathrm{AP}, \mathrm{L})$ be a transition system.

For state $\mathbf{s}$:

$$Act(\mathbf{s}) \;=\; \{\, \alpha \in Act \;:\; \exists \mathbf{t} \in \mathbf{S} \text{ s.t. } \mathbf{s} \xrightarrow{\alpha} \mathbf{t} \,\}$$

$\mathcal{T}$ is called action-deterministic iff for all states $\mathbf{s}$ and all actions $\alpha \in Act(\mathbf{s})$:

$$|\, \{\mathbf{t} \in \mathbf{S} \;:\; \mathbf{s} \xrightarrow{\alpha} \mathbf{t}\,\} \,| \;\leq\; 1$$

Action-determinism

Let $\mathcal{T} = (S, Act, \rightarrow, S_0, AP, L)$ be a TS.

For state s:

$$Act(s) \; = \; \{ \alpha \in Act \; : \; \exists t \in S \text{ s.t. } s \xrightarrow{\alpha} t \}$$

$\mathcal{T}$ is called action-deterministic iff for all states s and all actions $\alpha \in Act(s)$:

$$|\{t \in S \; : \; s \xrightarrow{\alpha} t \}| \leq 1$$

notation: if $\alpha \in Act(s)$ then

$$\alpha(s) = \text{unique state } t \text{ s.t. } s \xrightarrow{\alpha} t$$

Independence of actions

LTL3.4-11

Independence of actions

Let $\mathcal{T}$ be an action-deterministic transition system with action-set Act, and $\alpha, \beta \in Act$.

Independence of actions

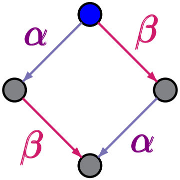
Let $\mathcal{T}$ be an action-deterministic transition system with action-set Act, and $\alpha, \beta \in Act$.

α, β are called independent in $\mathcal{T}$ if for all states s s.t. $\alpha, \beta \in Act(s)$:

Independence of actions

Let $\mathcal{T}$ be an action-deterministic transition system with action-set *Act*, and $\alpha, \beta \in Act$.

α, β are called independent in $\mathcal{T}$ if for all states s s.t. $\alpha, \beta \in Act(s)$:

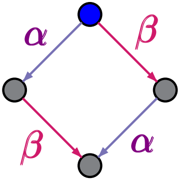


Independence of actions

Let $\mathcal{T}$ be an action-deterministic transition system with action-set Act, and $\alpha, \beta \in Act$.

α, β are called independent in $\mathcal{T}$ if for all states s s.t. $\alpha, \beta \in Act(s)$:

1. $\beta \in Act(\alpha(s))$

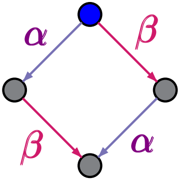


Independence of actions

Let $\mathcal{T}$ be an action-deterministic transition system with action-set Act, and $\alpha, \beta \in Act$.

α, β are called independent in $\mathcal{T}$ if for all states s s.t. $\alpha, \beta \in Act(s)$:

1. $\beta \in Act(\alpha(s))$
2. $\alpha \in Act(\beta(s))$

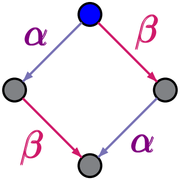


Independence of actions

Let $\mathcal{T}$ be an action-deterministic transition system with action-set Act, and $\alpha, \beta \in Act$.

α, β are called independent in $\mathcal{T}$ if for all states s s.t. $\alpha, \beta \in Act(s)$:

1. $\beta \in Act(\alpha(s))$
2. $\alpha \in Act(\beta(s))$
3. $\beta(\alpha(s)) = \alpha(\beta(s))$

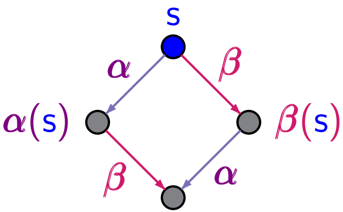


Independence of actions

Let $\mathcal{T}$ be an action-deterministic transition system with action-set Act, and $\alpha, \beta \in Act$.

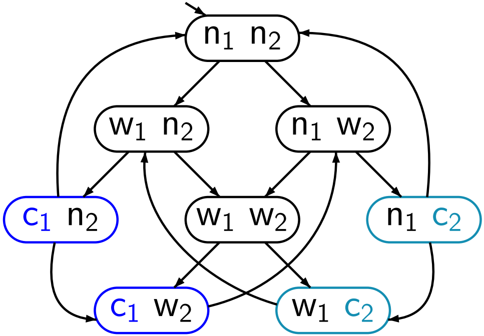
α, β are called independent in $\mathcal{T}$ if for all states s s.t. $\alpha, \beta \in Act(s)$:

1. $\beta \in Act(\alpha(s))$
2. $\alpha \in Act(\beta(s))$
3. $\beta(\alpha(s)) = \alpha(\beta(s))$

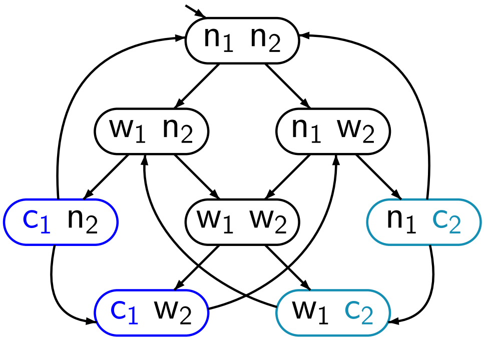


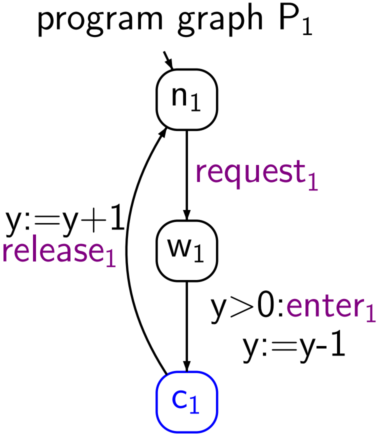
$\beta(\alpha(s)) = \alpha(\beta(s))$

Mutual exclusion with semaphore

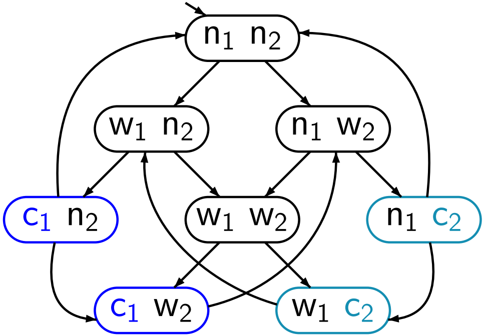


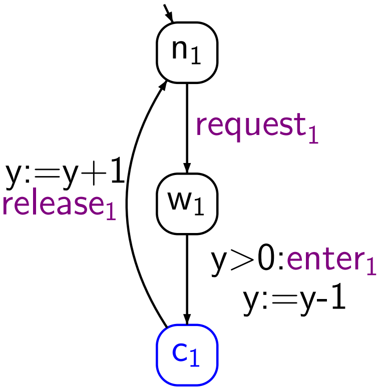
Mutual exclusion with semaphore





Mutual exclusion with semaphore

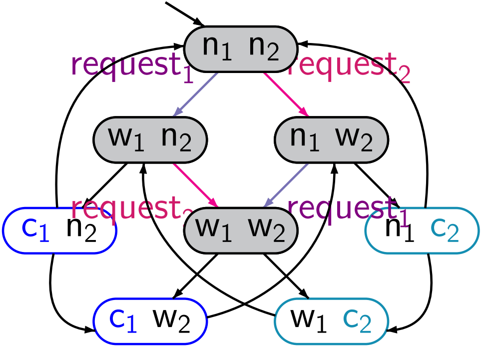




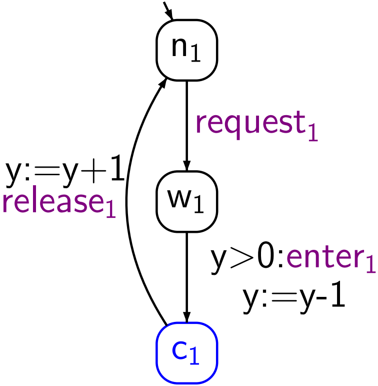
independent actions:

request_1, request_2

Mutual exclusion with semaphore



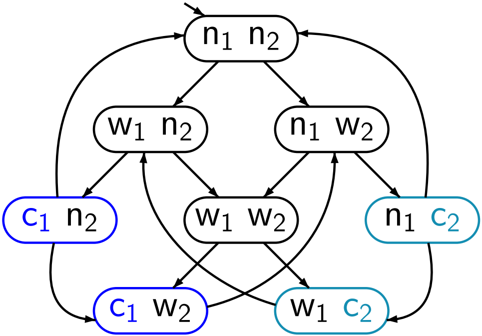
program graph P_1



independent actions:

$request_1$, $request_2$

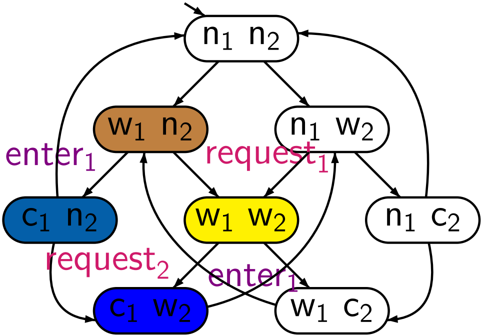
Example: independent actions for MUTEX



independent actions:

$request_1$, $request_2$

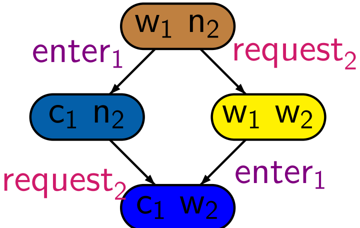
Example: independent actions for MUTEX



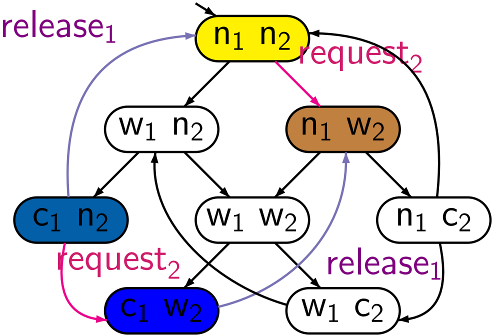
independent actions:

$request_1$, $request_2$

$enter_1$, $request_2$



Example: independent actions for MUTEX

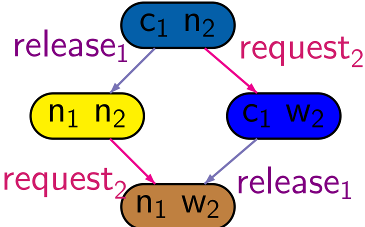


independent actions:

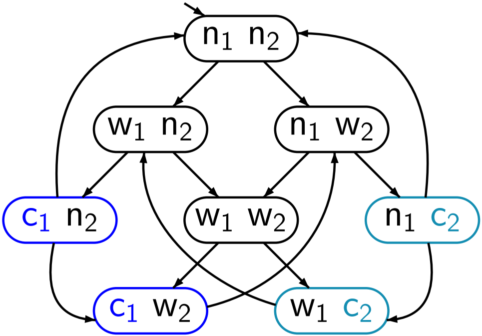
request$_1$, request$_2$

enter$_1$, request$_2$

release$_1$, request$_2$



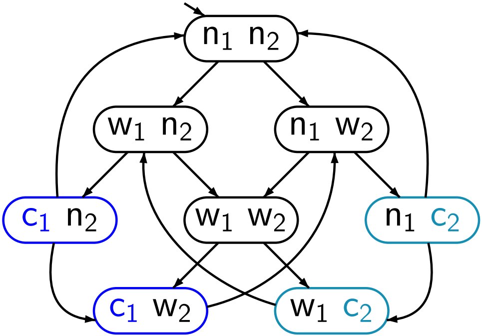
Example: independent actions for MUTEX



independent actions:

$\text{request}_1, \text{request}_2$
$\text{enter}_1, \text{request}_2$
$\text{release}_1, \text{request}_2$
$\text{request}_1, \text{enter}_2$
$\text{request}_1, \text{release}_2$

Example: independent actions for MUTEX

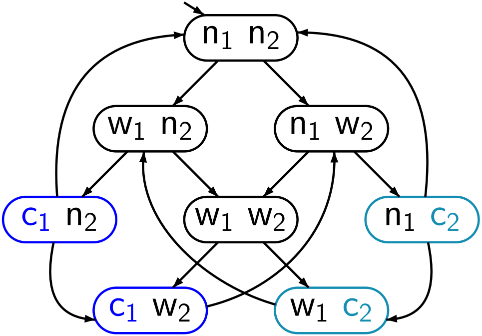


independent actions:

$request_1$, $request_2$
$enter_1$, $request_2$
$release_1$, $request_2$
$request_1$, $enter_2$
$request_1$, $release_2$

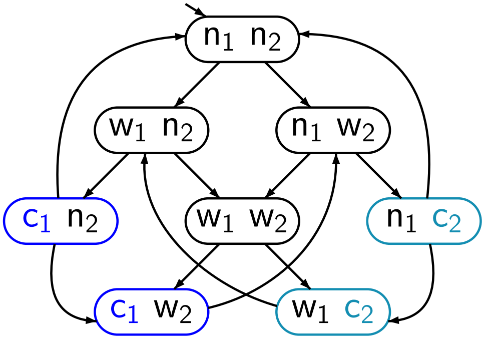
$request_1$ is independent from the action-set $\{request_2, enter_2, release_2\}$

Example: dependent actions for MUTEX



dependent actions:

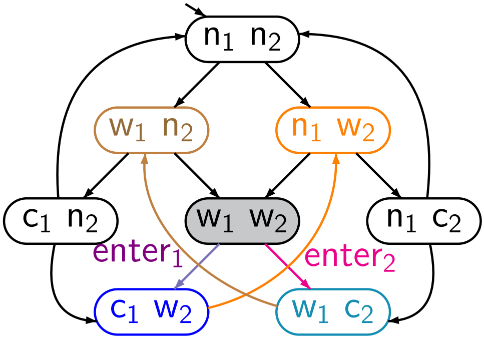
Example: dependent actions for MUTEX

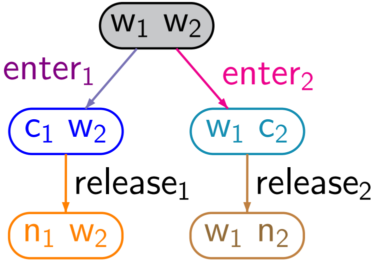


dependent actions:

$enter_1$, $enter_2$

Example: dependent actions for MUTEX



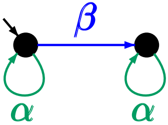


dependent actions:

$enter_1$, $enter_2$

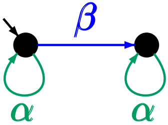
access both to the semaphore

Correct or wrong?



α and β are independent **?**

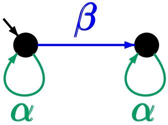
Correct or wrong?



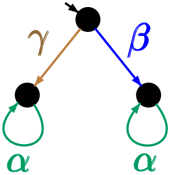
α and β are independent ✓

Correct or wrong?

LTL3.4-15

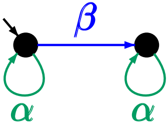


α and β are independent √

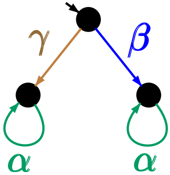


α and β are independent **?**

Correct or wrong?



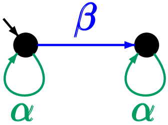
α and β are independent $\surd$



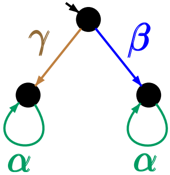
α and β are independent $\surd$

Correct or wrong?

LTL3.4-15



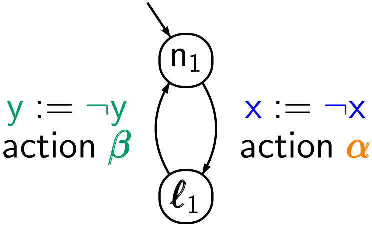
α and β are independent $\surd$

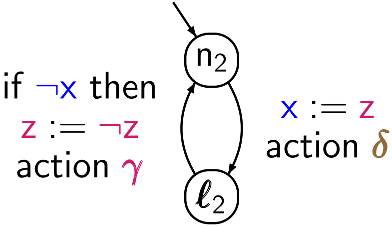


α and β are independent $\surd$

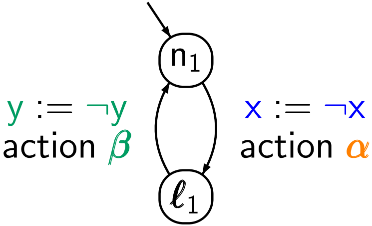
note: there is no state in which α and β are enabled

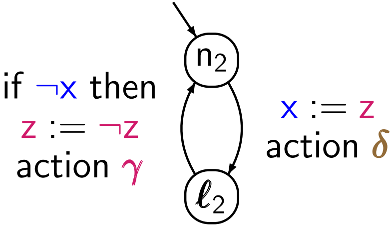
Independent or not?





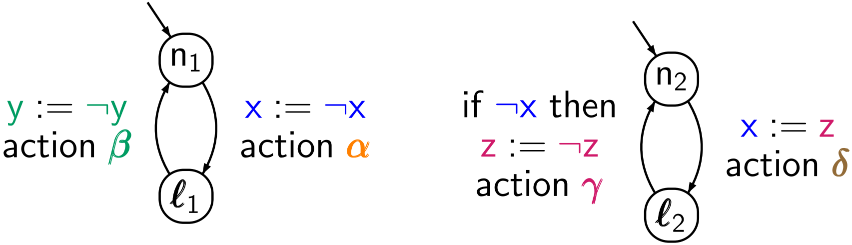
Independent or not?





Are actions α, δ independent for $\mathcal{T}_{P_1 \,|||\, P_2}$?

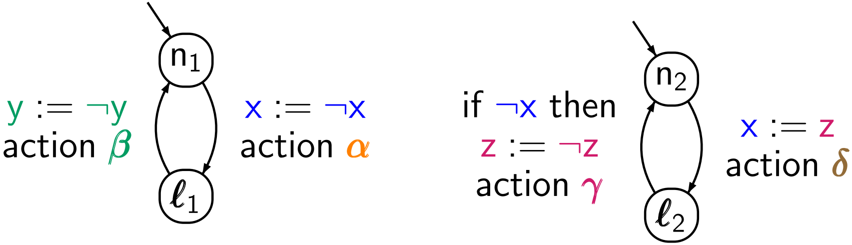
Independent or not?



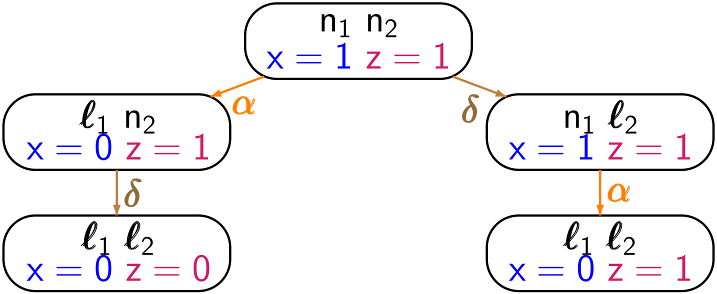
Are actions α, δ independent for $\mathcal{T}_{P_1 \,|||\, P_2}$?

no

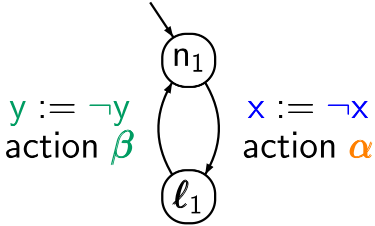
Independent or not?

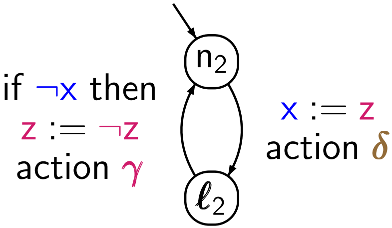


α, δ are dependent for $\mathcal{T}_{P_1 \,|||\, P_2}$



Independent or not?

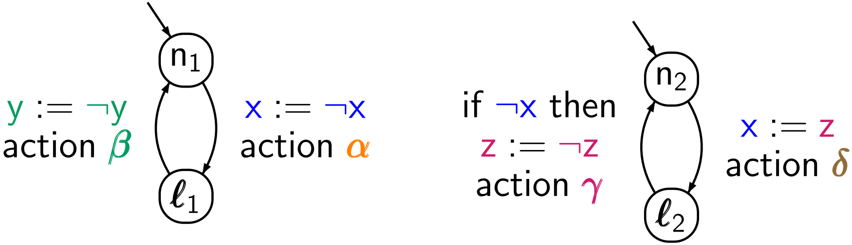




α, δ are dependent

β, δ are independent **?**

Independent or not?

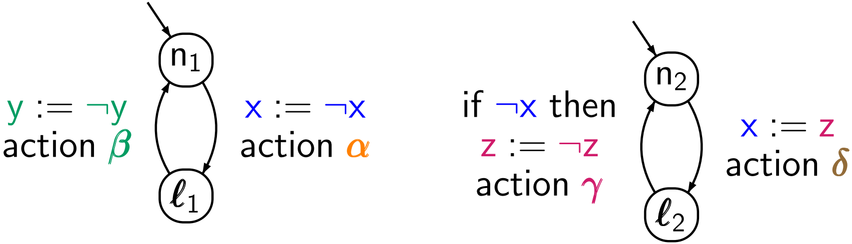


α, δ are dependent

β, δ are independent **?**

yes

Independent or not?

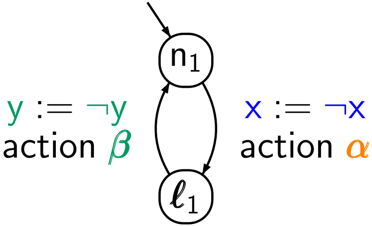


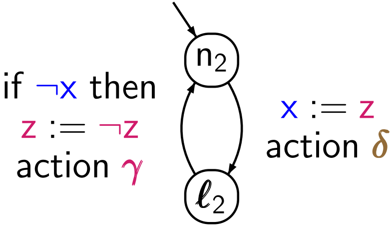
α, δ are dependent

β, δ are independent,

as they access different variables

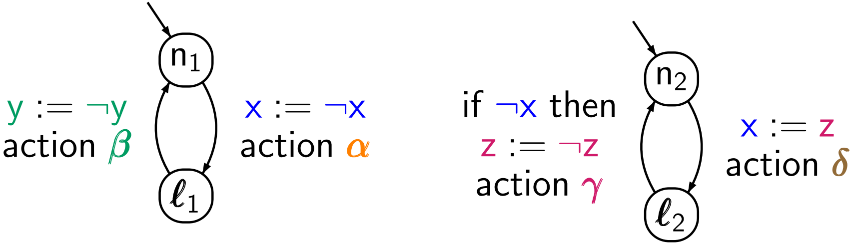
Independent or not?





α, γ are independent **?**

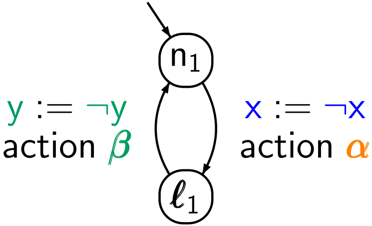
Independent or not?

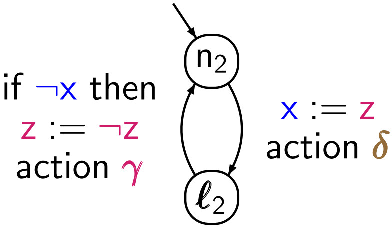


α, γ are independent **?**

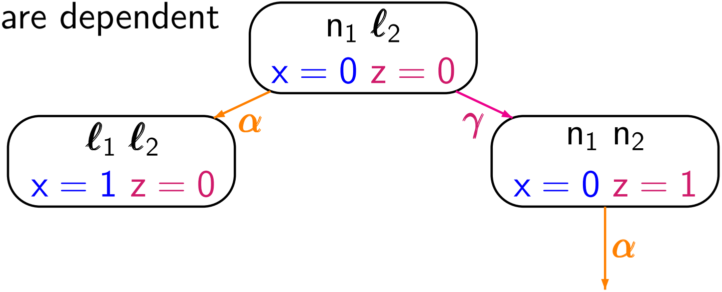
no

Independent or not?

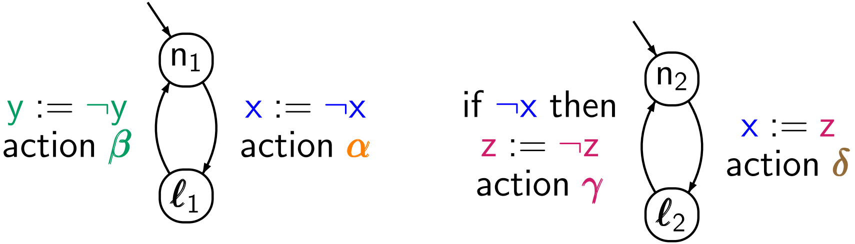




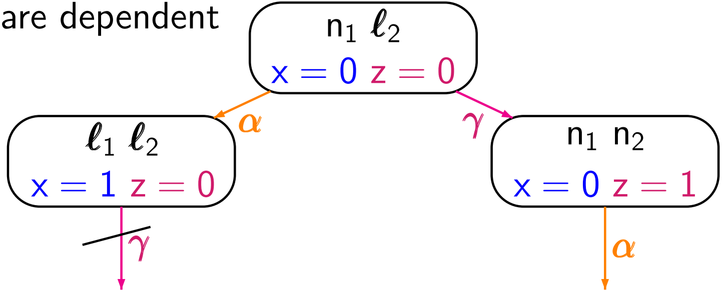
α, γ are dependent



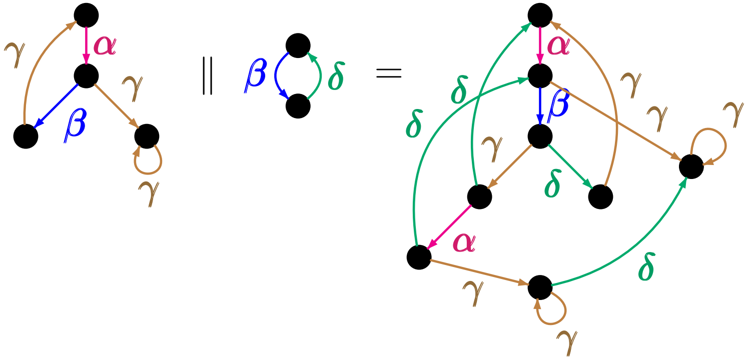
Independent or not?



α, γ are dependent

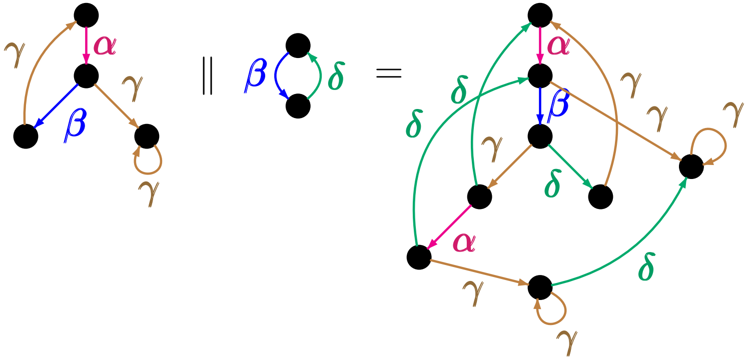


Independent or not?



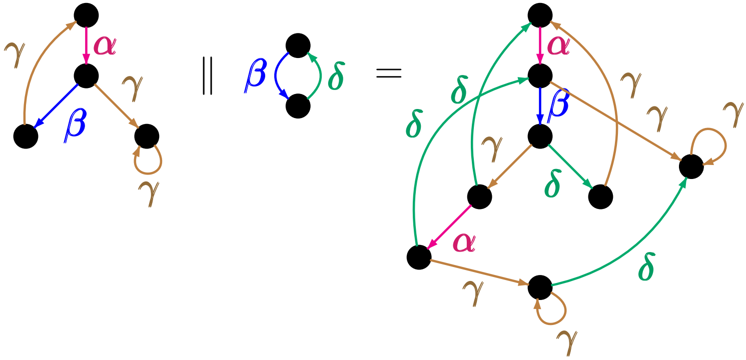
TS that results by synchronization over the common action β

Independent or not?



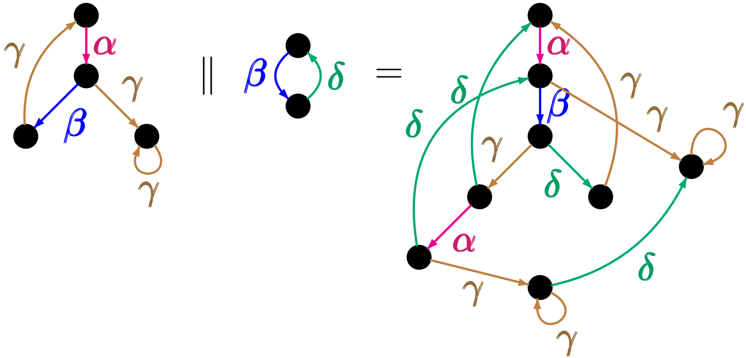
α, δ are independent ?

Independent or not?



α, δ independent $\surd$

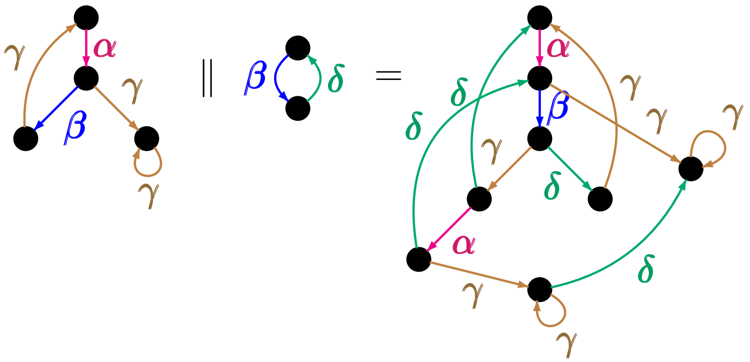
Independent or not?



α, δ independent $\surd$

γ, δ are independent **?**

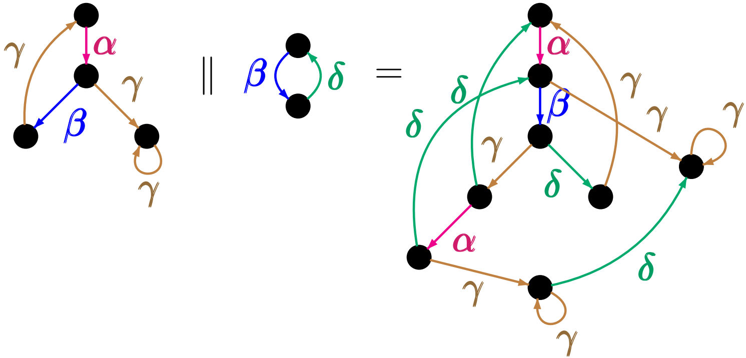
Independent or not?



α, δ independent $\surd$

γ, δ independent $\surd$

Independent or not?

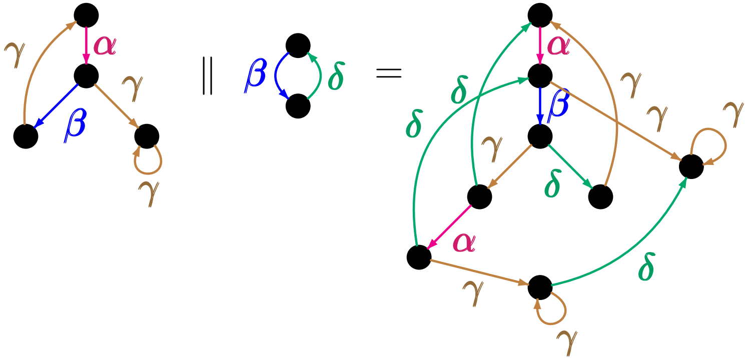


α, δ independent $\surd$

γ, δ independent $\surd$

β, γ are independent **?**

Independent or not?

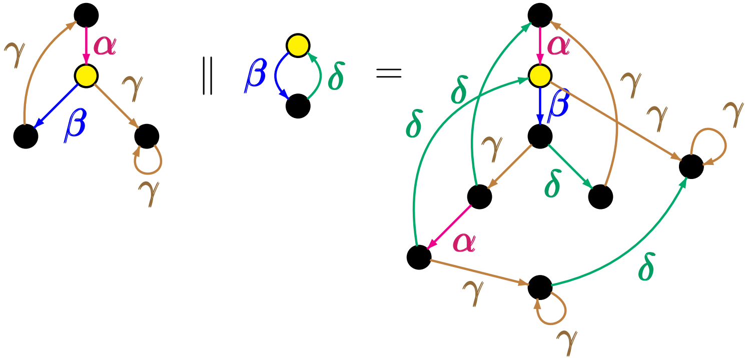


α, δ independent $\surd$

γ, δ independent $\surd$

β, γ dependent

Independent or not?



α, δ independent $\surd$

γ, δ independent $\surd$

β, γ dependent

Permutation of independent actions

Permutation of independent actions

Let α is independent from $\beta_1, \ldots, \beta_n$.

Permutation of independent actions

Let α is independent from $\beta_1, \ldots, \beta_n$. I.e., for $1 \leq \mathsf{i} \leq \mathsf{n}$, the actions α and β_i are independent.

Permutation of independent actions

Let α is independent from $\beta_1, \ldots, \beta_n$. I.e., for $1 \leq i \leq n$, the actions α and β_i are independent.

Then the action sequence

$$\beta_1 \, \beta_2 \, \ldots \, \beta_n \, \alpha$$

can be replaced with the action sequence

$$\alpha \, \beta_1 \, \beta_2 \, \ldots \, \beta_n$$

Permutation of independent actions

Let $\alpha \in Act(s_0)$ and

$$s_0 \xrightarrow{\beta_1} s_1 \xrightarrow{\beta_2} \ldots \xrightarrow{\beta_n} s_n \xrightarrow{\alpha} t$$

be a path fragment such that α is independent from $\beta_1, \ldots, \beta_n$.

Permutation of independent actions

Let $\alpha \in Act(s_0)$ and

$$s_0 \xrightarrow{\beta_1} s_1 \xrightarrow{\beta_2} \ldots \xrightarrow{\beta_n} s_n \xrightarrow{\alpha} t$$

be a path fragment such that α is independent from $\beta_1, \ldots, \beta_n$.

Then there exists a path fragment

$$s_0 \xrightarrow{\alpha} t_0 \xrightarrow{\beta_1} t_1 \xrightarrow{\beta_2} \ldots \xrightarrow{\beta_n} t_n$$

Permutation of independent actions

Let $\alpha \in Act(s_0)$ and

$$s_0 \xrightarrow{\beta_1} s_1 \xrightarrow{\beta_2} \ldots \xrightarrow{\beta_n} s_n \xrightarrow{\alpha} t$$

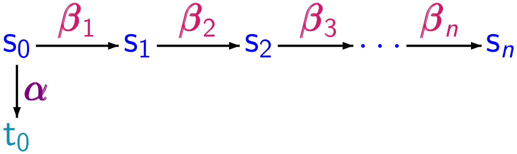
be a path fragment such that α is independent from $\beta_1, \ldots, \beta_n$.

Then there exists a path fragment

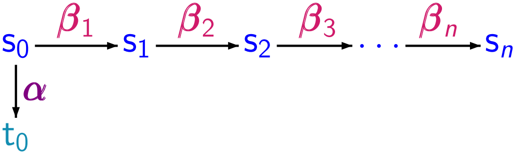
$$s_0 \xrightarrow{\alpha} t_0 \xrightarrow{\beta_1} t_1 \xrightarrow{\beta_2} \ldots \xrightarrow{\beta_n} t_n$$

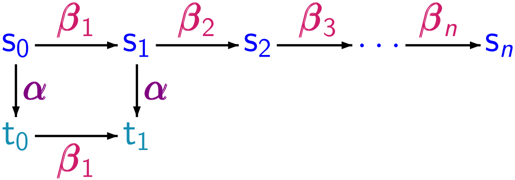
with $t_n = t$

α independent from $\beta_1, \ldots, \beta_n$

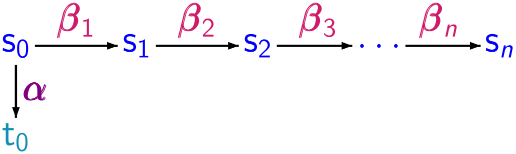


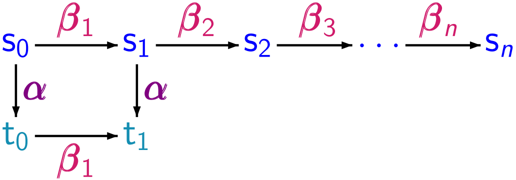
α independent from $\beta_1, \ldots, \beta_n$

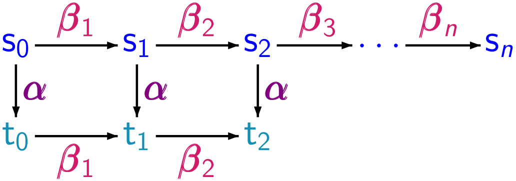




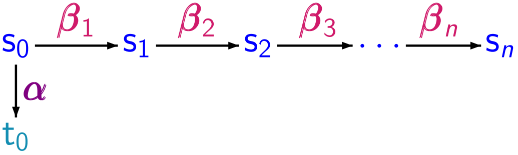
α independent from $\beta_1, \ldots, \beta_n$

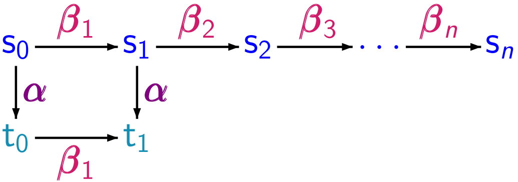


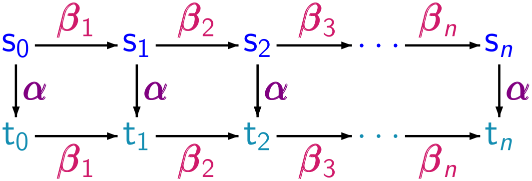




α independent from $\beta_1, \ldots, \beta_n$







The ample set method

given: action-deterministic, finite transition system

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

goal: define "small" action-sets $\text{ample}(s) \subseteq Act(s)$

for all states $s \in S$ s.t.

The ample set method for $\mathrm{LTL}_{\setminus\bigcirc}$

given: action-deterministic, finite transition system

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

goal: define "small" action-sets $ample(s) \subseteq Act(s)$ for all states $s \in S$ s.t.

$$\mathcal{T} \triangleq \mathcal{T}_{red}$$

where $\mathcal{T}_{red} = (S, Act, \Longrightarrow, S_0, AP, L)$ results from $\mathcal{T}$ by an on-the-fly construction

The ample set method for $\text{LTL}_{\setminus\bigcirc}$

given: action-deterministic, finite transition system

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

goal: define "small" action-sets $ample(s) \subseteq Act(s)$ for all states $s \in S$ s.t.

$$\mathcal{T} \triangleq \mathcal{T}_{red}$$

where $\mathcal{T}_{red} = (S, Act, \Longrightarrow, S_0, AP, L)$ results from $\mathcal{T}$ by an on-the-fly construction using the reduced transition relation

$$\frac{s \xrightarrow{\alpha} s' \wedge \alpha \in ample(s)}{s \overset{\alpha}{\Longrightarrow} s'}$$

The ample set method for $\mathrm{LTL}_{\setminus\bigcirc}$

given: action-deterministic, finite transition system

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

goal: define "small" action-sets $ample(s) \subseteq Act(s)$ for all states $s \in S$ s.t.

$$\mathcal{T} \triangleq \mathcal{T}_{red}$$

where $\mathcal{T}_{red} = (S', Act, \Longrightarrow, S_0, AP, L)$ results from $\mathcal{T}$ by an on-the-fly construction using the reduced transition relation

$$\frac{s \xrightarrow{\alpha} s' \wedge \alpha \in ample(s)}{s \overset{\alpha}{\Longrightarrow} s'}$$

S' = reachable fragment w.r.t. $\Longrightarrow$

given: syntactic representation of the processes of an action-deterministic, finite transition system

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

goal: define "small" action-sets $ample(s) \subseteq Act(s)$

for all states $s \in S$ s.t. $\mathcal{T} \triangleq \mathcal{T}_{red}$

Conditions for the ample sets

given: syntactic representation of the processes of an action-deterministic, finite transition system

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

goal: define "small" action-sets $ample(s) \subseteq Act(s)$ for all states $s \in S$ s.t. $\mathcal{T} \triangleq \mathcal{T}_{red}$

provide *conditions* for the ample sets that

Conditions for the ample sets

given: syntactic representation of the processes of an action-deterministic, finite transition system

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

goal: define "small" action-sets $ample(s) \subseteq Act(s)$ for all states $s \in S$ s.t. $\mathcal{T} \triangleq \mathcal{T}_{red}$

provide *conditions* for the ample sets that

- are suitable for an on-the-fly construction of $\mathcal{T}_{red}$

Conditions for the ample sets

given: syntactic representation of the processes of an action-deterministic, finite transition system

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

goal: define "small" action-sets $ample(s) \subseteq Act(s)$ for all states $s \in S$ s.t. $\mathcal{T} \triangleq \mathcal{T}_{red}$

provide *conditions* for the ample sets that

- are suitable for an on-the-fly construction of $\mathcal{T}_{red}$
- can be checked efficiently (without analyzing $\mathcal{T}$)

Conditions for the ample sets

given: syntactic representation of the processes of an action-deterministic, finite transition system

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

goal: define "small" action-sets $ample(s) \subseteq Act(s)$ for all states $s \in S$ s.t. $\mathcal{T} \triangleq \mathcal{T}_{red}$

provide *conditions* for the ample sets that

- are suitable for an on-the-fly construction of $\mathcal{T}_{red}$
- can be checked efficiently (without analyzing $\mathcal{T}$)
- ensure that $\mathcal{T} \triangleq \mathcal{T}_{red}$

Stutter trace equivalence of $\mathcal{T}$ and $\mathcal{T}_{\text{red}}$

idea: the conditions for the ample sets

should ensure that

for each execution ρ in $\mathcal{T}$,

a stutter trace equivalent execution ρ_{red} in $\mathcal{T}_{\text{red}}$

can be constructed

Stutter trace equivalence of $\mathcal{T}$ and $\mathcal{T}_{\mathsf{red}}$

idea: the conditions for the ample sets
should ensure that

for each execution ρ in $\mathcal{T}$,

a stutter trace equivalent execution ρ_{red} in $\mathcal{T}_{\mathsf{red}}$

can be constructed by successively

- permutating the order independent actions

Stutter trace equivalence of $\mathcal{T}$ and $\mathcal{T}_{\text{red}}$

idea: the conditions for the ample sets
should ensure that
for each execution ρ in $\mathcal{T}$,
a stutter trace equivalent execution ρ_{red} in $\mathcal{T}_{\text{red}}$
can be constructed by successively

- permutating the order independent actions
- adding independent stutter actions

Stutter trace equivalence of $\mathcal{T}$ and $\mathcal{T}_{\text{red}}$

idea: the conditions for the ample sets
should ensure that

for each execution ρ in $\mathcal{T}$,
a stutter trace equivalent execution ρ_{red} in $\mathcal{T}_{\text{red}}$
can be constructed by successively

- permutating the order independent actions
- adding independent stutter actions

execution ρ in $\mathcal{T}$ $\rightsquigarrow$ execution ρ_{red} in $\mathcal{T}_{\text{red}}$ s.t. $\rho \overset{\Delta}{=} \rho_{\text{red}}$

Stutter trace equivalence of $\mathcal{T}$ and $\mathcal{T}_{\mathsf{red}}$

execution ρ in $\mathcal{T}$ $\rightsquigarrow$ execution ρ_{red} in $\mathcal{T}_{\mathsf{red}}$ s.t. $\rho \overset{\Delta}{=} \rho_{\mathsf{red}}$

Stutter trace equivalence of $\mathcal{T}$ and $\mathcal{T}_{\text{red}}$

execution $\boldsymbol{\rho}$ in $\mathcal{T}$ $\rightsquigarrow$ execution $\boldsymbol{\rho}_{\text{red}}$ in $\mathcal{T}_{\text{red}}$ s.t. $\boldsymbol{\rho} \stackrel{\Delta}{=} \boldsymbol{\rho}_{\text{red}}$

by successively applying the following transformations:

execution ρ in $\mathcal{T}$ $\rightsquigarrow$ execution ρ_{red} in $\mathcal{T}_{\text{red}}$ s.t. $\rho \overset{\Delta}{=} \rho_{\text{red}}$

case 0: $\rho = s_0 \xrightarrow{\alpha} s_0' \longrightarrow \ldots$ with $\alpha \in ample(s_0)$
case 1: $\rho = s_0 \xrightarrow{\beta_1} \ldots \xrightarrow{\beta_n} \xrightarrow{\alpha} \longrightarrow \ldots$ with $\alpha \in ample(s_0)$, $\beta_i \notin ample(s_0)$
case 2: $\rho = s_0 \xrightarrow{\beta_1} \xrightarrow{\beta_2} \xrightarrow{\beta_3} \ldots$ with $\beta_i \notin ample(s_0)$

Stutter trace equivalence of $\mathcal{T}$ and $\mathcal{T}_{\text{red}}$

execution ρ in $\mathcal{T}$ $\rightsquigarrow$ execution ρ_{red} in $\mathcal{T}_{\text{red}}$ s.t. $\rho \triangleq \rho_{\text{red}}$

case 0: $\rho =$	$s_0 \xrightarrow{\alpha} s_0' \longrightarrow \ldots$ with $\alpha \in \text{ample}(s_0)$ $s_0 \xRightarrow{\alpha} s_0' \longrightarrow \ldots$	
case 1: $\rho =$	$s_0 \xrightarrow{\beta_1} \ldots \xrightarrow{\beta_n} \xrightarrow{\alpha} \longrightarrow \ldots$ with $\alpha \in \text{ample}(s_0)$ $\beta_i \notin \text{ample}(s_0)$	
case 2: $\rho =$	$s_0 \xrightarrow{\beta_1} \xrightarrow{\beta_2} \xrightarrow{\beta_3} \ldots$ with $\beta_i \notin \text{ample}(s_0)$	

execution ρ in $\mathcal{T}$ $\rightsquigarrow$ execution ρ_{red} in $\mathcal{T}_{red}$ s.t. $\rho \triangleq \rho_{red}$

case 0:	$\rho =$	$s_0 \xrightarrow{\alpha} s_0' \longrightarrow \ldots$ with $\alpha \in ample(s_0)$
		$s_0 \overset{\alpha}{\Longrightarrow} s_0' \longrightarrow \ldots$
case 1:	$\rho =$	$s_0 \xrightarrow{\beta_1} \ldots \xrightarrow{\beta_n} \xrightarrow{\alpha} \longrightarrow \ldots$ with $\alpha \in ample(s_0)$, $\beta_i \notin ample(s_0)$
		$s_0 \overset{\alpha}{\Longrightarrow} \xrightarrow{\beta_1} \ldots \xrightarrow{\beta_n} \longrightarrow \ldots$
case 2:	$\rho =$	$s_0 \xrightarrow{\beta_1} \xrightarrow{\beta_2} \xrightarrow{\beta_3} \ldots$ with $\beta_i \notin ample(s_0)$

Stutter trace equivalence of $\mathcal{T}$ and $\mathcal{T}_{red}$

execution ρ in $\mathcal{T}$ $\rightsquigarrow$ execution ρ_{red} in $\mathcal{T}_{red}$ s.t. $\rho \stackrel{\Delta}{=} \rho_{red}$

case 0:	$\rho =$	$s_0 \xrightarrow{\alpha} s_0' \longrightarrow \ldots$ with $\alpha \in ample(s_0)$
		$s_0 \stackrel{\alpha}{\Rightarrow} s_0' \longrightarrow \ldots$
case 1:	$\rho =$	$s_0 \xrightarrow{\beta_1} \ldots \xrightarrow{\beta_n} \xrightarrow{\alpha} \longrightarrow \ldots$ with $\alpha \in ample(s_0)$, $\beta_i \notin ample(s_0)$
		$s_0 \stackrel{\alpha}{\Rightarrow} \xrightarrow{\beta_1} \ldots \xrightarrow{\beta_n} \longrightarrow \ldots$
case 2:	$\rho =$	$s_0 \xrightarrow{\beta_1} \xrightarrow{\beta_2} \xrightarrow{\beta_3} \ldots$ with $\beta_i \notin ample(s_0)$
		$s_0 \stackrel{\alpha}{\Rightarrow} \xrightarrow{\beta_1} \xrightarrow{\beta_2} \xrightarrow{\beta_3} \ldots$ for some $\alpha \in ample(s_0)$

Stutter trace equivalence of $\mathcal{T}$ and $\mathcal{T}_{\text{red}}$

execution ρ in $\mathcal{T}$ $\rightsquigarrow$ execution ρ_{red} in $\mathcal{T}_{\text{red}}$ s.t. $\rho \stackrel{\Delta}{=} \rho_{\text{red}}$

ρ_{red} results by an infinite sequence application of *cases 0*, *1* and *2*, i.e.,

$$\rho \rightsquigarrow \rho_1 \rightsquigarrow \rho_2 \rightsquigarrow \rho_3 \rightsquigarrow \ldots$$

Stutter trace equivalence of $\mathcal{T}$ and $\mathcal{T}_{\mathrm{red}}$

execution ρ in $\mathcal{T}$ $\rightsquigarrow$ execution ρ_{red} in $\mathcal{T}_{\mathrm{red}}$ s.t. $\rho \triangleq \rho_{\mathrm{red}}$

ρ_{red} results by an infinite sequence application of *cases 0*, *1* and *2*, i.e.,

$$\rho \rightsquigarrow \rho_1 \rightsquigarrow \rho_2 \rightsquigarrow \rho_3 \rightsquigarrow \ldots$$

where for $i < j$ the executions ρ_j and ρ_i have a common prefix of length i which is a path fragment in $\mathcal{T}_{\mathrm{red}}$

Stutter trace equivalence of $\mathcal{T}$ and $\mathcal{T}_{red}$

execution ρ in $\mathcal{T}$ $\rightsquigarrow$ execution ρ_{red} in $\mathcal{T}_{red}$ s.t. $\rho \overset{\Delta}{=} \rho_{red}$

ρ_{red} results by an infinite sequence application of *cases 0*, *1* and *2*, i.e.,

$$\rho \rightsquigarrow \rho_1 \rightsquigarrow \rho_2 \rightsquigarrow \rho_3 \rightsquigarrow \ldots$$

where for $i < j$ the executions ρ_j and ρ_i have a common prefix of length i which is a path fragment in $\mathcal{T}_{red}$, i.e., ρ_i has the form

$$\rho_i = \underbrace{s_0 \Rightarrow s_1 \Rightarrow \ldots \Rightarrow s_i}_{\text{in } \mathcal{T}_{red}} \underbrace{\rightarrow s_{i+1} \rightarrow s_{i+2} \rightarrow \ldots}_{\text{in } \mathcal{T}}$$

Stutter trace equivalence of $\mathcal{T}$ and $\mathcal{T}_{red}$

execution ρ in $\mathcal{T}$ $\rightsquigarrow$ execution ρ_{red} in $\mathcal{T}_{red}$ s.t. $\rho \triangleq \rho_{red}$

ρ_{red} results by an infinite sequence application of *cases 0*, *1* and *2*, i.e.,

$$\rho \rightsquigarrow \rho_1 \rightsquigarrow \rho_2 \rightsquigarrow \rho_3 \rightsquigarrow \ldots$$

where

$$\rho_i = s_0 \Rightarrow s_1 \Rightarrow \ldots \Rightarrow s_i \longrightarrow s_{i+1} \longrightarrow s_{i+2} \longrightarrow s_{i+3} \longrightarrow \ldots$$

$$\rho_{i+1} = s_0 \Rightarrow s_1 \Rightarrow \ldots \Rightarrow s_i \Rightarrow s_{i+1} \longrightarrow s_{i+2} \longrightarrow s_{i+3} \longrightarrow \ldots$$

$$\rho_{i+2} = s_0 \Rightarrow s_1 \Rightarrow \ldots \Rightarrow s_i \Rightarrow s_{i+1} \Rightarrow s_{i+2} \longrightarrow s_{i+3} \longrightarrow \ldots$$

Transformation $\rho \rightsquigarrow \rho_1$

case 0: $\rho = \quad s_0 \xrightarrow{\alpha} s_0' \longrightarrow \ldots$ with $\alpha \in ample(s_0)$

---

case 1: $\rho = \quad s_0 \xrightarrow{\beta_1} \ldots \xrightarrow{\beta_n} \xrightarrow{\alpha} \longrightarrow \ldots$ with $\alpha \in ample(s_0)$

$\beta_i \notin ample(s_0)$

---

case 2: $\rho = \quad s_0 \xrightarrow{\beta_1} \xrightarrow{\beta_2} \xrightarrow{\beta_3} \ldots$ with $\beta_i \notin ample(s_0)$

Transformation $\rho \rightsquigarrow \rho_1$

case 0: $\rho = \quad s_0 \xrightarrow{\alpha} s_0' \longrightarrow \ldots$ with $\alpha \in \text{ample}(s_0)$

$\rho_1 = \quad s_0 \overset{\alpha}{\Longrightarrow} s_0' \longrightarrow \ldots$

---

case 1: $\rho = \quad s_0 \xrightarrow{\beta_1} \ldots \xrightarrow{\beta_n} \xrightarrow{\alpha} \longrightarrow \ldots$ with $\alpha \in \text{ample}(s_0)$

$\beta_i \notin \text{ample}(s_0)$

---

case 2: $\rho = \quad s_0 \xrightarrow{\beta_1} \xrightarrow{\beta_2} \xrightarrow{\beta_3} \ldots$ with $\beta_i \notin \text{ample}(s_0)$

Transformation $\rho \rightsquigarrow \rho_1$

case 0: $\rho = \quad s_0 \xrightarrow{\alpha} s_0' \longrightarrow \ldots$ with $\alpha \in \mathit{ample}(s_0)$

$\rho_1 = \quad s_0 \xRightarrow{\alpha} s_0' \longrightarrow \ldots$

---

case 1: $\rho = \quad s_0 \xrightarrow{\beta_1} \ldots \xrightarrow{\beta_n} \xrightarrow{\alpha} \longrightarrow \ldots$ with $\alpha \in \mathit{ample}(s_0)$

$\beta_i \notin \mathit{ample}(s_0)$

$\rho_1 = \quad s_0 \xRightarrow{\alpha} s_0' \xrightarrow{\beta_1} \ldots \xrightarrow{\beta_n} \longrightarrow \ldots$

---

case 2: $\rho = \quad s_0 \xrightarrow{\beta_1} \xrightarrow{\beta_2} \xrightarrow{\beta_3} \ldots$ with $\beta_i \notin \mathit{ample}(s_0)$

Transformation $\rho \rightsquigarrow \rho_1$

case 0: $\rho = \quad s_0 \xrightarrow{\alpha} s_0' \longrightarrow \ldots$ with $\alpha \in \text{ample}(s_0)$

$\rho_1 = \quad s_0 \overset{\alpha}{\Longrightarrow} s_0' \longrightarrow \ldots$

---

case 1: $\rho = \quad s_0 \xrightarrow{\beta_1} \ldots \xrightarrow{\beta_n} \xrightarrow{\alpha} \longrightarrow \ldots$ with $\alpha \in \text{ample}(s_0)$

$\beta_i \notin \text{ample}(s_0)$

$\rho_1 = \quad s_0 \overset{\alpha}{\Longrightarrow} s_0' \xrightarrow{\beta_1} \ldots \xrightarrow{\beta_n} \longrightarrow \ldots$

---

case 2: $\rho = \quad s_0 \xrightarrow{\beta_1} \xrightarrow{\beta_2} \xrightarrow{\beta_3} \ldots$ with $\beta_i \notin \text{ample}(s_0)$

$\rho_1 = \quad s_0 \overset{\alpha}{\Longrightarrow} s_0' \xrightarrow{\beta_1} \xrightarrow{\beta_2} \xrightarrow{\beta_3} \ldots$ for some $\alpha \in \text{ample}(s_0)$

Transformation $\rho \rightsquigarrow \rho_1$

case 0: $\rho = \quad s_0 \xrightarrow{\alpha} s_0' \longrightarrow \ldots$ with $\alpha \in ample(s_0)$

$\rho_1 = \quad s_0 \overset{\alpha}{\Longrightarrow} s_0' \longrightarrow \ldots$

---

case 1: $\rho = \quad s_0 \xrightarrow{\beta_1} \ldots \xrightarrow{\beta_n} \xrightarrow{\alpha} \longrightarrow \ldots$ with $\alpha \in ample(s_0)$

$\beta_i \notin ample(s_0)$

$\rho_1 = \quad s_0 \overset{\alpha}{\Longrightarrow} s_0' \xrightarrow{\beta_1} \ldots \xrightarrow{\beta_n} \longrightarrow \ldots$

---

case 2: $\rho = \quad s_0 \xrightarrow{\beta_1} \xrightarrow{\beta_2} \xrightarrow{\beta_3} \ldots$ with $\beta_i \notin ample(s_0)$

$\rho_1 = \quad s_0 \overset{\alpha}{\Longrightarrow} s_0' \xrightarrow{\beta_1} \xrightarrow{\beta_2} \xrightarrow{\beta_3} \ldots$ for some $\alpha \in ample(s_0)$

for the transformation $\rho_1 \rightsquigarrow \rho_2$:

apply case 0,1 or 2 to the suffix starting in state s_0'

Conditions for ample sets

(A1) nonemptiness condition

$$\emptyset \neq \mathsf{ample}(\mathsf{s}) \subseteq Act(\mathsf{s})$$

Conditions for ample sets

(A1) nonemptiness condition

$$\emptyset \neq \mathsf{ample}(s) \subseteq Act(s)$$

(A2) dependency condition

Conditions for ample sets

(A1) nonemptiness condition

$$\emptyset \neq \text{ample}(s) \subseteq Act(s)$$

(A2) dependency condition

for each execution fragment in $\mathcal{T}$

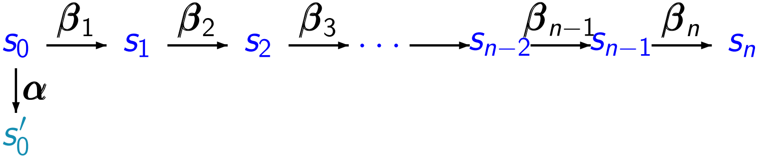
$$s \xrightarrow{\beta_1} \xrightarrow{\beta_2} \ldots \xrightarrow{\beta_{i-1}} \xrightarrow{\beta_i} \xrightarrow{\beta_{i+1}} \ldots \xrightarrow{\beta_{n-1}} \xrightarrow{\beta_n}$$

such that β_n is dependent from ample(s)
there is some $i < n$ with

$$\beta_i \in \text{ample}(s)$$

Condition (A2)

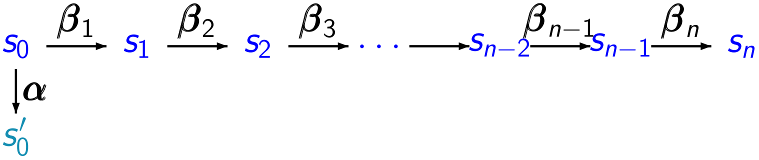
suppose $\alpha \in ample(s_0)$, $\beta_i \notin ample(s_0)$



Condition (A2)

suppose $\alpha \in$ ample(s_0), $\beta_i \notin$ ample(s_0)

$\stackrel{(A2)}{\Longrightarrow}$ α, β_i independent

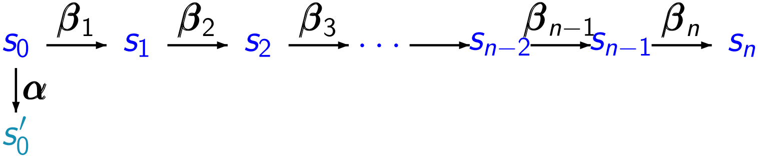


Condition (A2)

suppose $\alpha \in ample(s_0)$, $\beta_i \notin ample(s_0)$

$\overset{(A2)}{\Longrightarrow}$ α, β_i independent

$\Longrightarrow$ $\alpha \in Act(s_i)$ for $i = 0, 1, 2, \ldots$

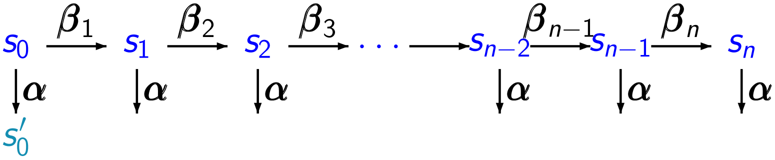


Condition (A2)

suppose $\alpha \in \text{ample}(s_0)$, $\beta_i \notin \text{ample}(s_0)$

$\overset{\text{(A2)}}{\Longrightarrow}$ α, β_i independent

$\Longrightarrow$ $\alpha \in Act(s_i)$ for $i = 0, 1, 2, \ldots$

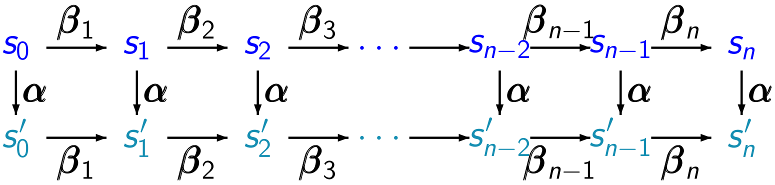


Condition (A2)

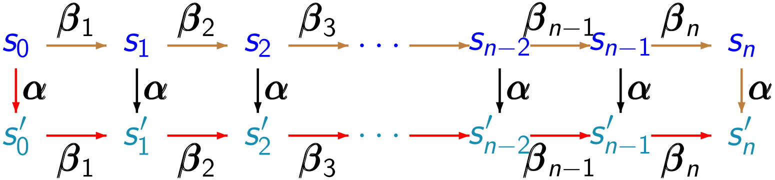
suppose $\alpha \in ample(s_0)$, $\beta_i \notin ample(s_0)$

$\overset{(A2)}{\Longrightarrow}$ α, β_i independent

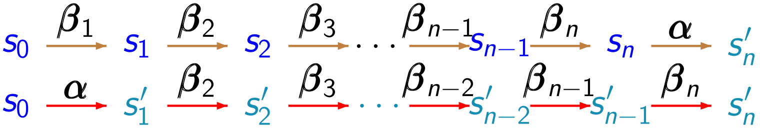
$\Longrightarrow$ $\alpha \in Act(s_i)$ for $i = 0, 1, 2, \ldots$

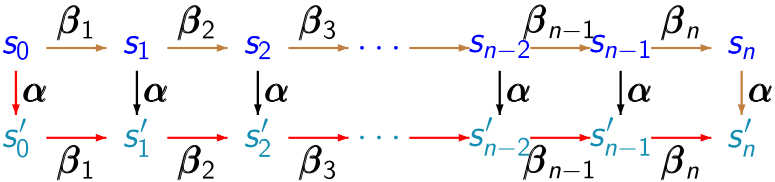


$$
\begin{array}{ccccccccccccc}
s_0 & \xrightarrow{\beta_1} & s_1 & \xrightarrow{\beta_2} & s_2 & \xrightarrow{\beta_3} & \cdots & \longrightarrow & s_{n-2} & \xrightarrow{\beta_{n-1}} & s_{n-1} & \xrightarrow{\beta_n} & s_n \\
\downarrow\alpha & & \downarrow\alpha & & \downarrow\alpha & & & & \downarrow\alpha & & \downarrow\alpha & & \downarrow\alpha \\
s'_0 & \xrightarrow[\beta_1]{} & s'_1 & \xrightarrow[\beta_2]{} & s'_2 & \xrightarrow[\beta_3]{} & \cdots & \longrightarrow & s'_{n-2} & \xrightarrow[\beta_{n-1}]{} & s'_{n-1} & \xrightarrow[\beta_n]{} & s'_n
\end{array}
$$

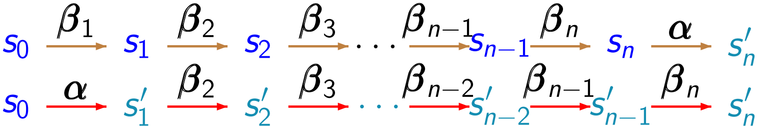


case 1:

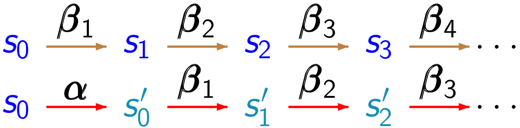



s0 β1 s1 β2 s2 β3 . . . sn−2 βn−1 sn−1 βn sn
α α α α α α
s′0 β1 s′1 β2 s′2 β3 . . . s′n−2 βn−1 s′n−1 βn s′n

case 1:


s0 β1 s1 β2 s2 β3 . . . βn−1 sn−1 βn sn α s′n
s0 α s′1 β2 s′2 β3 . . . βn−2 s′n−2 βn−1 s′n−1 βn s′n

case 2:


s0 β1 s1 β2 s2 β3 s3 β4 . . .
s0 α s′0 β1 s′1 β2 s′2 β3 . . .

Conditions for ample sets

(A1) nonemptiness condition

$$\emptyset \neq \mathsf{ample}(s) \subseteq Act(s)$$

(A2) dependency condition

for each execution fragment in $\mathcal{T}$

$$s \xrightarrow{\beta_1} \xrightarrow{\beta_2} \ldots \xrightarrow{\beta_{i-1}} \xrightarrow{\beta_i} \xrightarrow{\beta_{i+1}} \ldots \xrightarrow{\beta_{n-1}} \xrightarrow{\beta_n}$$

such that β_n is *dependent* from $\mathsf{ample}(s)$ there is some $i < n$ with $\beta_i \in \mathsf{ample}(s)$

Conditions for ample sets

(A1) nonemptiness condition

$\emptyset \neq \mathsf{ample}(s) \subseteq Act(s)$

(A2) dependency condition

for each execution fragment in $\mathcal{T}$

$$s \xrightarrow{\beta_1} \xrightarrow{\beta_2} \ldots \xrightarrow{\beta_{i-1}} \xrightarrow{\beta_i} \xrightarrow{\beta_{i+1}} \ldots \xrightarrow{\beta_{n-1}} \xrightarrow{\beta_n}$$

such that β_n is *dependent* from $\mathsf{ample}(s)$ there is some $i < n$ with $\beta_i \in \mathsf{ample}(s)$

(A3) stutter condition

(A1) nonemptiness condition

$\emptyset \neq \text{ample}(s) \subseteq Act(s)$

(A2) dependency condition

for each execution fragment in $\mathcal{T}$

$$s \xrightarrow{\beta_1} \xrightarrow{\beta_2} \ldots \xrightarrow{\beta_{i-1}} \xrightarrow{\beta_i} \xrightarrow{\beta_{i+1}} \ldots \xrightarrow{\beta_{n-1}} \xrightarrow{\beta_n}$$

such that β_n is *dependent* from ample(s) there is some i < n with $\beta_i \in \text{ample}(s)$

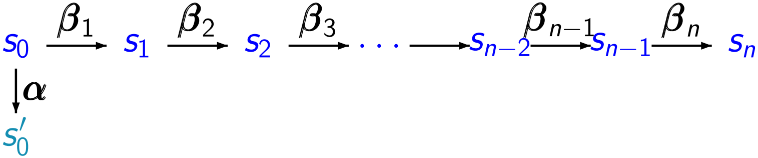
(A3) stutter condition

if $\text{ample}(s) \neq Act(s)$ then all actions in ample(s) are stutter actions

Conditions (A2) and (A3)

Suppose

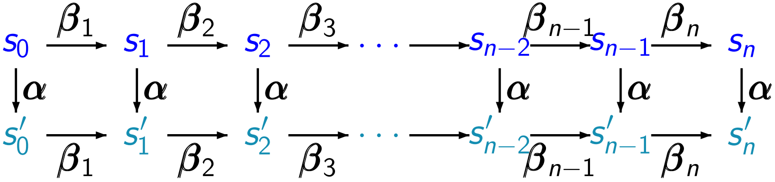
- $\alpha \in \text{ample}(s_0)$, $\beta_i \notin \text{ample}(s_0)$
- α stutter action



Conditions (A2) and (A3)

Suppose

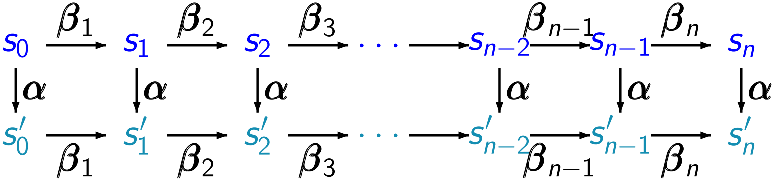
- $\alpha \in ample(s_0)$, $\beta_i \notin ample(s_0)$
- α stutter action



Conditions (A2) and (A3)

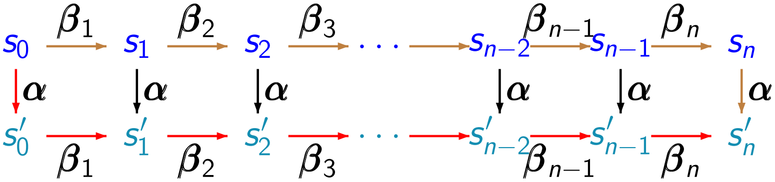
Suppose

- $\alpha \in \text{ample}(s_0)$, $\beta_i \notin \text{ample}(s_0)$
- α stutter action $\Rightarrow \text{L}(s_i) = \text{L}(s_i')$, $i = 0, 1, 2, \ldots$

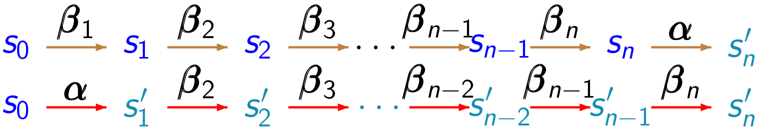


Conditions (A2) and (A3)

- $\alpha \in ample(s_0)$, $\beta_i \notin ample(s_0)$
- α stutter action $\Rightarrow L(s_i) = L(s_i')$, $i = 0, 1, 2, \ldots$

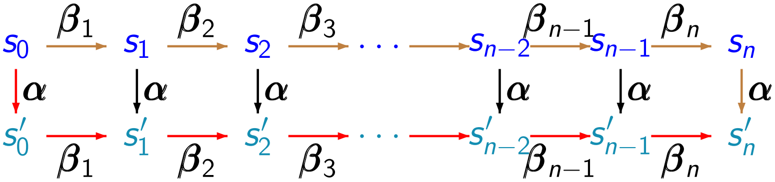


case 1:

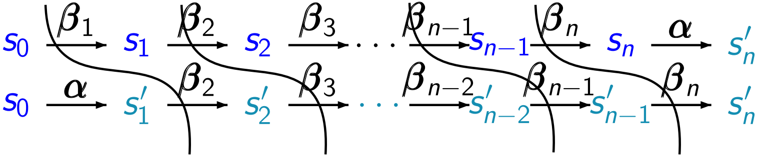


Conditions (A2) and (A3)

- $\alpha \in$ ample(s_0), $\beta_i \notin$ ample(s_0)
- α stutter action $\Rightarrow$ $L(s_i) = L(s_i')$, $i = 0, 1, 2, \ldots$

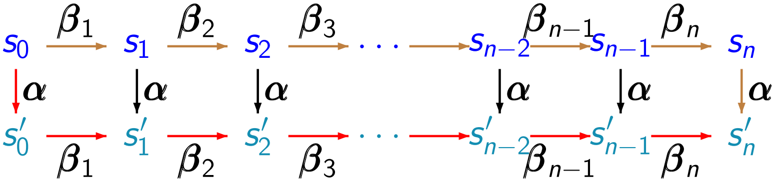


case 1:

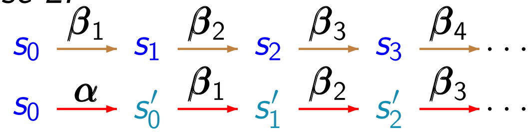


Conditions (A2) and (A3)

- $\alpha \in ample(s_0)$, $\beta_i \notin ample(s_0)$
- α stutter action $\Rightarrow L(s_i) = L(s_i')$, $i = 0, 1, 2, \ldots$

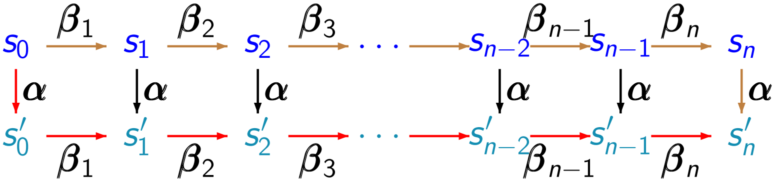


case 2:

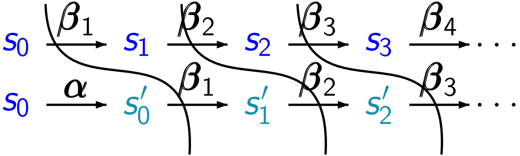


Conditions (A2) and (A3)

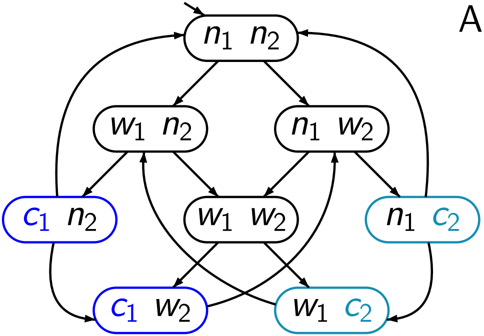
- $\alpha \in \text{ample}(s_0)$, $\beta_i \notin \text{ample}(s_0)$
- α stutter action $\Rightarrow$ $L(s_i) = L(s_i')$, $i = 0, 1, 2, \ldots$



case 2:

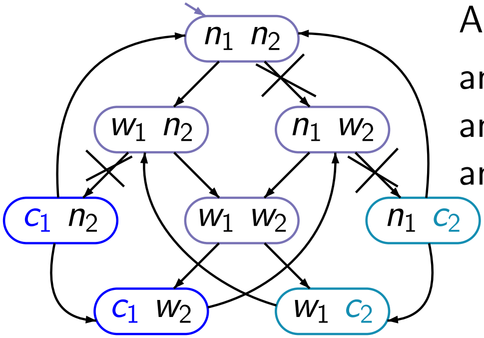


Ample sets for MUTEX



$\mathrm{AP} = \{c_1, c_2\}$

Ample sets for MUTEX



$\mathrm{AP} = \{c_1, c_2\}$

$\mathrm{ample}(n_1, n_2) = \{\mathrm{request}_1\}$

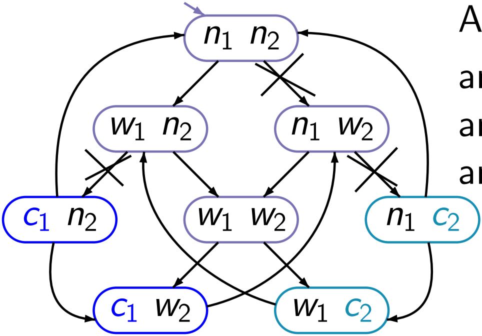
$\mathrm{ample}(w_1, n_2) = \{\mathrm{request}_2\}$

$\mathrm{ample}(w_1, w_2) =$

$\{\mathrm{enter}_1, \mathrm{enter}_2\}$

$\ldots$

Ample sets for MUTEX



$\text{AP} = \{c_1, c_2\}$

$\text{ample}(n_1, n_2) = \{\text{request}_1\}$

$\text{ample}(w_1, n_2) = \{\text{request}_2\}$

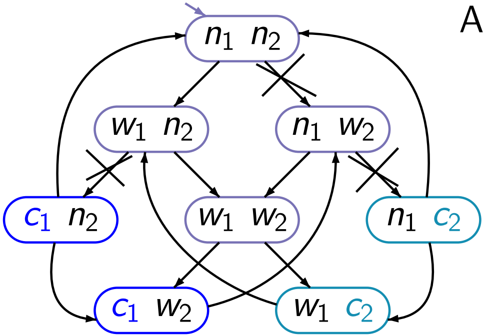
$\text{ample}(w_1, w_2) =$

$\{\text{enter}_1, \text{enter}_2\}$

$\cdots$

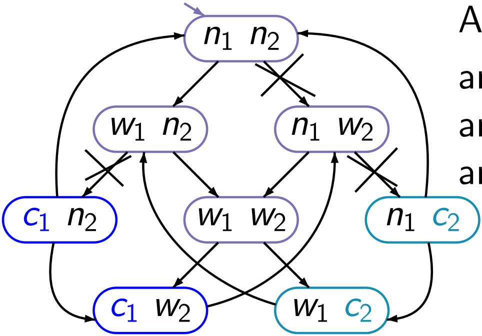
(A1), (A2), (A3) are satisfied

Ample sets for MUTEX



$AP = \{c_1, c_2\}$

Ample sets for MUTEX



$AP = \{c_1, c_2\}$

$\text{ample}(n_1, n_2) = \{\text{request}_1\}$

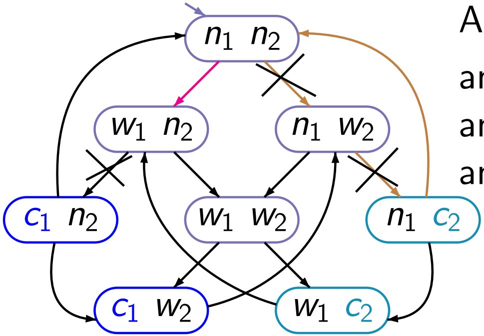
$\text{ample}(w_1, n_2) = \{\text{request}_2\}$

$\text{ample}(w_1, w_2) =$

$\{\text{enter}_1, \text{enter}_2\}$

$\ldots$

Ample sets for MUTEX



$\mathrm{AP} = \{c_1, c_2\}$

$\mathrm{ample}(n_1, n_2) = \{\mathrm{request}_1\}$

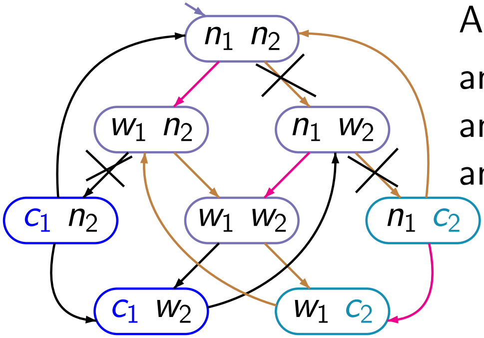
$\mathrm{ample}(w_1, n_2) = \{\mathrm{request}_2\}$

$\mathrm{ample}(w_1, w_2) =$
$\{\mathrm{enter}_1, \mathrm{enter}_2\}$

$\ldots$

$$n_1 n_2 \xrightarrow{\mathrm{request}_2} n_1 w_2 \xrightarrow{\mathrm{enter}_2} n_1 c_2 \xrightarrow{\mathrm{release}_2} n_1 n_2 \xrightarrow{\mathrm{request}_1} w_1 n_2$$

Ample sets for MUTEX



$AP = \{c_1, c_2\}$

$\text{ample}(n_1, n_2) = \{\text{request}_1\}$

$\text{ample}(w_1, n_2) = \{\text{request}_2\}$

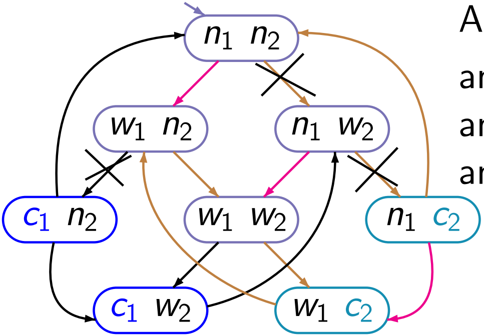
$\text{ample}(w_1, w_2) =$
$\{\text{enter}_1, \text{enter}_2\}$

$\ldots$

$$n_1 n_2 \xrightarrow{\text{request}_2} n_1 w_2 \xrightarrow{\text{enter}_2} n_1 c_2 \xrightarrow{\text{release}_2} n_1 n_2 \xrightarrow{\text{request}_1} w_1 n_2$$

$$n_1 n_2 \xrightarrow{\text{request}_2} n_1 w_2 \xrightarrow{\text{enter}_2} n_1 c_2 \xrightarrow{\text{request}_1} w_1 c_2 \xrightarrow{\text{release}_2} w_1 n_2$$

Ample sets for MUTEX



$AP = \{c_1, c_2\}$

$\text{ample}(n_1, n_2) = \{\text{request}_1\}$

$\text{ample}(w_1, n_2) = \{\text{request}_2\}$

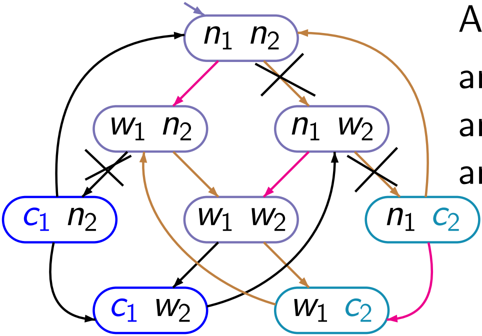
$\text{ample}(w_1, w_2) =$
$\{\text{enter}_1, \text{enter}_2\}$

$\ldots$

$n_1 n_2 \xrightarrow{\text{request}_2} n_1 w_2 \xrightarrow{\text{enter}_2} n_1 c_2 \xrightarrow{\text{release}_2} n_1 n_2 \xrightarrow{\text{request}_1} w_1 n_2$

$n_1 n_2 \xrightarrow{\text{request}_2} n_1 w_2 \xrightarrow{\text{enter}_2} n_1 c_2 \xrightarrow{\text{request}_1} w_1 c_2 \xrightarrow{\text{release}_2} w_1 n_2$

$n_1 n_2 \xrightarrow{\text{request}_2} n_1 w_2 \xrightarrow{\text{request}_1} w_1 w_2 \xrightarrow{\text{enter}_2} w_1 c_2 \xrightarrow{\text{release}_2} w_1 n_2$



$AP = \{c_1, c_2\}$

$\text{ample}(n_1, n_2) = \{\text{request}_1\}$

$\text{ample}(w_1, n_2) = \{\text{request}_2\}$

$\text{ample}(w_1, w_2) =$

$\{\text{enter}_1, \text{enter}_2\}$

...

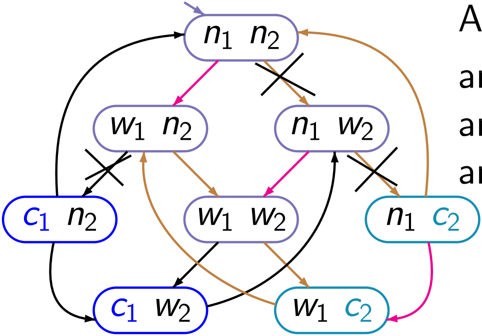
$n_1 n_2 \xrightarrow{\text{request}_2} n_1 w_2 \xrightarrow{\text{enter}_2} n_1 c_2 \xrightarrow{\text{release}_2} n_1 n_2 \xrightarrow{\text{request}_1} w_1 n_2$

$n_1 n_2 \xrightarrow{\text{request}_2} n_1 w_2 \xrightarrow{\text{enter}_2} n_1 c_2 \xrightarrow{\text{request}_1} w_1 c_2 \xrightarrow{\text{release}_2} w_1 n_2$

$n_1 n_2 \xrightarrow{\text{request}_2} n_1 w_2 \xrightarrow{\text{request}_1} w_1 w_2 \xrightarrow{\text{enter}_2} w_1 c_2 \xrightarrow{\text{release}_2} w_1 n_2$

$n_1 n_2 \xrightarrow{\text{request}_1} w_1 n_2 \xrightarrow{\text{request}_2} w_1 w_2 \xrightarrow{\text{enter}_2} w_1 c_2 \xrightarrow{\text{release}_2} w_1 n_2$

Ample sets for MUTEX



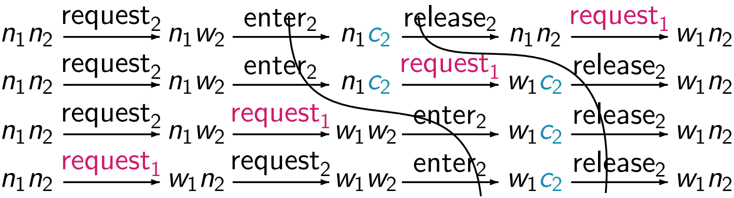
$AP = \{c_1, c_2\}$

$ample(n_1, n_2) = \{request_1\}$

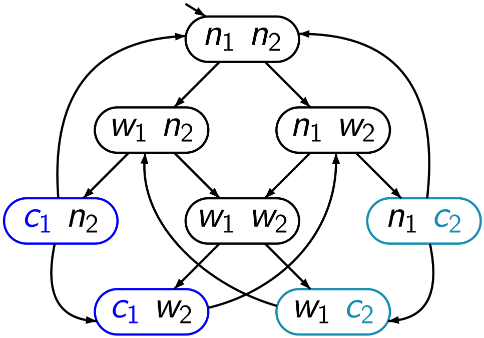
$ample(w_1, n_2) = \{request_2\}$

$ample(w_1, w_2) =$
$\{enter_1, enter_2\}$

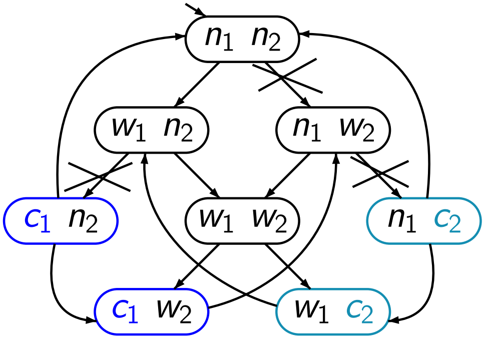
$\ldots$



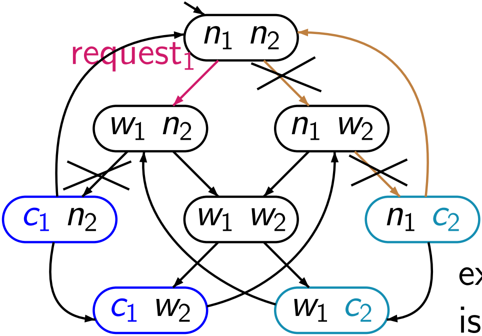
Example: case 2



Example: case 2



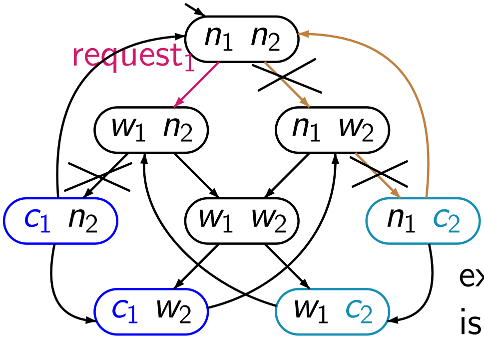
Example: case 2



execution where request$_1$
is not executed

$$n_1 n_2 \xrightarrow{\text{requ}_2} n_1 w_2 \xrightarrow{\text{enter}_2} n_1 c_2 \xrightarrow{\text{release}_2} n_1 n_2 \xrightarrow{\text{requ}_2} \ldots$$

Example: case 2

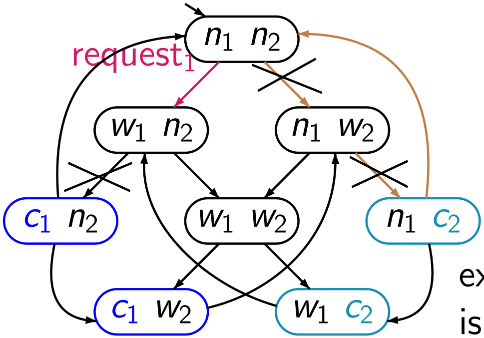


execution where request$_1$ is not executed

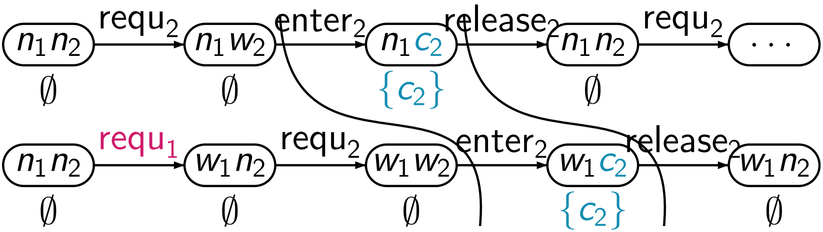
$$n_1 n_2 \xrightarrow{\text{requ}_2} n_1 w_2 \xrightarrow{\text{enter}_2} n_1 c_2 \xrightarrow{\text{release}_2} n_1 n_2 \xrightarrow{\text{requ}_2} \ldots$$

$$n_1 n_2 \xrightarrow{\text{requ}_1} w_1 n_2 \xrightarrow{\text{requ}_2} w_1 w_2 \xrightarrow{\text{enter}_2} w_1 c_2 \xrightarrow{\text{release}_2} w_1 n_2$$

Example: case 2



execution where request$_1$ is not executed

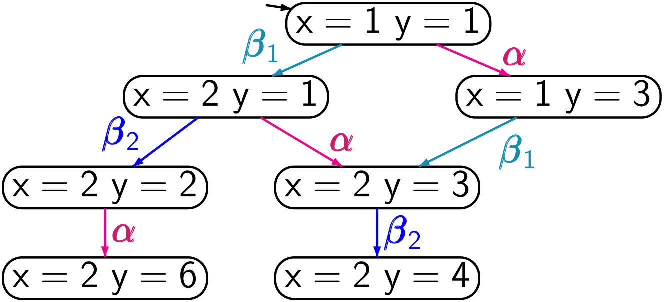


Example

$$\underbrace{\mathsf{x} := 2 \cdot \mathsf{x}}_{\beta_1} \;\; ; \;\; \underbrace{\mathsf{y} := \mathsf{y} + 1}_{\beta_2} \;\; ||| \;\; \underbrace{\mathsf{y} := 3 \cdot \mathsf{y}}_{\alpha}$$

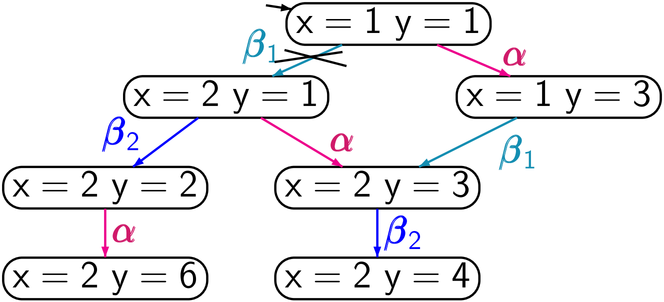
Example

$$\underbrace{x := 2 \cdot x}_{\beta_1} \quad ; \quad \underbrace{y := y + 1}_{\beta_2} \quad ||| \quad \underbrace{y := 3 \cdot y}_{\alpha}$$



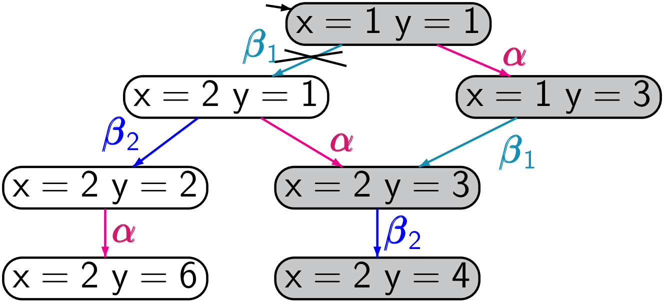
Example

$$\underbrace{x := 2 \cdot x}_{\beta_1} \quad ; \quad \underbrace{y := y + 1}_{\beta_2} \quad ||| \quad \underbrace{y := 3 \cdot y}_{\alpha}$$



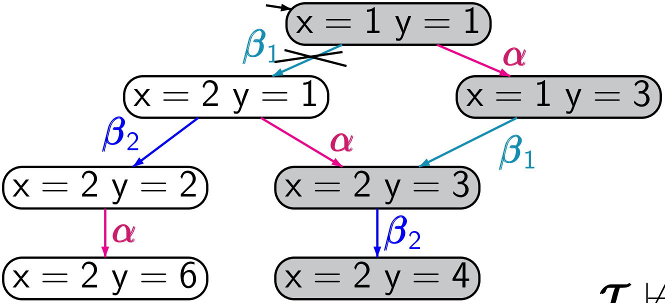
Example

$$\underbrace{x := 2 \cdot x}_{\beta_1} \;\; ; \;\; \underbrace{y := y + 1}_{\beta_2} \;\; ||| \;\; \underbrace{y := 3 \cdot y}_{\alpha}$$



Example

$$\underbrace{x := 2 \cdot x}_{\beta_1} \;\; ; \;\; \underbrace{y := y + 1}_{\beta_2} \;\; ||| \;\; \underbrace{y := 3 \cdot y}_{\alpha}$$

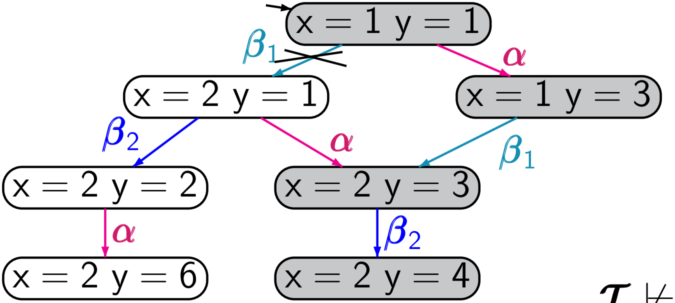


$$\mathcal{T} \not\models \Box(y \neq 6)$$

$$\mathcal{T}_{red} \models \Box(y \neq 6)$$

Example

$$\underbrace{x := 2 \cdot x}_{\beta_1} \;\; ; \;\; \underbrace{y := y + 1}_{\beta_2} \;\; ||| \;\; \underbrace{y := 3 \cdot y}_{\alpha}$$

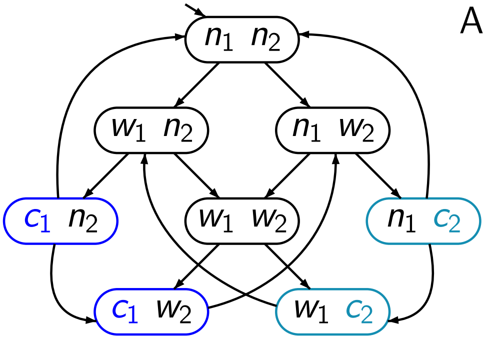


$\mathcal{T} \not\models \Box(y \neq 6)$

$\mathcal{T}_{\text{red}} \models \Box(y \neq 6)$

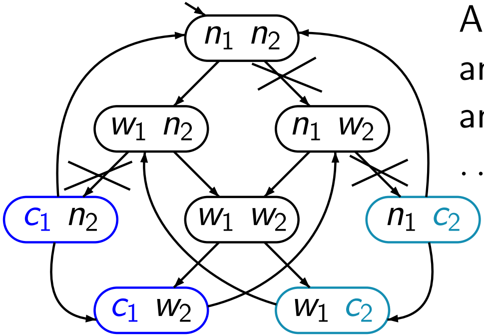
(A2) violated as β_2, α dependent

Which conditions (A1), (A2) or (A3) are satisfied?



$\text{AP} = \{c_1, c_2\}$

Which conditions (A1), (A2) or (A3) are satisfied?



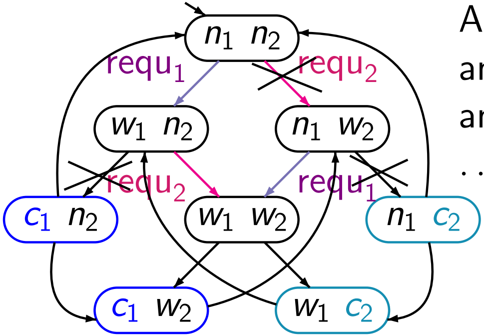
$\text{AP} = \{c_1, c_2\}$

$\text{ample}(n_1, n_2) = \{\text{request}_1\}$

$\text{ample}(w_1, n_2) = \{\text{request}_2\}$

...

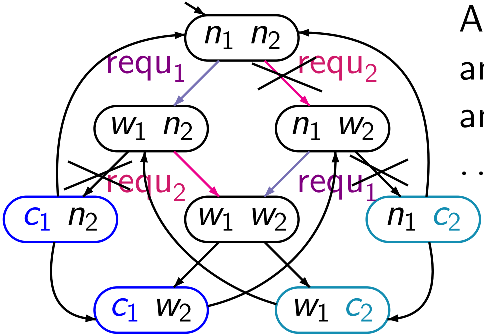
Which conditions (A1), (A2) or (A3) are satisfied?



$\mathrm{AP} = \{c_1, c_2\}$
$\mathrm{ample}(n_1, n_2) = \{\mathrm{request}_1\}$
$\mathrm{ample}(w_1, n_2) = \{\mathrm{request}_2\}$
$\ldots$

Does the nonemptiness condition (A1) hold?

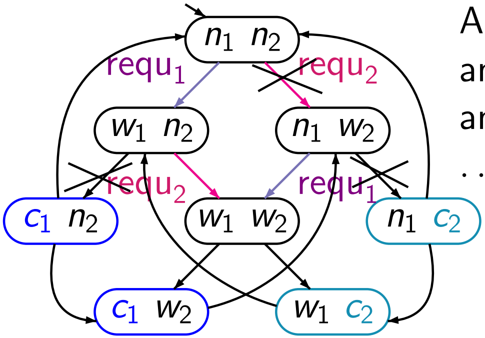
Which conditions (A1), (A2) or (A3) are satisfied?



AP $= \{c_1, c_2\}$
ample$(n_1, n_2) = \{$request$_1\}$
ample$(w_1, n_2) = \{$request$_2\}$
$\ldots$

Does the nonemptiness condition (A1) hold? **yes**

Which conditions (A1), (A2) or (A3) are satisfied?



$\mathrm{AP} = \{c_1, c_2\}$

$\mathrm{ample}(n_1, n_2) = \{\mathrm{request}_1\}$

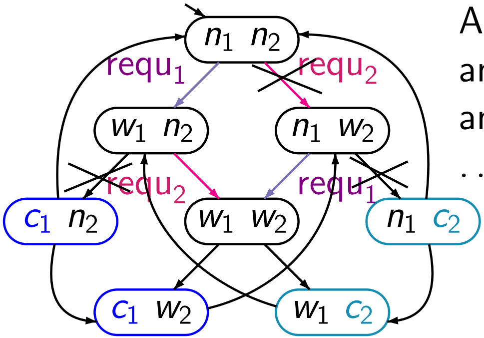
$\mathrm{ample}(w_1, n_2) = \{\mathrm{request}_2\}$

...

Does the nonemptiness condition (A1) hold? **yes**

Does the dependency condition (A2) hold?

Which conditions (A1), (A2) or (A3) are satisfied?

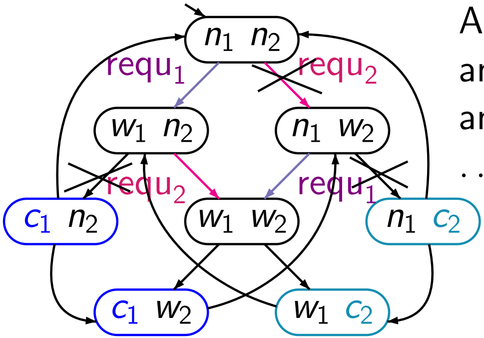


AP $= \{c_1, c_2\}$
ample$(n_1, n_2) = \{$request$_1\}$
ample$(w_1, n_2) = \{$request$_2\}$
$\ldots$

Does the nonemptiness condition (A1) hold? **yes**

Does the dependency condition (A2) hold?
yes, as request$_1$ and request$_2$ are independent from all other actions

Which conditions (A1), (A2) or (A3) are satisfied?



$AP = \{c_1, c_2\}$

$ample(n_1, n_2) = \{request_1\}$

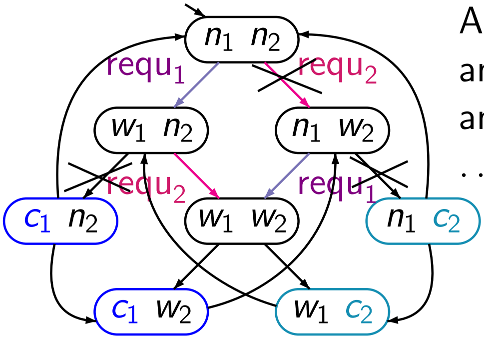
$ample(w_1, n_2) = \{request_2\}$

...

(A1), (A2): $\surd$

Does the stutter condition (A3) hold?

Which conditions (A1), (A2) or (A3) are satisfied?



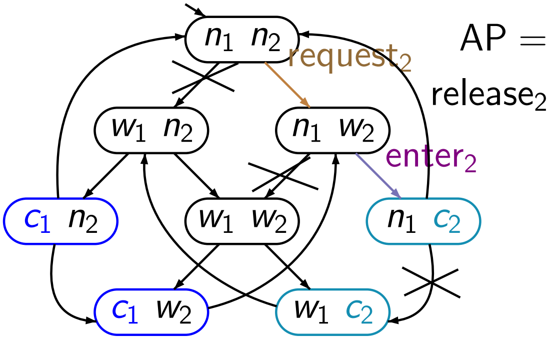
$AP = \{c_1, c_2\}$
$ample(n_1, n_2) = \{request_1\}$
$ample(w_1, n_2) = \{request_2\}$
$\ldots$

(A1), (A2): $\surd$

Does the stutter condition (A3) hold?

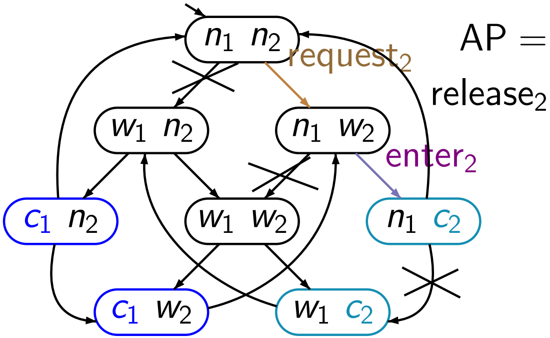
yes, as $request_1$ and $request_2$ are stutter actions

Which conditions (A1), (A2) or (A3) are satisfied?



$\text{AP} = \{c_1, c_2\}$

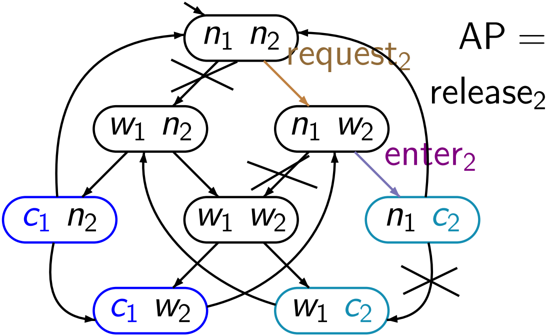
Which conditions (A1), (A2) or (A3) are satisfied?



$AP = \{c_1, c_2\}$

(A1): $\surd$

Which conditions (A1), (A2) or (A3) are satisfied?

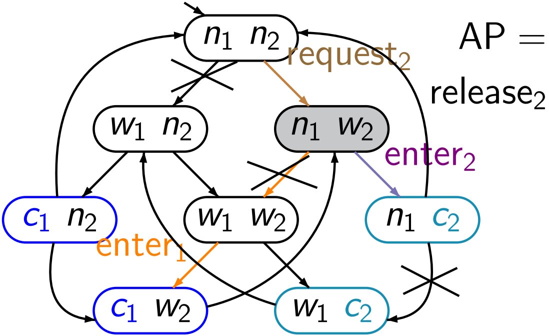


$AP = \{c_1, c_2\}$

(A1): $\surd$

(A2)

Which conditions (A1), (A2) or (A3) are satisfied?

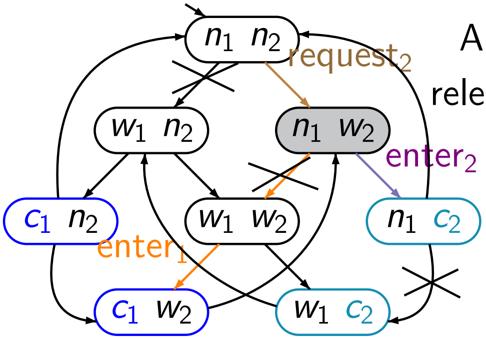


$AP = \{c_1, c_2\}$

(A1): $\surd$

(A2) violated as enter$_1$ is dependent from enter$_2$

Which conditions (A1), (A2) or (A3) are satisfied?



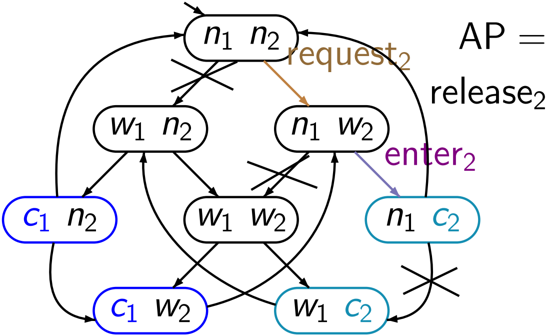
$\text{AP} = \{c_1, c_2\}$

(A1): $\surd$

(A2) violated as enter$_1$ is dependent from enter$_2$

(A3)

Which conditions (A1), (A2) or (A3) are satisfied?



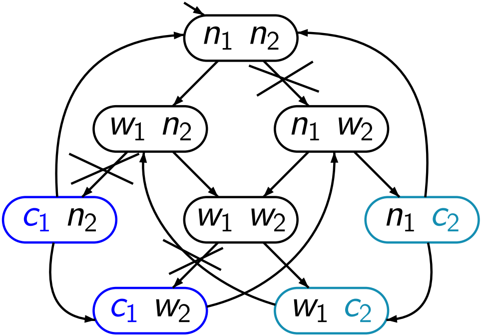
$AP = \{c_1, c_2\}$

(A1): $\surd$

(A2) violated as enter$_1$ is dependent from enter$_2$

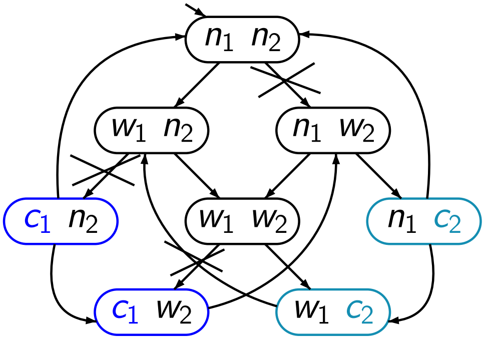
(A3) violated as ample$(n_1 w_2) = \{$enter$_2\} \subsetneq Act(n_1 w_2)$
and enter$_2$ is a visible action

Which conditions (A2) or (A3) are satisfied?



$\mathsf{AP} = \{c_1, c_2\}$

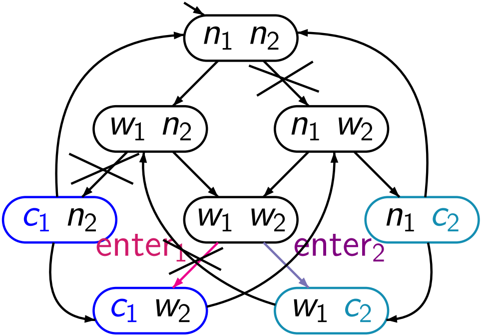
Which conditions (A2) or (A3) are satisfied?



$\mathsf{AP} = \{c_1, c_2\}$

(A2) does not hold

Which conditions (A2) or (A3) are satisfied?

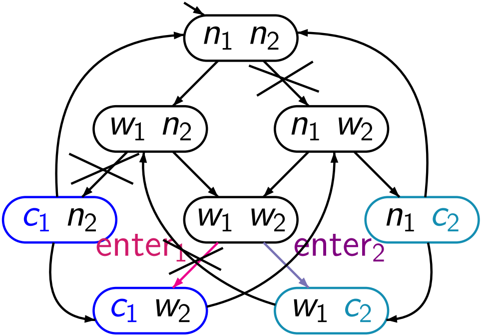


$\mathsf{AP} = \{c_1, c_2\}$

(A2) does not hold

as enter$_1$ and enter$_2$ are dependent and ample$(w_1 w_2) = \{\text{enter}_2\}$

Which conditions (A2) or (A3) are satisfied?

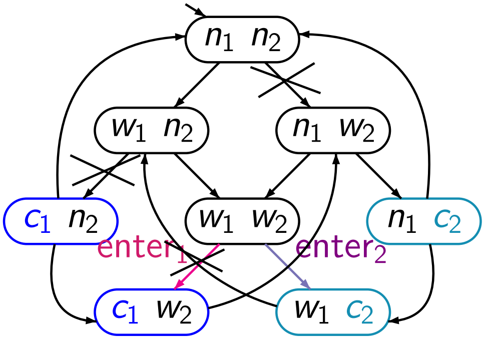


$\text{AP} = \{c_1, c_2\}$

neither (A2) nor (A3) is fulfilled

- $w_1 w_2 \xrightarrow{\text{enter}_1} \ldots$ $\rightsquigarrow$ (A2) violated

Which conditions (A2) or (A3) are satisfied?

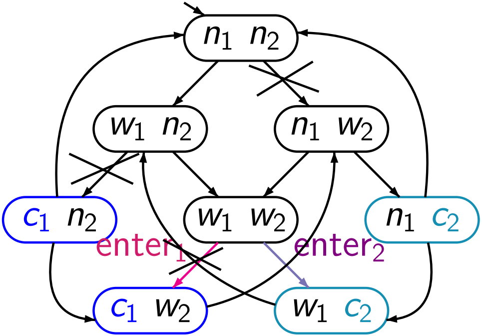


$AP = \{c_1, c_2\}$

neither (A2) nor (A3) is fulfilled

- $w_1 w_2 \xrightarrow{enter_1} \ldots$ $\rightsquigarrow$ (A2) violated
- $enter_2$ visible action $\rightsquigarrow$ (A3) violated

Which conditions (A2) or (A3) are satisfied?



$AP = \{c_1, c_2\}$

$\mathcal{T} \not\models \Box\neg c_1$

$\mathcal{T}_{red} \models \Box\neg c_1$

neither (A2) nor (A3) is fulfilled

- $w_1 w_2 \xrightarrow{\text{enter}_1} \ldots$ $\rightsquigarrow$ (A2) violated
- enter$_2$ visible action $\rightsquigarrow$ (A3) violated

Correct or wrong?

Let $\mathcal{T}_1, \mathcal{T}_2$ be TS with disjoint action-sets Act_1 and Act_2, respectively, and

$$\mathcal{T} = \mathcal{T}_1 \,|||\, \mathcal{T}_2$$

Correct or wrong?

Let $\mathcal{T}_1, \mathcal{T}_2$ be TS with disjoint action-sets Act_1 and Act_2, respectively, and

$$\mathcal{T} = \mathcal{T}_1 \,|||\, \mathcal{T}_2$$

remind: $|||$ denotes full interleaving

Correct or wrong?

Let $\mathcal{T}_1, \mathcal{T}_2$ be TS with disjoint action-sets Act_1 and Act_2, respectively, and

$$\mathcal{T} = \mathcal{T}_1 \,|||\, \mathcal{T}_2$$

Then, the ample sets given by

$$\mathit{ample}(\langle s_1, s_2 \rangle) = \begin{cases} Act_1(s_1) : & \text{if every act. in } Act_1(s_1) \\ & \text{is stutter a action} \\ Act_1(s_1) \cup Act_2(s_2) : & \text{else} \end{cases}$$

satisfy (A2) and (A3).

Let $\mathcal{T}_1, \mathcal{T}_2$ be TS with disjoint action-sets Act_1 and Act_2, respectively, and

$$\mathcal{T} = \mathcal{T}_1 \,|||\, \mathcal{T}_2$$

Then, the ample sets given by

$$ample(\langle s_1, s_2 \rangle) = \begin{cases} Act_1(s_1) : \text{ if every act. in } Act_1(s_1) \\ \qquad\qquad\qquad \text{is stutter a action} \\ Act_1(s_1) \cup Act_2(s_2) : \text{ else} \end{cases}$$

satisfy $(\mathrm{A2})$ and $(\mathrm{A3})$.

correct.

Let $\mathcal{T}_1, \mathcal{T}_2$ be TS with disjoint action-sets Act_1 and Act_2, respectively, and

$$\mathcal{T} = \mathcal{T}_1 \,|||\, \mathcal{T}_2$$

Then, the ample sets given by

$$\text{ample}(\langle s_1, s_2 \rangle) = \begin{cases} Act_1(s_1) : \text{ if every act. in } Act_1(s_1) \\ \qquad\qquad \text{is stutter a action} \\ Act_1(s_1) \cup Act_2(s_2) : \text{ else} \end{cases}$$

satisfy (A2) and (A3).

correct.

Note: all actions $\alpha \in Act_1$ and $\beta \in Act_2$ are independent in $\mathcal{T}$.