Model Checking

Partial-Order Reduction

[Baier & Katoen, Chapter 8]

Joost-Pieter Katoen and Tim Quatmann

Software Modeling and Verification Group

RWTH Aachen, SoSe 2022

Overview

Overview

Motivation

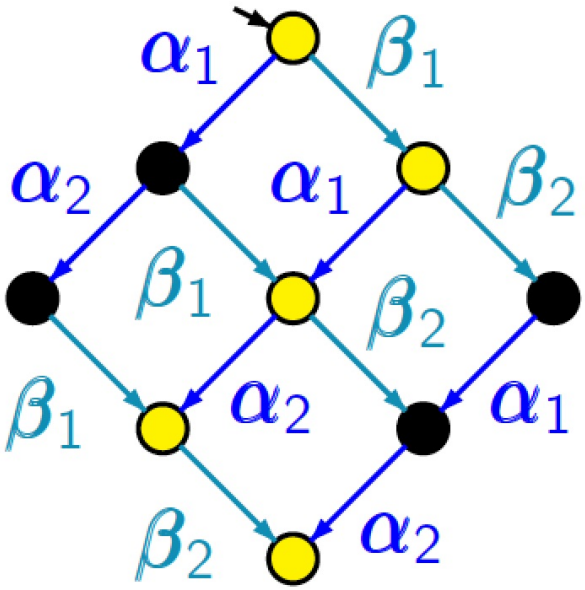
- Interleaving semantics
  - independent concurrent actions are interleaved
  - a run is defined by a totally ordered sequence of states

Motivation

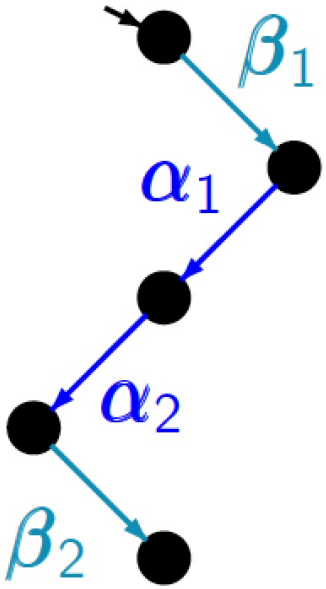
- Interleaving semantics
  - independent concurrent actions are interleaved
  - a run is defined by a totally ordered sequence of states
- Modelling concurrency by interleaving
  - may enforce an order of actions that has no real "meaning"
  - state space size = product of number of states of threads
  - this is a major cause of the state-space explosion problem
- Partial-order reduction (POR)
  - groups runs for which the order of "independent" actions is irrelevant
  - considers a single representative run for equivalent runs

Idea of Partial-Order Reduction

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



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



Inventors of Partial-Order Reduction

Patrice Godefroid (USA)

Pierre Wolper (Belgium)

Antti Valmari (Finland)

Doron Peled (Israel)

Outline of Ample-Set POR

- Given: a syntactic description of transition system TS
- Aim: On-the-fly construction of "reduced" transition system TS_{red}
  - for state s only consider outgoing actions $ample(s) \subseteq Act(s)$ where $Act(s) = \{ \alpha \in Act \mid \exists s' \in S.\, s \xrightarrow{\alpha} s' \}$, the enabled actions in s
  - expand only α-successors with $\alpha \in ample(s)$

Outline of Ample-Set POR

- Given: a syntactic description of transition system TS
- Aim: On-the-fly construction of "reduced" transition system TS_{red}
  - for state s only consider outgoing actions $ample(s) \subseteq Act(s)$ where $Act(s) = \{ \alpha \in Act \mid \exists s' \in S.\, s \xrightarrow{\alpha} s' \}$, the enabled actions in s
  - expand only α-successors with $\alpha \in ample(s)$
- Intuition: actions in $ample(s)$ already capture all relevant behavior at s
- Key issue: which actions to choose from $Act(s)$?
- Requirements:
  - such that $TS_{red} \equiv_{sttrace} TS$, hence TS_{red} and TS are $\mathrm{LTL}_{\setminus\bigcirc}$-equivalent
  - TS_{red} is (much) smaller than TS
  - TS_{red} can be obtained efficiently

Reminder: Stutter Equivalence

Definition: stutter step

Transition $s \to s'$ in transition system TS is a stutter step if $L(s) = L(s')$.

Definition: stutter equivalence

Paths π_1 and π_2 are stutter equivalent, denoted $\pi_1 \equiv_{sttrace} \pi_2$ whenever

$$trace(\pi_1) \text{ and } trace(\pi_2) \text{ are both of the form } {A_0}^+ {A_1}^+ {A_2}^+ \ldots$$

for $A_i \subseteq AP$.

For positive integers n_i and m_i:

$$trace(\pi_1) = \underbrace{A_0 \ldots A_0}_{n_0 \text{ times}} \underbrace{A_1 \ldots A_1}_{n_1 \text{ times}} \underbrace{A_2 \ldots A_2}_{n_2 \text{ times}} \ldots$$

$$trace(\pi_2) = \underbrace{A_0 \ldots A_0}_{m_0 \text{ times}} \underbrace{A_1 \ldots A_1}_{m_1 \text{ times}} \underbrace{A_2 \ldots A_2}_{m_2 \text{ times}} \ldots$$

Stutter Trace Equivalence

Definition: stutter trace equivalence

Transition systems TS_i over AP, i=1, 2, are stutter-trace equivalent:

$$TS_1 \equiv_{sttrace} TS_2 \quad \text{if and only if} \quad TS_1 \trianglelefteq TS_2 \text{ and } TS_2 \trianglelefteq TS_1$$

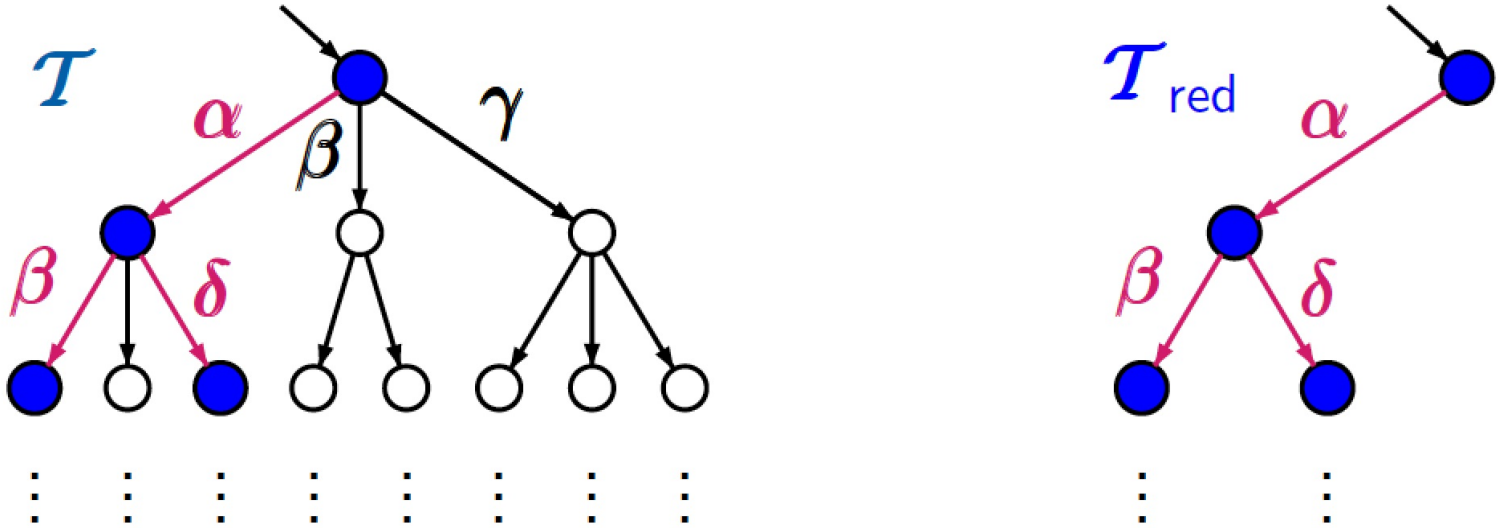
where the stutter trace inclusion relation $\trianglelefteq$ is defined by:

$$TS_1 \trianglelefteq TS_2 \quad \text{iff} \quad \forall \sigma_1 \in Traces(TS_1)\ \big(\exists \sigma_2 \in Traces(TS_2).\ \ \sigma_1 \equiv_{sttrace} \sigma_2\ \big)$$

Corollary

$$TS_1 \equiv_{sttrace} TS_2 \quad \text{if and only if} \quad \Big(TS_1 \equiv_{LTL\setminus\bigcirc} TS_2\Big).$$

Idea of Ample Sets

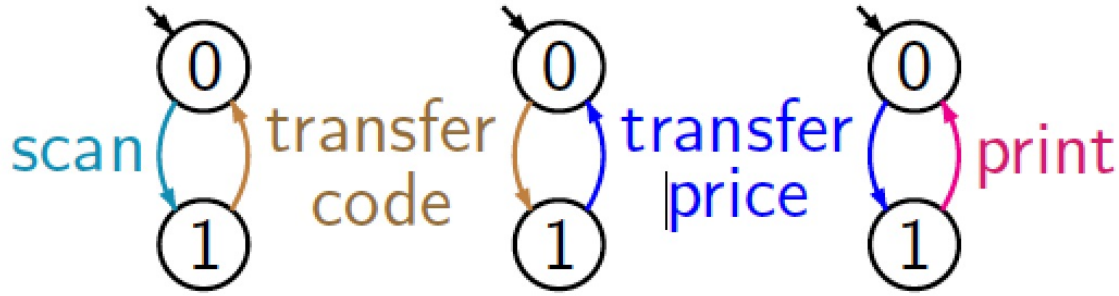


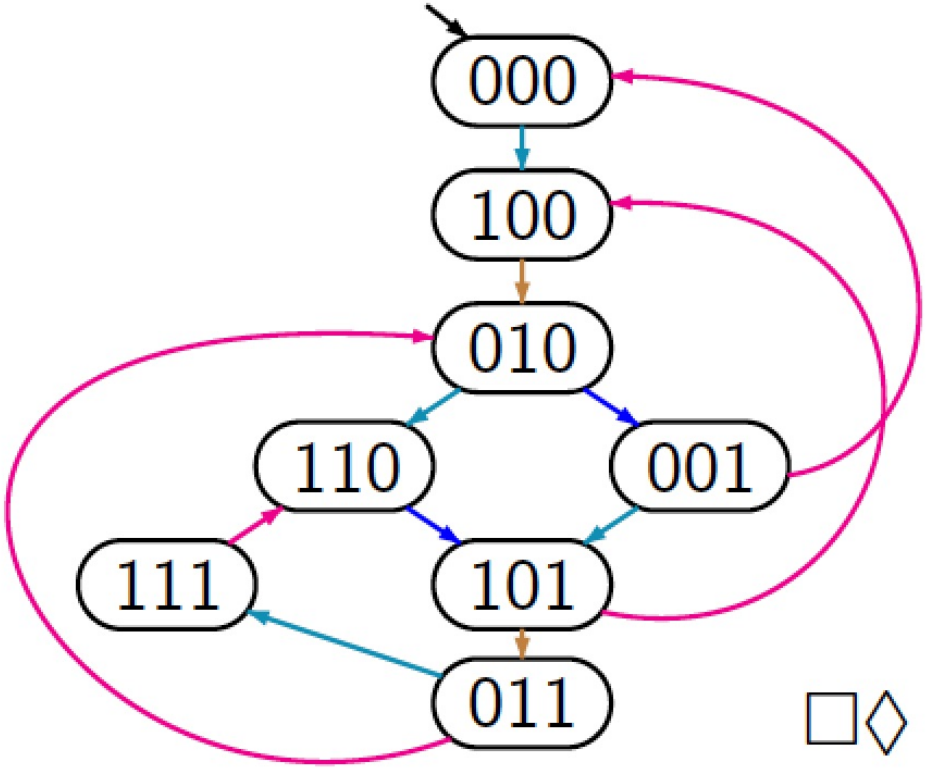
Transition relation $\Longrightarrow$ of reduced transition system TS_{red} defined by:

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

The actions outside of $\mathit{ample}(s)$ are pruned.

Example: Booking System





independent actions:

scan - transf. price

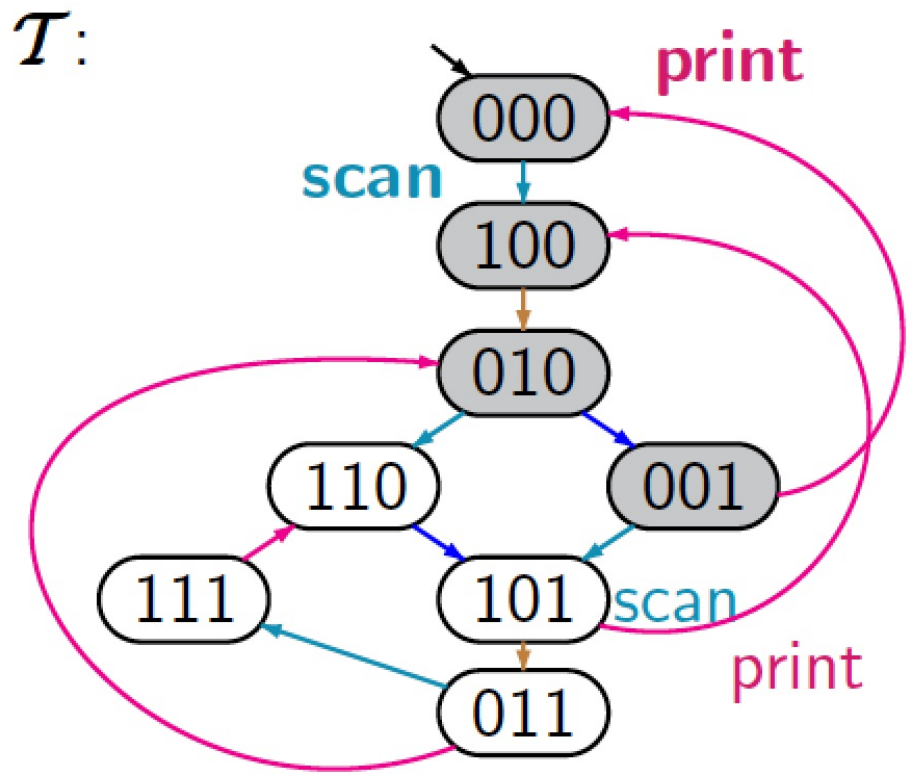
transfer code - print

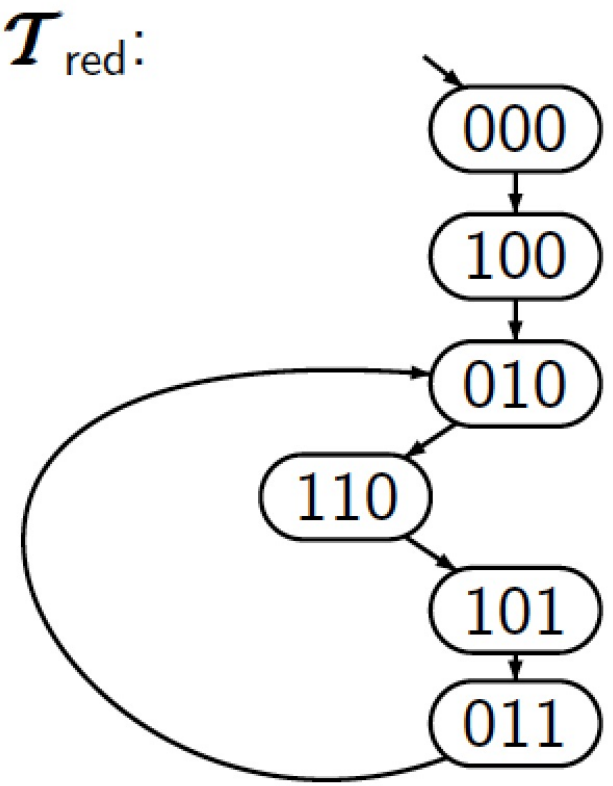
scan - print

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

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

Example: Booking System

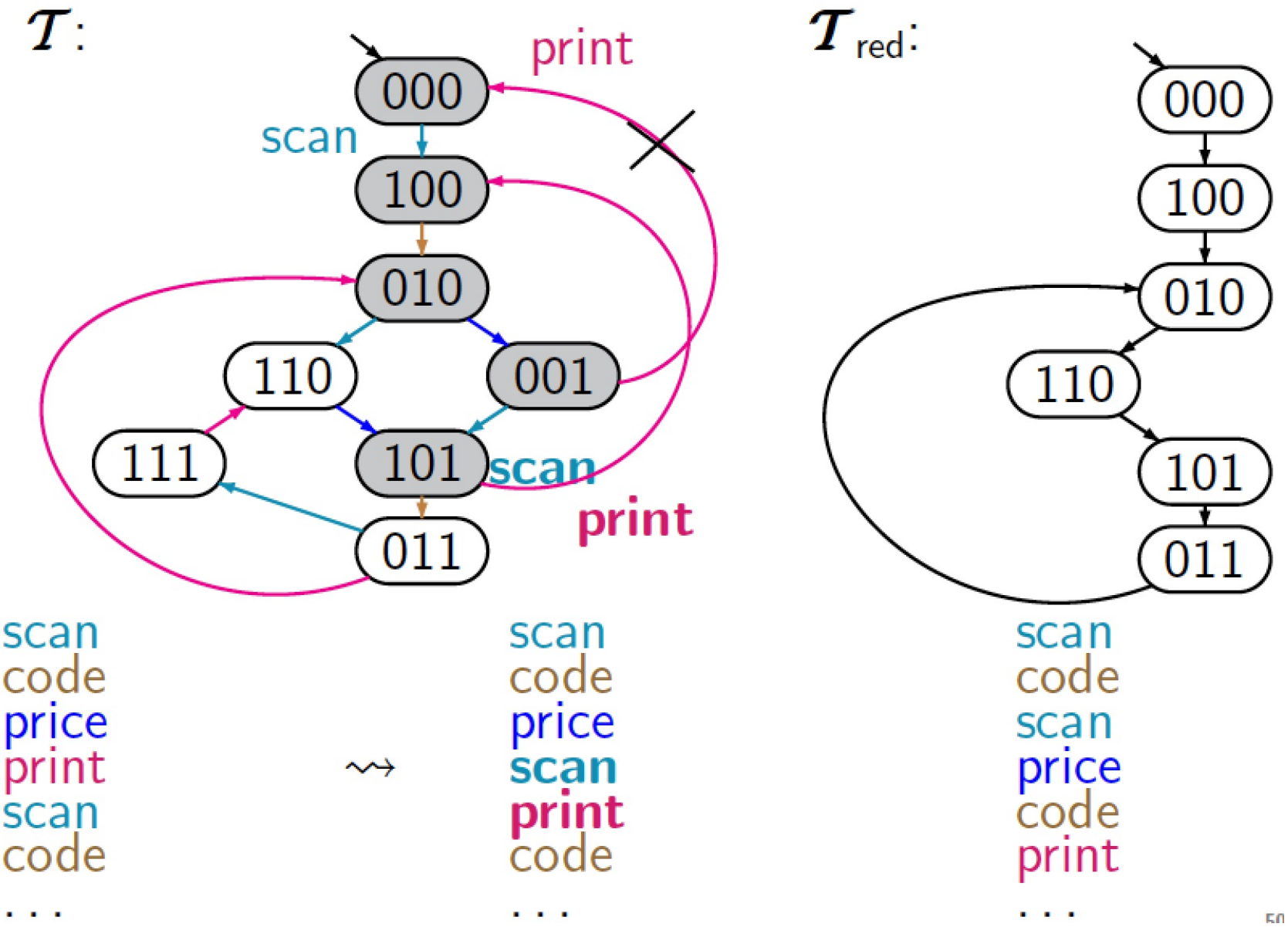




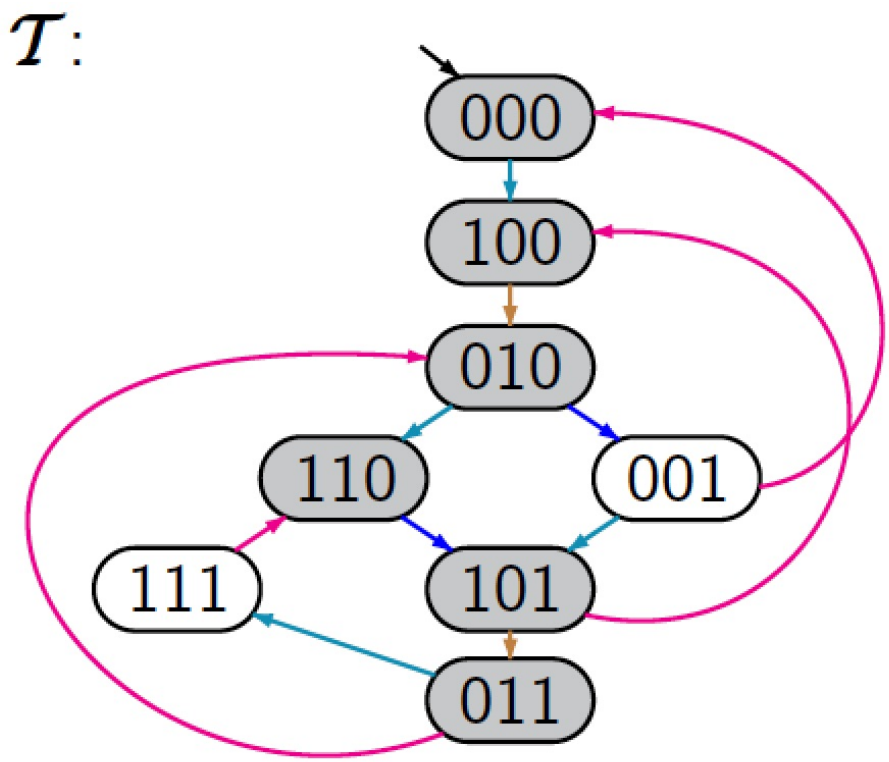
scan
code
price
print
scan
code
...

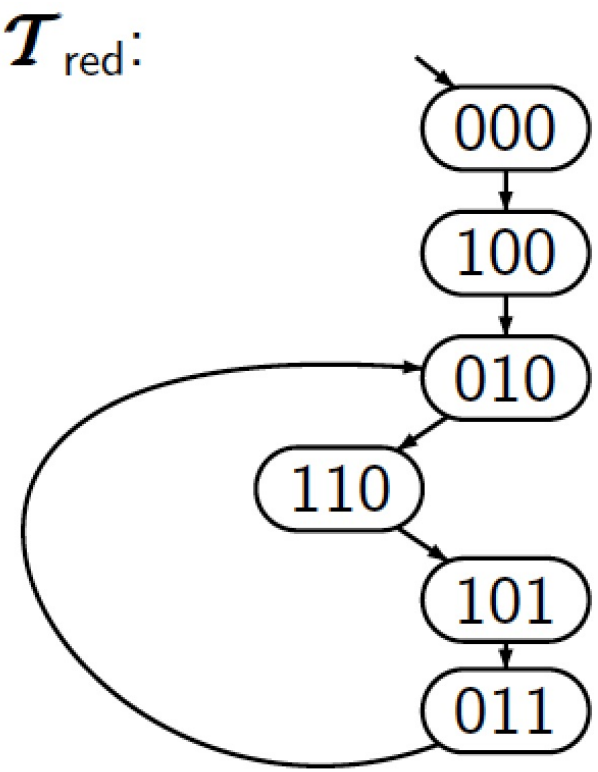
scan
code
scan
price
code
print
...

Example: Booking System



Example: Booking System





scan
code
price
print
scan
code
...

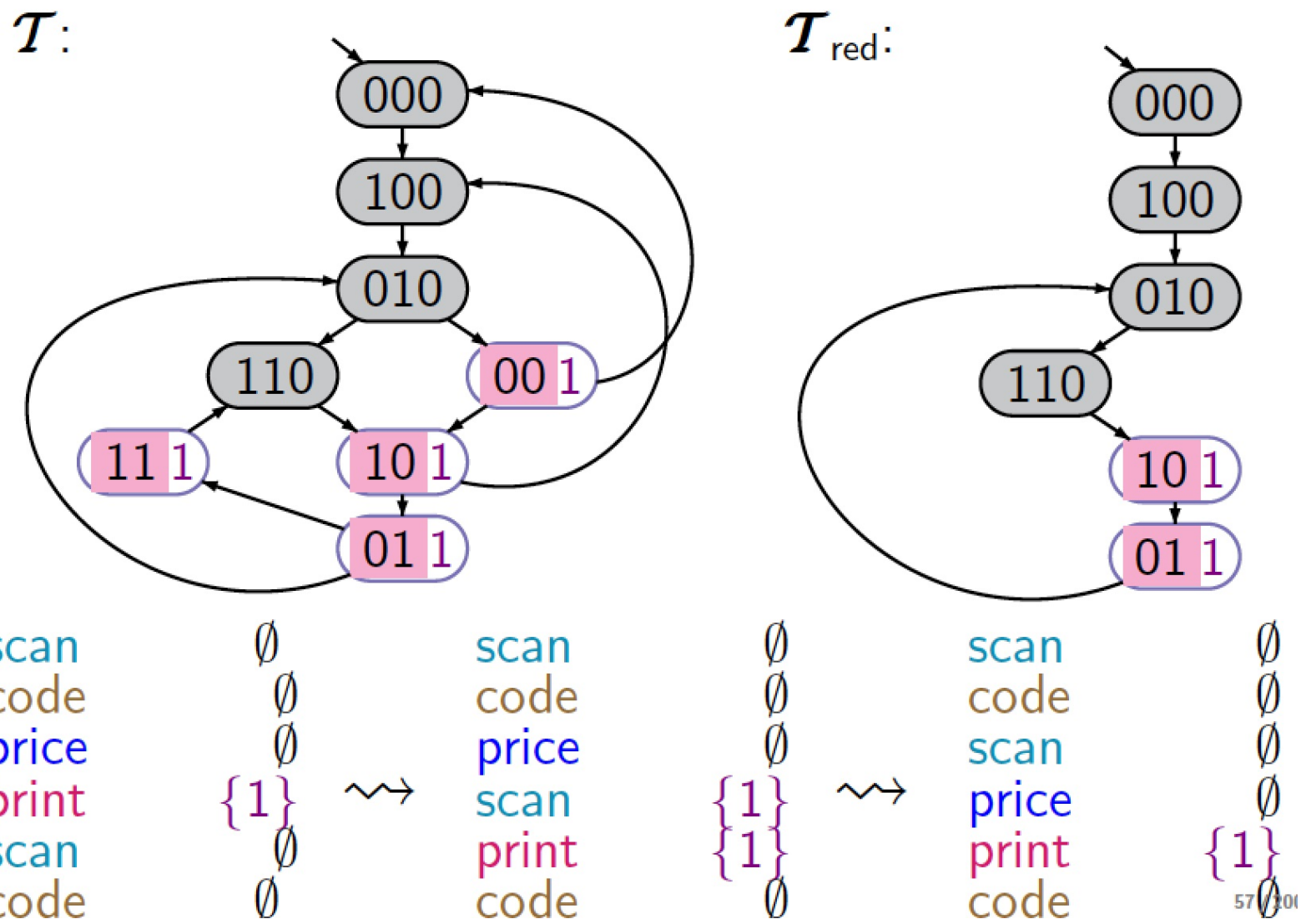
$\rightsquigarrow$

scan
code
scan
price
code
print
...

$=$

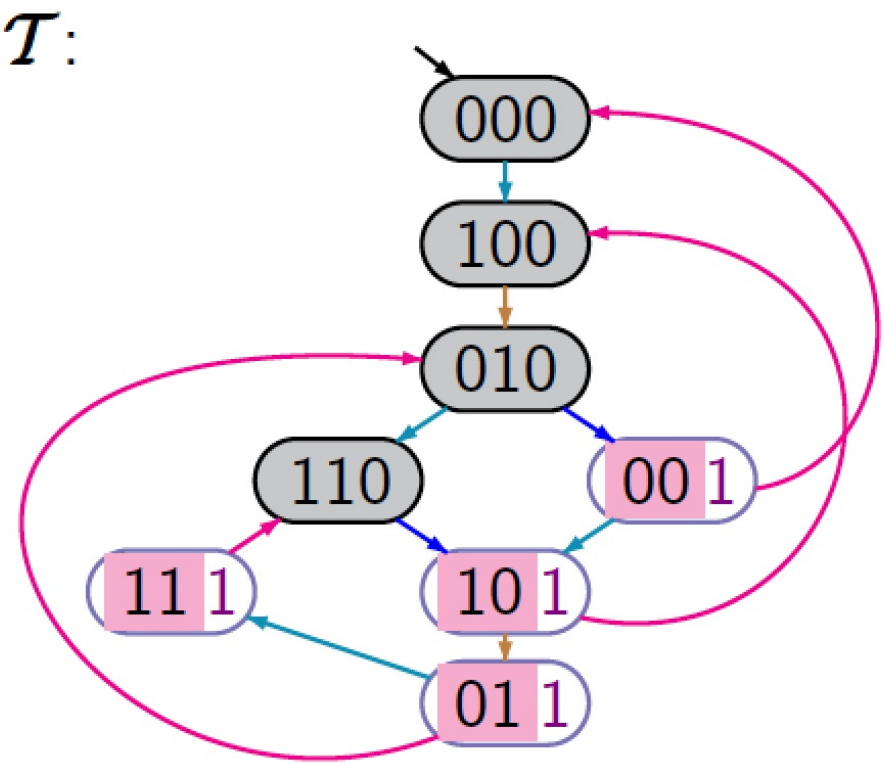
scan
code
scan
price
code
print
...

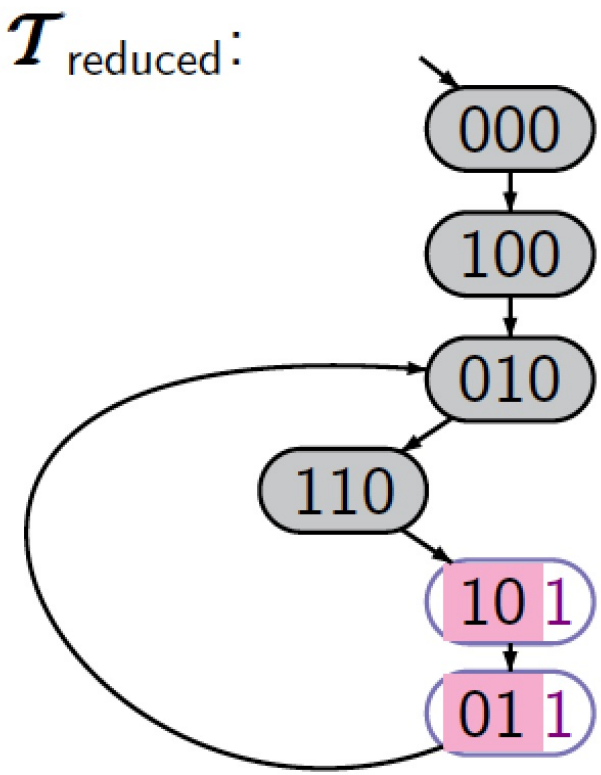
Example: Booking System



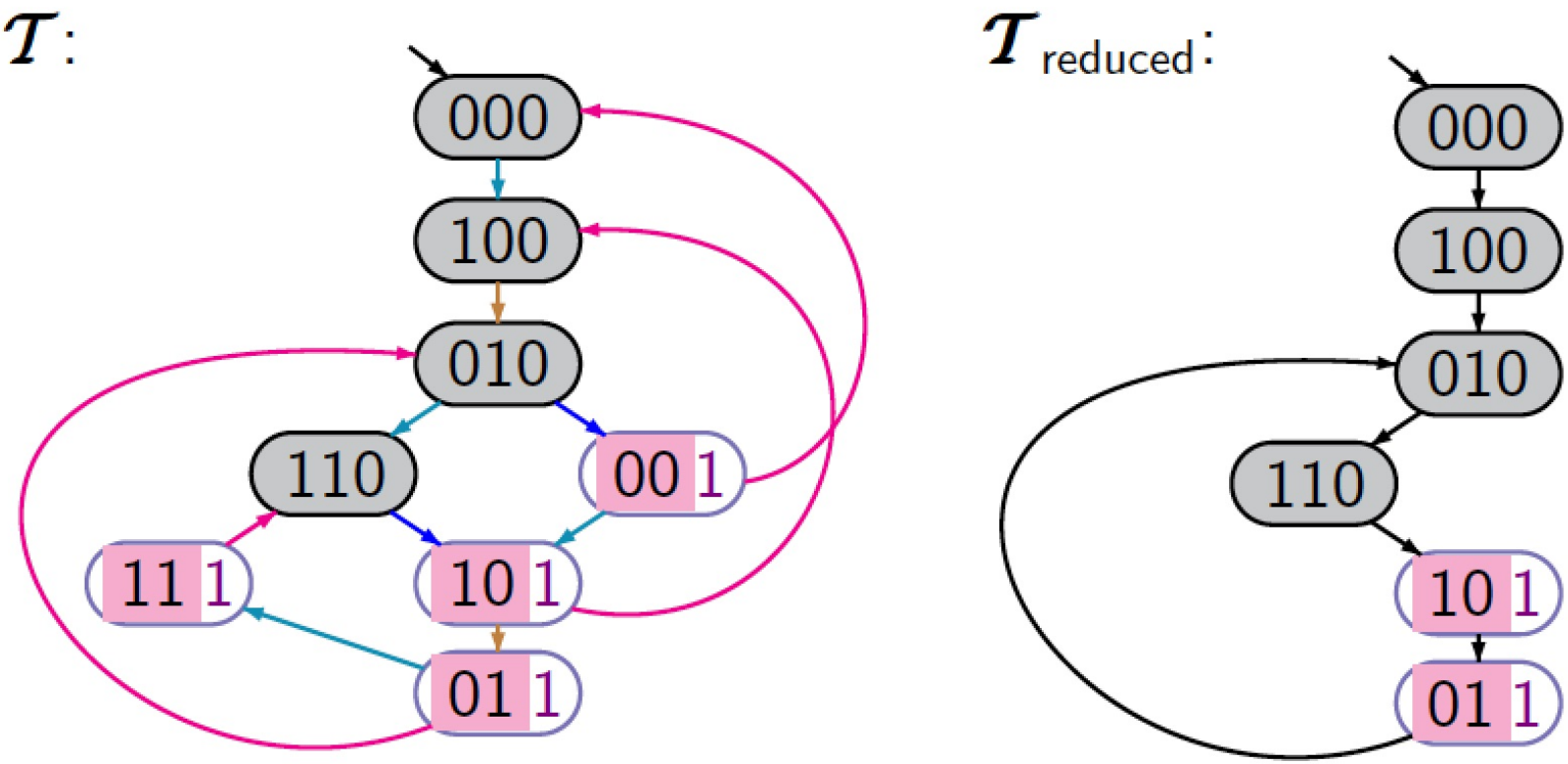
Proposition = "printer is in control state 1".

Example: Booking System





Example: Booking System



$TS_{red} \equiv_{sttrace} TS$, hence $TS_{red} \models \varphi$ iff $TS \models \varphi$

for $\varphi \in \mathrm{LTL}_{\setminus\bigcirc}$, e.g., $\varphi = \Box\Diamond$ "printer is in control state 1"

Overview

Action Determinism

Definition: action deterministic

Transition system TS is action deterministic whenever for any state s in TS and action α, it holds $s \xrightarrow{\alpha} u$ and $s \xrightarrow{\alpha} t$ implies $u = t$.

Action Determinism

Definition: action deterministic

Transition system TS is action deterministic whenever for any state s in TS and action α, it holds $s \xrightarrow{\alpha} u$ and $s \xrightarrow{\alpha} t$ implies $u = t$.

Every transition system can be made action deterministic by renaming actions.

Assumption: from now on, transition systems are action deterministic.

Let $\alpha(s)$ denote the unique α-successor of s, i.e., $s \xrightarrow{\alpha} \alpha(s)$.

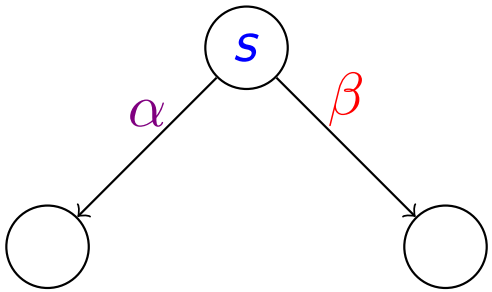
Action Independence

Definition: action independence

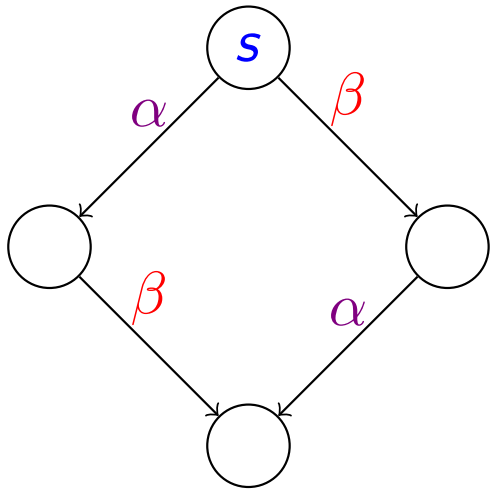
Let TS be an action-deterministic transition system with action-set Act.

Actions $\alpha \in Act$ and $\beta \in Act$ are independent in TS if for all states s with $\{\alpha, \beta\} \subseteq Act(s)$ the following holds:

$$\beta \in Act(\alpha(s)) \quad \text{and} \quad \alpha \in Act(\beta(s)) \quad \text{and} \quad \beta(\alpha(s)) = \alpha(\beta(s)).$$



can always be completed to



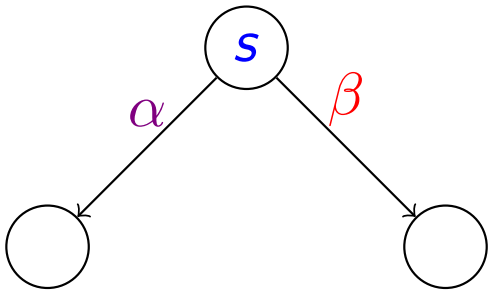
Action Independence

Definition: action independence

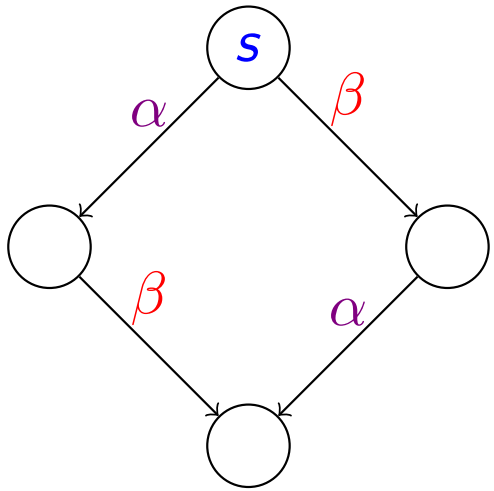
Let TS be an action-deterministic transition system with action-set Act.

Actions $\alpha \in Act$ and $\beta \in Act$ are independent in TS if for all states s with $\{\alpha, \beta\} \subseteq Act(s)$ the following holds:

$$\beta \in Act(\alpha(s)) \quad \text{and} \quad \alpha \in Act(\beta(s)) \quad \text{and} \quad \beta(\alpha(s)) = \alpha(\beta(s)).$$

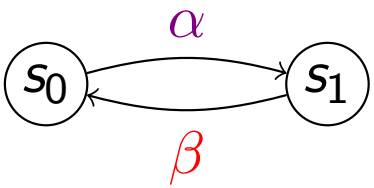


can always be completed to

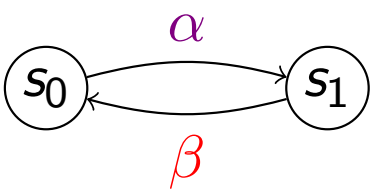


- $\alpha, \beta \in Act$ are called dependent if they are not independent

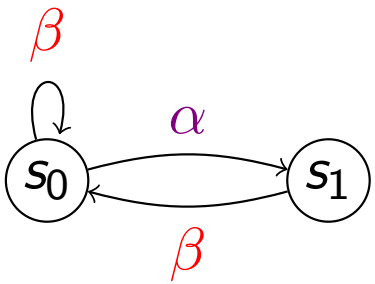
Independent or Not?



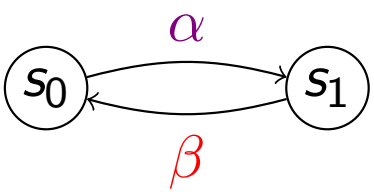
Independent or Not?



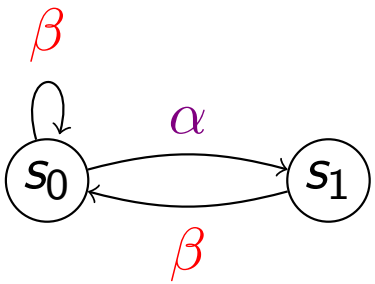
Independent: there is no s with $\{\alpha, \beta\} \subseteq Act(s)$.



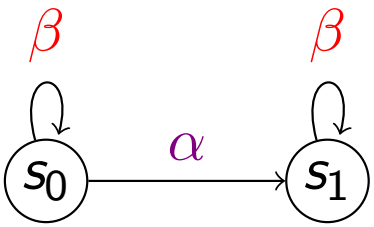
Independent or Not?



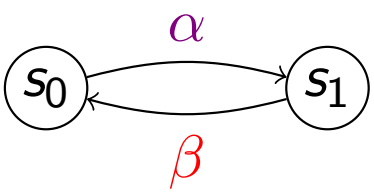
Independent: there is no s with $\{\alpha, \beta\} \subseteq Act(s)$.



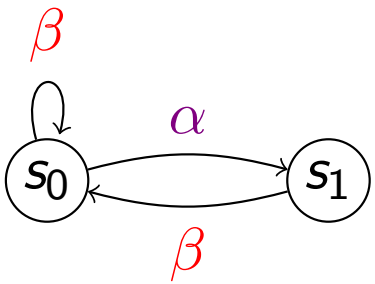
Dependent:, $\beta(\alpha(s_0)) = s_0 \neq s_1 = \alpha(\beta(s_0))$.



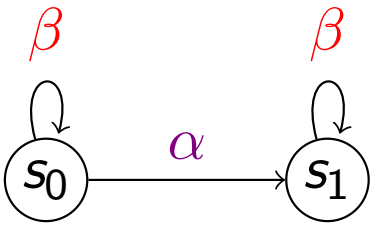
Independent or Not?



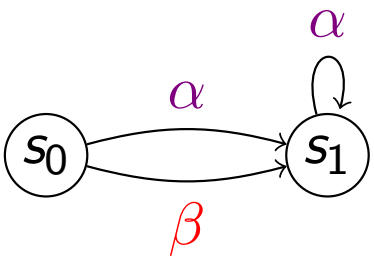
Independent: there is no s with $\{\alpha, \beta\} \subseteq Act(s)$.



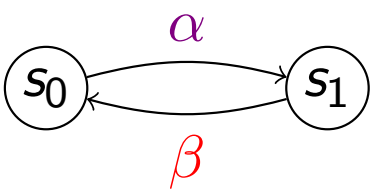
Dependent:, $\beta(\alpha(s_0)) = s_0 \neq s_1 = \alpha(\beta(s_0))$.



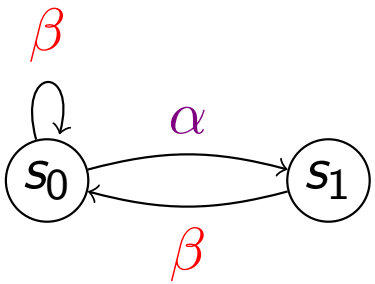
Independent: $\beta(\alpha(s_0)) = s_1 = \alpha(\beta(s_0))$.



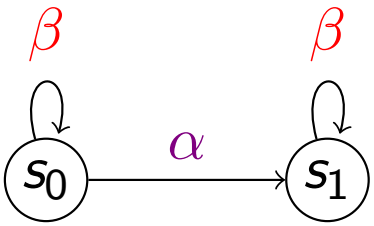
Independent or Not?



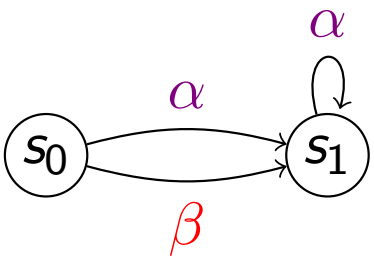
Independent: there is no s with $\{\alpha, \beta\} \subseteq Act(s)$.



Dependent:, $\beta(\alpha(s_0)) = s_0 \neq s_1 = \alpha(\beta(s_0))$.



Independent: $\beta(\alpha(s_0)) = s_1 = \alpha(\beta(s_0))$.



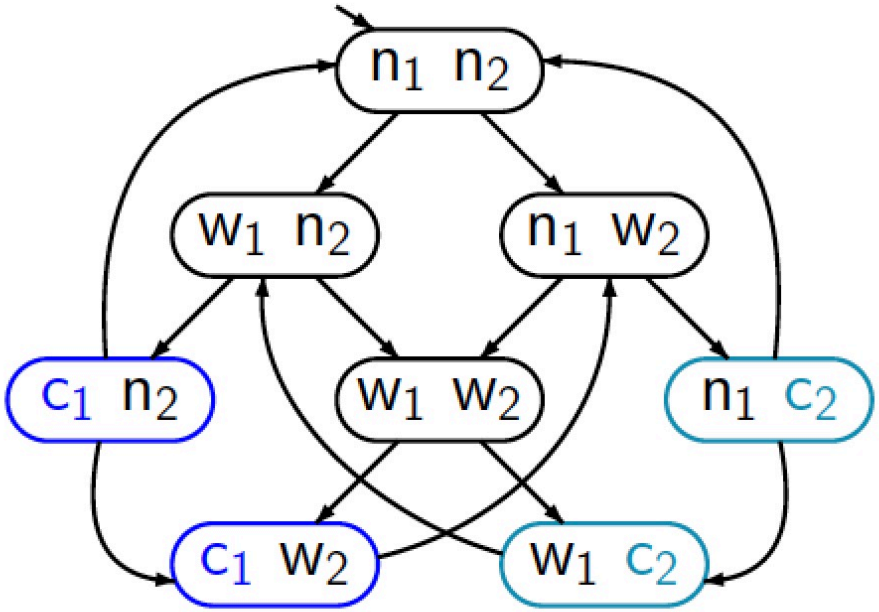
Dependent: $\beta \notin Act(\alpha(s_0))$

Action Independence

The notion of indepencence (and dependence) is lifted to sets:

- for $B \subseteq Act$ and $\alpha \in Act \setminus B$, α is independent of B if for all $\beta \in B$, α is independent of β
- $\alpha \in Act$ is dependent on $B \subseteq Act$ if $\alpha \in Act \setminus B$ and there is some $\beta \in B$ such that α and β are dependent.

Example: Semaphore-Based Mutual Exclusion

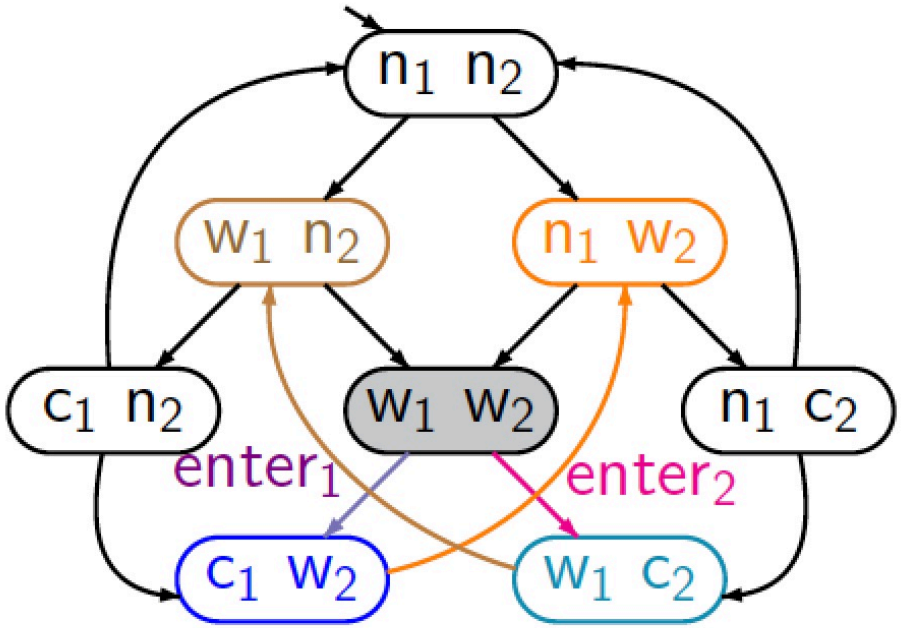


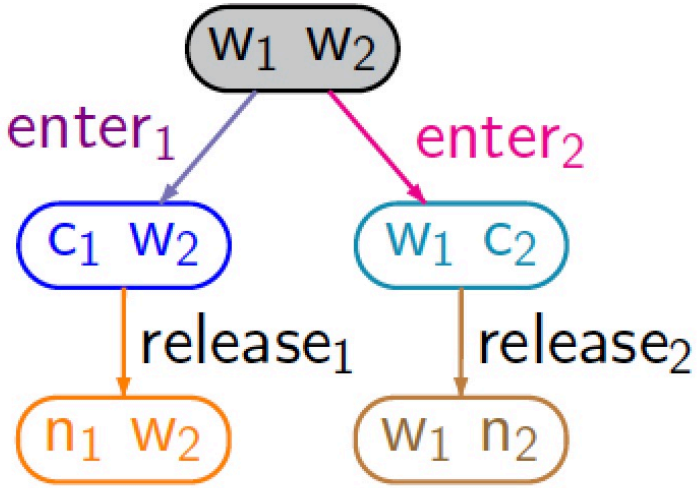
independent actions:

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

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

Example: Semaphore-Based Mutual Exclusion



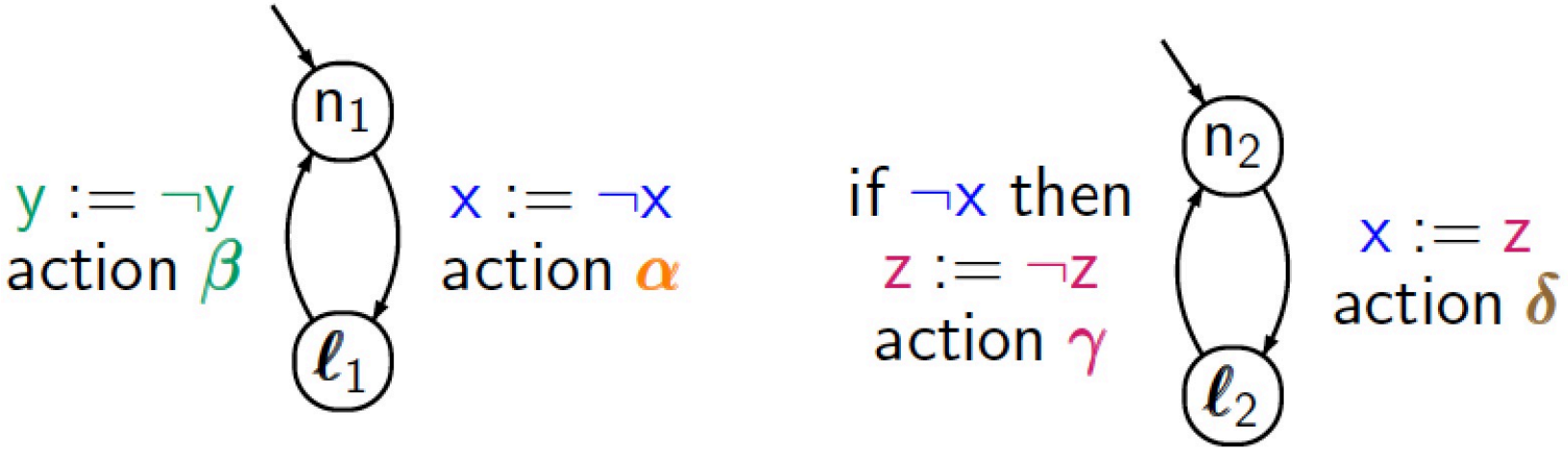


dependent actions:

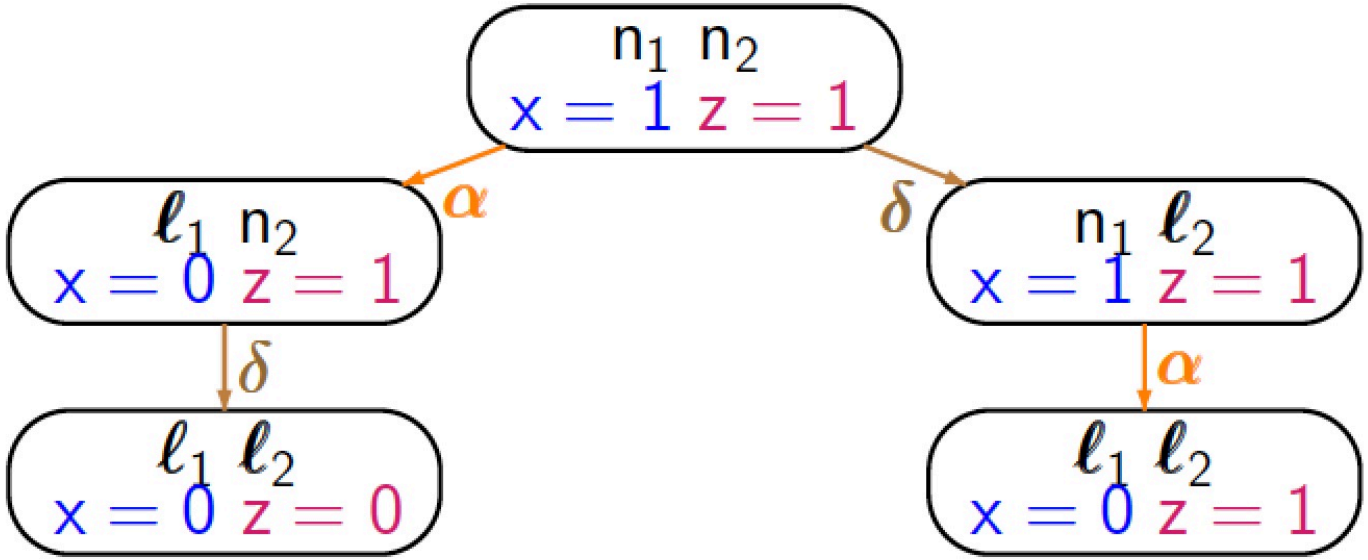
$enter_1$, $enter_2$

access both to the semaphore

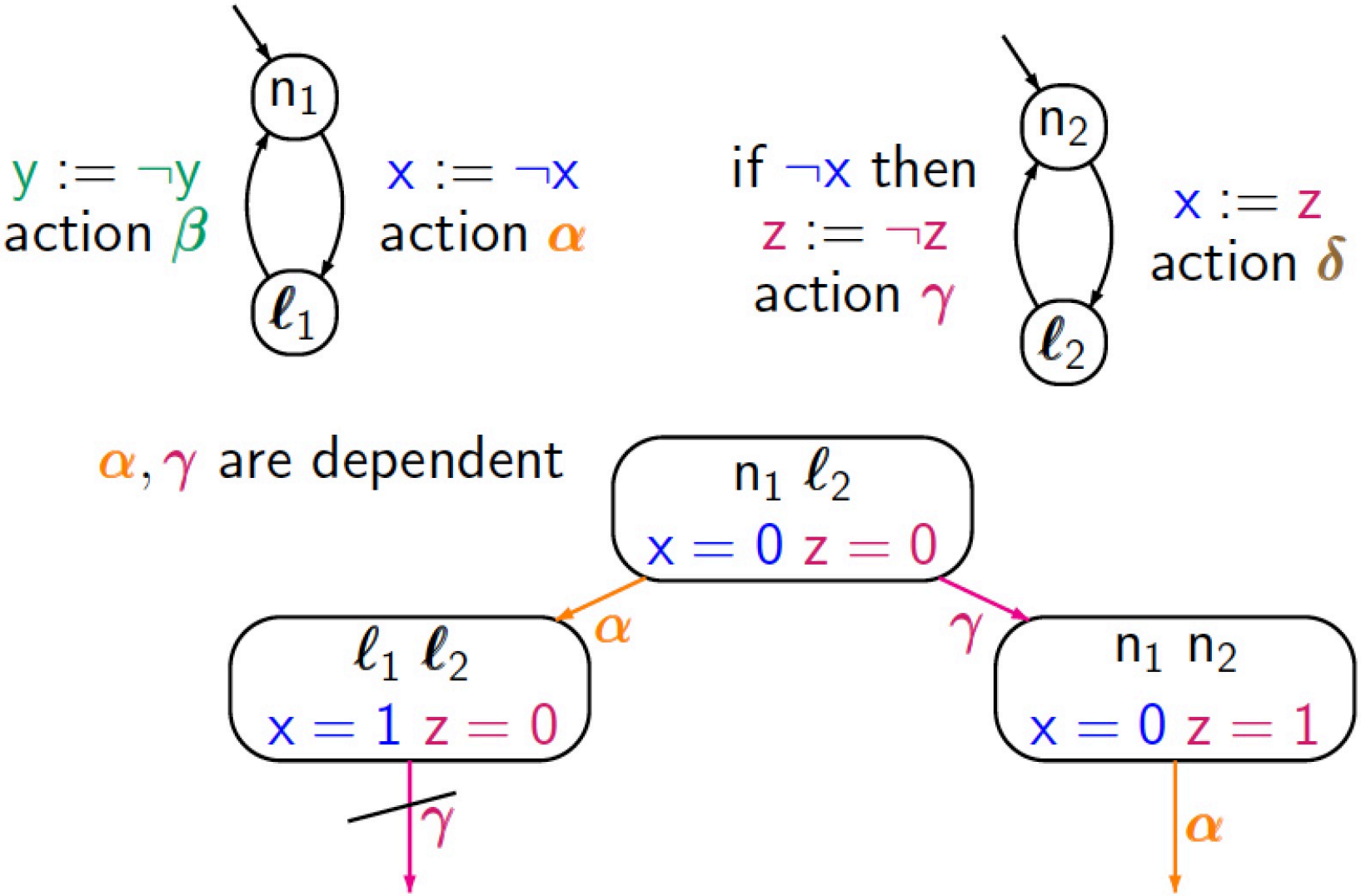
Example: Shared Variables



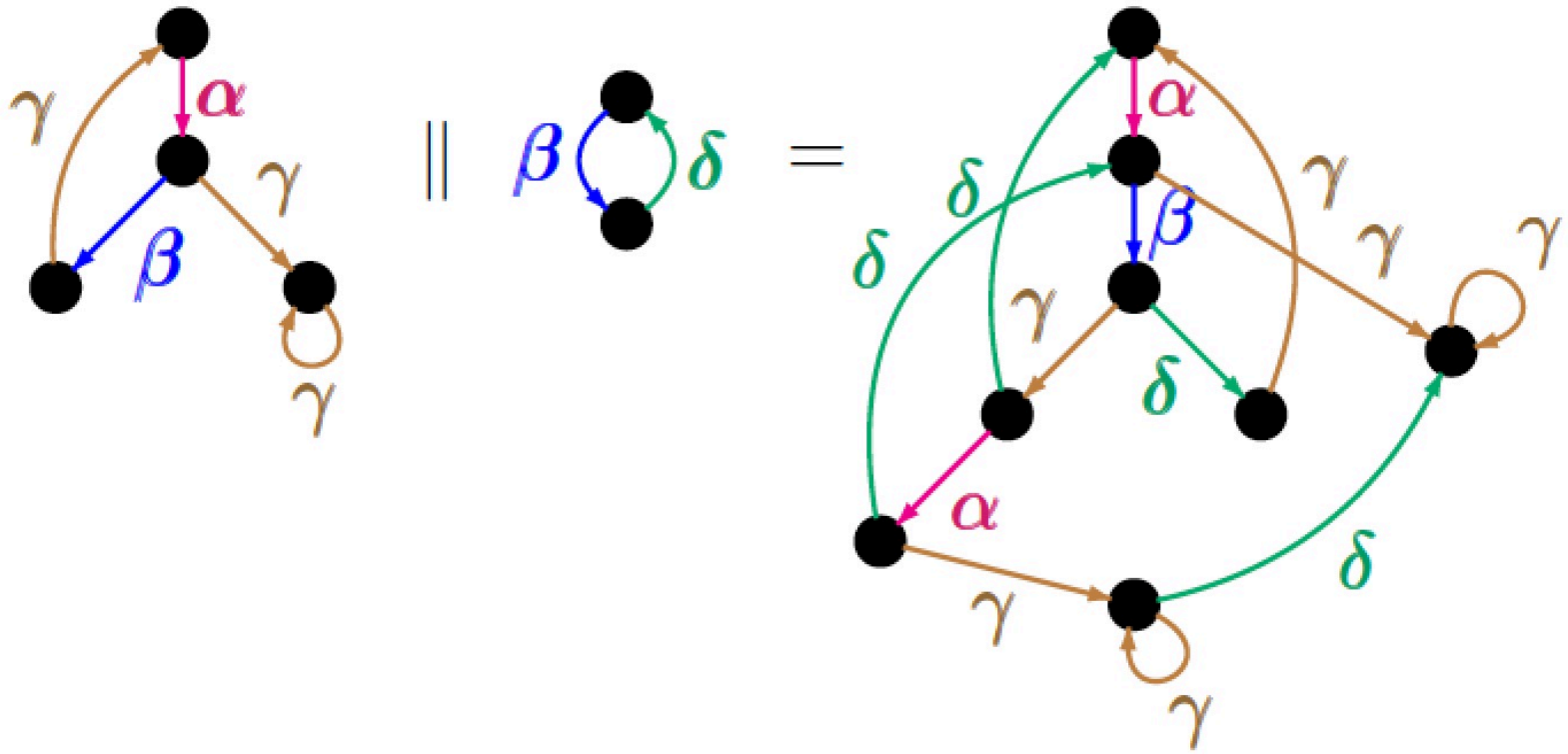
α, δ are dependent for $\mathcal{T}_{P_1 ||| P_2}$



Example: Shared Variables



Example: Synchronised Threads



α, δ independent $\surd$

γ, δ independent $\surd$

β, γ dependent

Permuting Independent Actions

Let TS be action-deterministic, s a state in TS and:

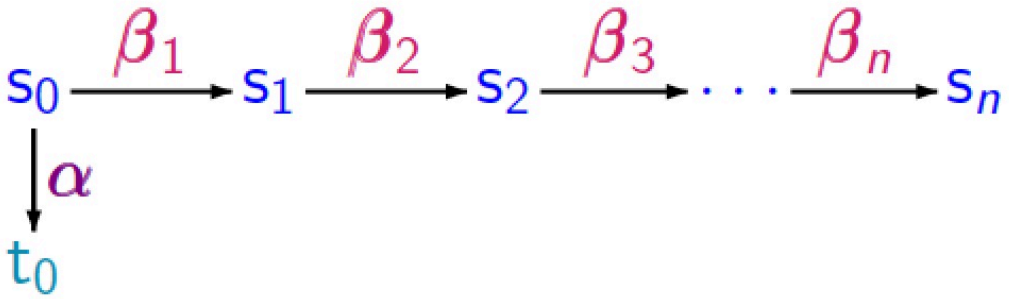
$$s = s_0 \xrightarrow{\beta_1} s_1 \xrightarrow{\beta_2} \ldots \xrightarrow{\beta_{n-1}} s_{n-1} \xrightarrow{\beta_n} s_n$$

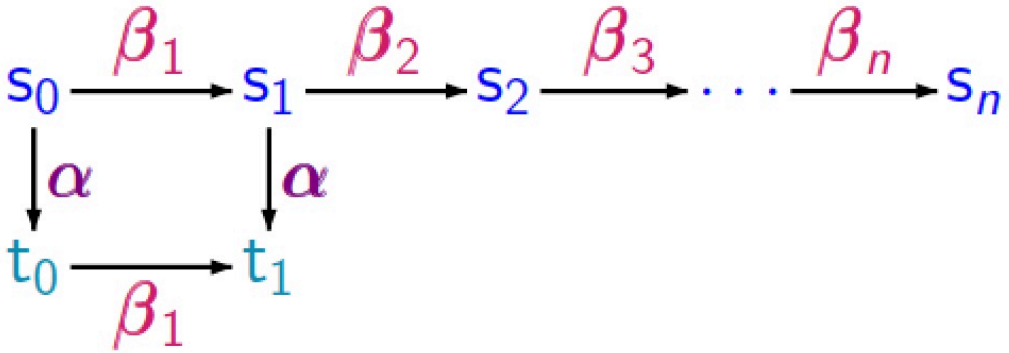
be a finite run in TS from s with action sequence $\beta_1 \ldots \beta_n$.

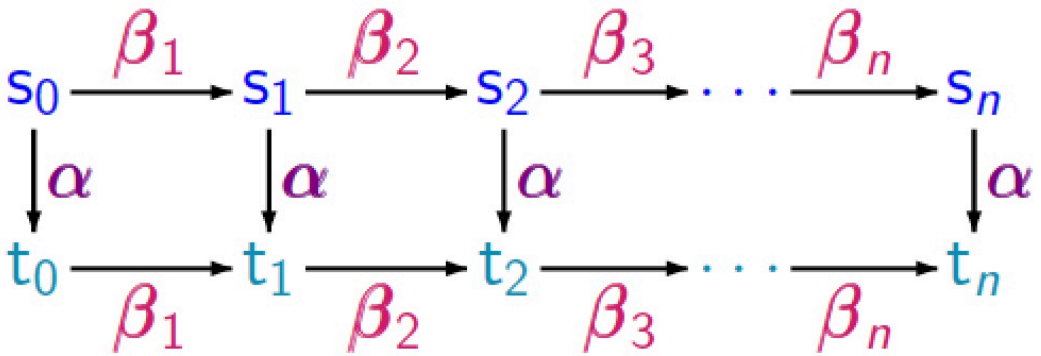
Then, for $\alpha \in Act(s)$ independent of $\{\beta_1, \ldots, \beta_n\}$:

- $\alpha \in Act(s_i)$ for all $i \in \{0, \ldots, n\}$ and
- $s = s_0 \xrightarrow{\alpha} \alpha(s_0) \xrightarrow{\beta_1} \alpha(s_1) \xrightarrow{\beta_2} \ldots \xrightarrow{\beta_{n-1}} \alpha(s_{n-1}) \xrightarrow{\beta_n} \alpha(s_n)$ is a run in TS from s with action sequence $\alpha\,\beta_1 \ldots \beta_n$

Pictorial Proof Sketch







Stutter Actions

- If no further assumptions are made, the traces of the runs:

$$\rho = s_0 \xrightarrow{\beta_1} s_1 \xrightarrow{\beta_2} \ldots \xrightarrow{\beta_n} s_n \xrightarrow{\alpha} t \text{ and}$$

$$\rho' = s_0 \xrightarrow{\alpha} t_0 \xrightarrow{\beta_1} \ldots \xrightarrow{\beta_{n-1}} t_{n-1} \xrightarrow{\beta_n} t$$

will be distinct

Stutter Actions

- If no further assumptions are made, the traces of the runs:

$$\rho = s_0 \xrightarrow{\beta_1} s_1 \xrightarrow{\beta_2} \ldots \xrightarrow{\beta_n} s_n \xrightarrow{\alpha} t \text{ and}$$

$$\rho' = s_0 \xrightarrow{\alpha} t_0 \xrightarrow{\beta_1} \ldots \xrightarrow{\beta_{n-1}} t_{n-1} \xrightarrow{\beta_n} t$$

  will be distinct

- If α does not affect the state-labelling ($=$ "invisible"): $\rho \equiv_{sttrace} \rho'$.

Definition: stutter action

Action $\alpha \in Act$ is a stutter action if for each $s \xrightarrow{\alpha} s'$ in TS: $L(s) = L(s')$.

Equivalently: α is a stutter action if all transitions $s \xrightarrow{\alpha} s'$ are stutter steps.

Permuting Independent Stutter Actions

Let TS be action-deterministic, s a state in TS and:

- ϱ a finite run from s with action sequence $\beta_1 \ldots \beta_n \alpha$
- ϱ' a finite run from s with action sequence $\alpha \beta_1 \ldots \beta_n$

Then:

if α is a stutter action and independent of $\{\beta_1, \ldots, \beta_n\}$, then $\varrho \equiv_{sttrace} \varrho'$.

Adding Independent Stutter Actions

Let TS be action-deterministic, s a state in TS and:

- ρ an infinite run from s with action sequence $\beta_1 \beta_2 \ldots$
- ρ' an infinite run from s with action sequence $\alpha \beta_1 \beta_2 \ldots$

Then:

if α is a stutter action and independent of $\{\beta_1, \beta_2, \ldots\}$, then $\rho \equiv_{sttrace} \rho'$.

Overview

The Ample-Set Approach

- Partial-order reduction for LTL formulas using ample sets
  - on state-space generation select $ample(s) \subseteq Act(s)$
  - such that $|ample(s)| \ll |Act(s)|$

- Reduced system $TS_{red} = (S', Act, \Longrightarrow, I, AP, L')$ where:
  - S' = the set of states reachable from some $s_0 \in I$ under $\Longrightarrow$
  - $\Longrightarrow$ is the smallest relation defined by:

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

  - $L'(s) = L(s)$ for any $s \in S'$

- Constraints: correctness ($\equiv_{sttrace}$), effectiveness and efficiency

Which Actions to Put in $ample(s)$?

(A1) **Non-emptiness condition**

Select in any state in TS_{red} at least one action.

Which Actions to Put in $ample(s)$?

(A1) **Non-emptiness condition**

Select in any state in TS_{red} at least one action.

(A2) **Dependency condition**

For any finite run in TS: an action depending on $ample(s)$ can only occur after some action in $ample(s)$ has occurred.

Which Actions to Put in $ample(s)$?

(A1) **Non-emptiness condition**

Select in any state in TS_{red} at least one action.

(A2) **Dependency condition**

For any finite run in TS: an action depending on $ample(s)$ can only occur after some action in $ample(s)$ has occurred.

(A3) **Stutter condition**

If an enabled action in s is not selected, then all selected actions are stutter actions.

Which Actions to Put in $ample(s)$?

(A1) **Non-emptiness condition**

Select in any state in TS_{red} at least one action.

(A2) **Dependency condition**

For any finite run in TS: an action depending on $ample(s)$ can only occur after some action in $ample(s)$ has occurred.

(A3) **Stutter condition**

If an enabled action in s is not selected, then all selected actions are stutter actions.

(A4) **Cycle condition**

Any action in $Act(s_i)$ (in TS) with s_i on a cycle in TS_{red} must be selected in some s_j on that cycle.

Which Actions to Put in $ample(s)$?

(A1) **Non-emptiness condition**
Select in any state in TS_{red} at least one action.

(A2) **Dependency condition**
For any finite run in TS: an action depending on $ample(s)$ can only occur after some action in $ample(s)$ has occurred.

(A3) **Stutter condition**
If an enabled action in s is not selected, then all selected actions are stutter actions.

(A4) **Cycle condition**
Any action in $Act(s_i)$ (in TS) with s_i on a cycle in TS_{red} must be selected in some s_j on that cycle.

(A1) through (A3) apply to state s in TS_{red}; (A4) to cycles in TS_{red}.

Example

Dependency Condition (A2)

Dependency Condition (A2)

Let $s \xrightarrow{\beta_1} s_1 \xrightarrow{\beta_2} \ldots \xrightarrow{\beta_n} s_n \xrightarrow{\alpha} t$ be a finite run in TS such that α depends on $ample(s)$.

Then: $\beta_i \in ample(s)$ for some $0 < i \leq n$.

Dependency Condition (A2)

Dependency Condition (A2)

Let $s \xrightarrow{\beta_1} s_1 \xrightarrow{\beta_2} \ldots \xrightarrow{\beta_n} s_n \xrightarrow{\alpha} t$ be a finite run in TS such that α depends on $ample(s)$.

Then: $\beta_i \in ample(s)$ for some $0 < i \leq n$.

Action α depends on the ample set, if it depends on some action in this set

- In every (!) finite run of TS, an action dependent on $ample(s)$ cannot occur before some action from $ample(s)$ occurs first
- (A2) ensures that for any state s with $ample(s) \neq Act(s)$, any $\alpha \in ample(s)$ is independent of $Act(s) \setminus ample(s)$

Naive Dependency Condition (A2')

Naive Dependency Condition (A2')

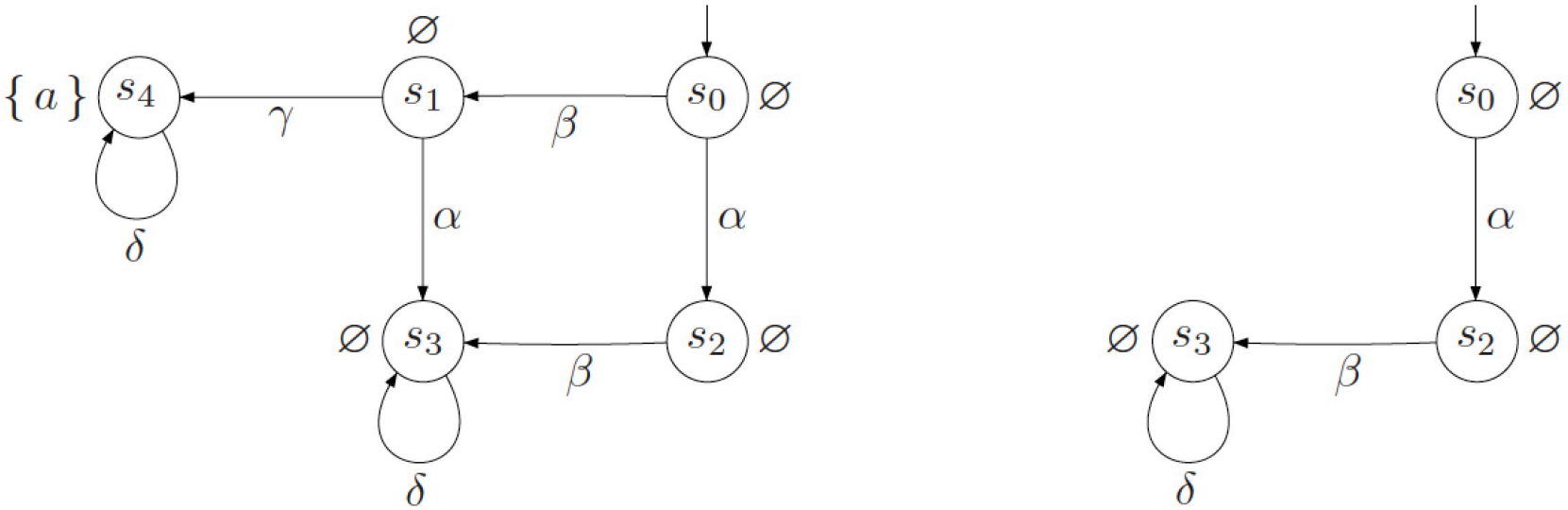
For any $s \in S'$ with $\mathit{ample}(s) \neq \mathit{Act}(s)$:
$\alpha \in \mathit{ample}(s)$ is independent of $\mathit{Act}(s) \setminus \mathit{ample}(s)$.

Naive Dependency Condition (A2')

Naive Dependency Condition (A2')

For any $s \in S'$ with $ample(s) \neq Act(s)$:
$\alpha \in ample(s)$ is independent of $Act(s) \setminus ample(s)$.

This is incorrect. (A2') allows the following reduction with $ample(s_0) = \alpha$:



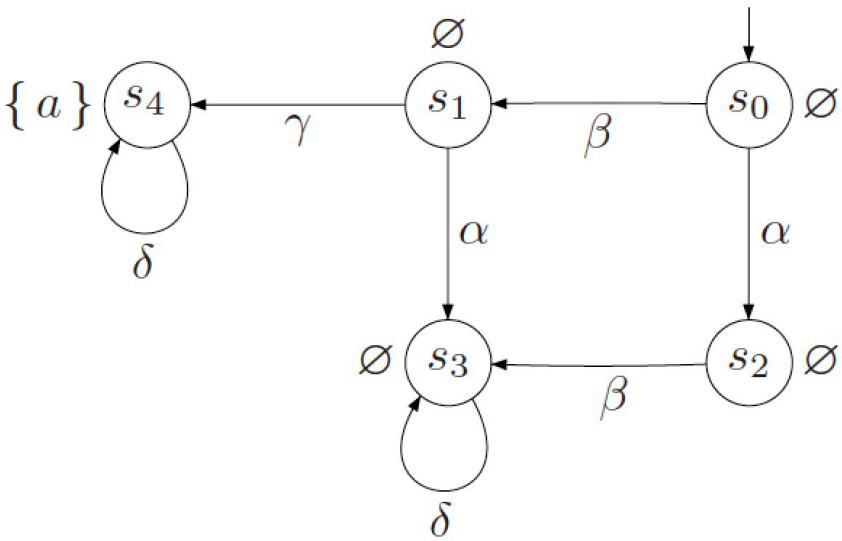
$TS \not\models \Box\neg a$ but $TS_{red} \models \Box\neg a$, so $TS \not\equiv_{sttrace} TS_{red}$

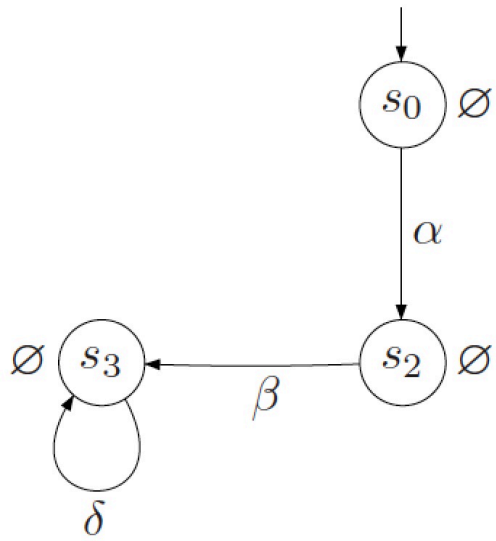
Dependency Condition (A2)

Dependency Condition (A2)

Let $s \xrightarrow{\beta_1} s_1 \xrightarrow{\beta_2} \ldots \xrightarrow{\beta_n} s_n \xrightarrow{\alpha} t$ be a finite run in TS such that α depends on $ample(s)$.

Then: $\beta_i \in ample(s)$ for some $0 < i \leq n$.





run $s_0 \xrightarrow{\beta} s_1 \xrightarrow{\gamma} s_4$ violates (A2) as γ depends on $\{\alpha\} = ample(s_0)$

Properties

For any $\alpha \in \mathit{ample}(s)$ and $s \in \mathit{Reach}(\mathit{TS})$:

If $\mathit{ample}(s)$ satisfies (A2), then α is independent of $\mathit{Act}(s) \setminus \mathit{ample}(s)$.

Properties

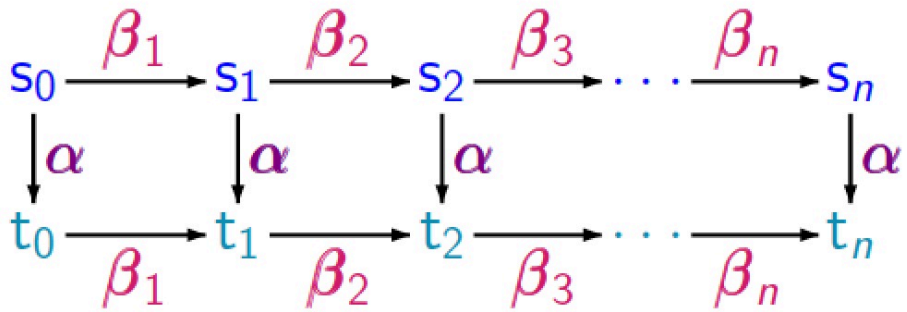
For any $\alpha \in \mathit{ample}(s)$ and $s \in \mathit{Reach}(\mathit{TS})$:

If $\mathit{ample}(s)$ satisfies (A2), then α is independent of $\mathit{Act}(s) \setminus \mathit{ample}(s)$.

For finite run $s = s_0 \xrightarrow{\beta_1} \ldots \xrightarrow{\beta_n} s_n$ in TS:

if $\mathit{ample}(s)$ satisfies (A2) and $\{\beta_1, \ldots, \beta_n\} \cap \mathit{ample}(s) = \emptyset$, then:

α is independent of $\{\beta_1, \ldots, \beta_n\}$ and $\alpha \in \mathit{Act}(s_i)$ for $0 \leq i \leq n$.



Ample Set Conditions So Far

(A1) **Nonemptiness condition**

$\varnothing \neq \mathit{ample}(s) \subseteq \mathit{Act}(s)$

(A2) **Dependency condition**

Let $s \xrightarrow{\beta_1} \ldots \xrightarrow{\beta_n} s_n \xrightarrow{\alpha} t$ be a finite run in TS such that α depends on $\mathit{ample}(s)$. Then: $\beta_i \in \mathit{ample}(s)$ for some $0 < i \leq n$.

(A3) **Stutter condition**

If $\mathit{ample}(s) \neq \mathit{Act}(s)$ then any $\alpha \in \mathit{ample}(s)$ is a stutter action.

First Consequence of (A1)–(A3)

Let ϱ be a finite run in $Reach(TS)$ of the form

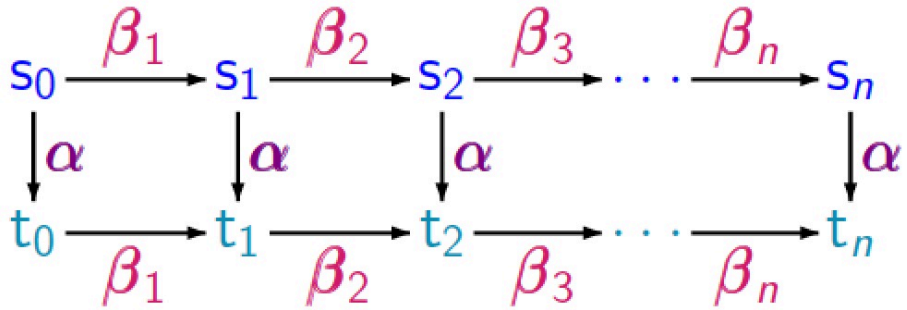
$$\varrho \;=\; s \xrightarrow{\beta_1} s_1 \xrightarrow{\beta_2} \ldots \xrightarrow{\beta_n} s_n \xrightarrow{\alpha} t$$

where $\beta_i \notin ample(s)$, for $0 < i \leq n$, and $\alpha \in ample(s)$.

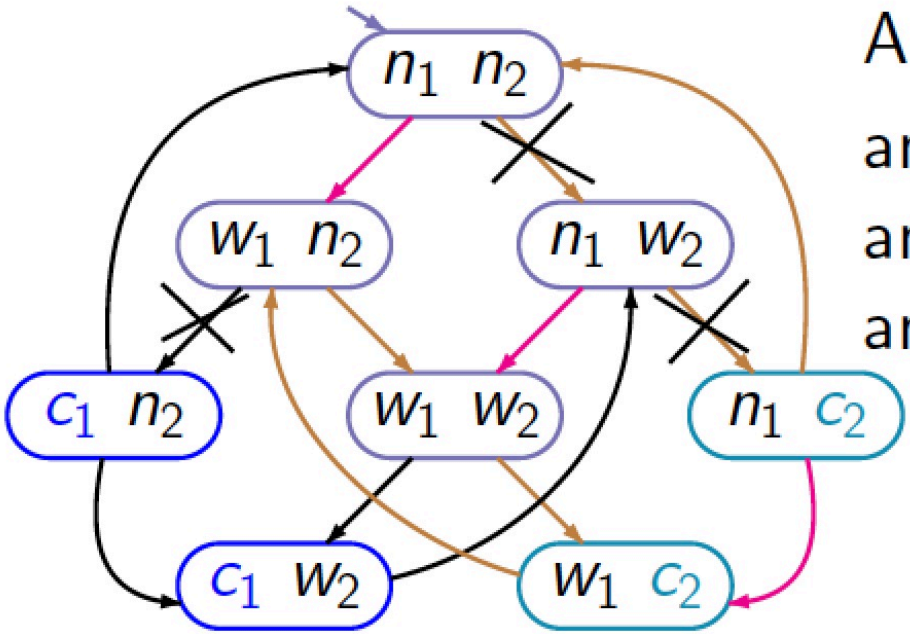
If $ample(s)$ satisfies (A1)–(A3), then there exists a run:

$$\varrho' \;=\; s \overset{\alpha}{\Longrightarrow} t_0 \xrightarrow{\beta_1} t_1 \xrightarrow{\beta_2} \ldots \xrightarrow{\beta_{n-1}} t_{n-1} \xrightarrow{\beta_n} t$$

such that $\varrho \equiv_{sttrace} \varrho'$.



Example: Ample Sets for Semaphore



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

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

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

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

$$\ldots$$

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

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

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

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

Second Consequence of (A1)–(A3)

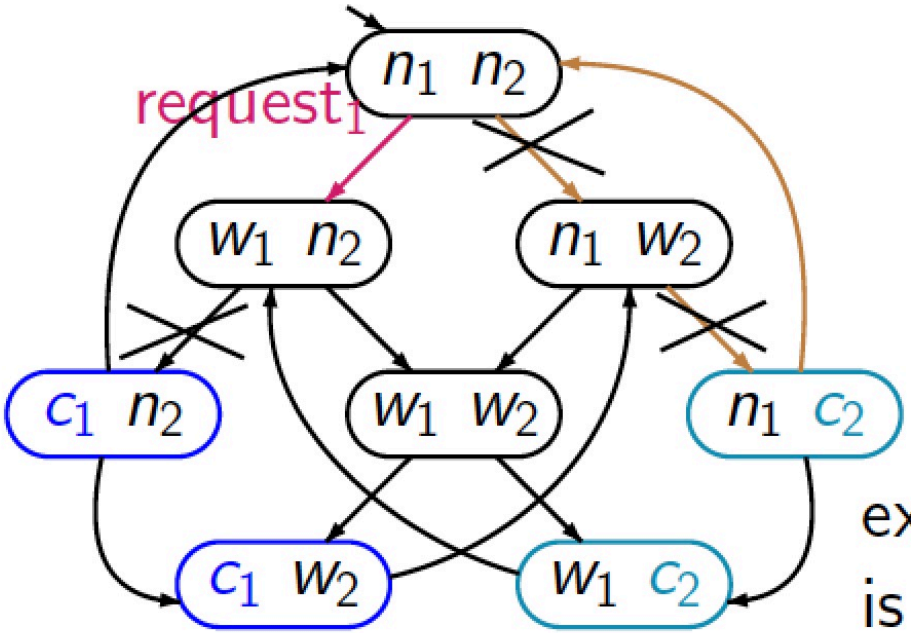
Let $\rho = s \xrightarrow{\beta_1} s_1 \xrightarrow{\beta_2} s_2 \xrightarrow{\beta_3} \ldots$ be an infinite run in $\mathit{Reach}(TS)$ where $\beta_i \notin \mathit{ample}(s)$, for $i > 0$.

If $\mathit{ample}(s)$ satisfies (A1)–(A3), then there exists a run:

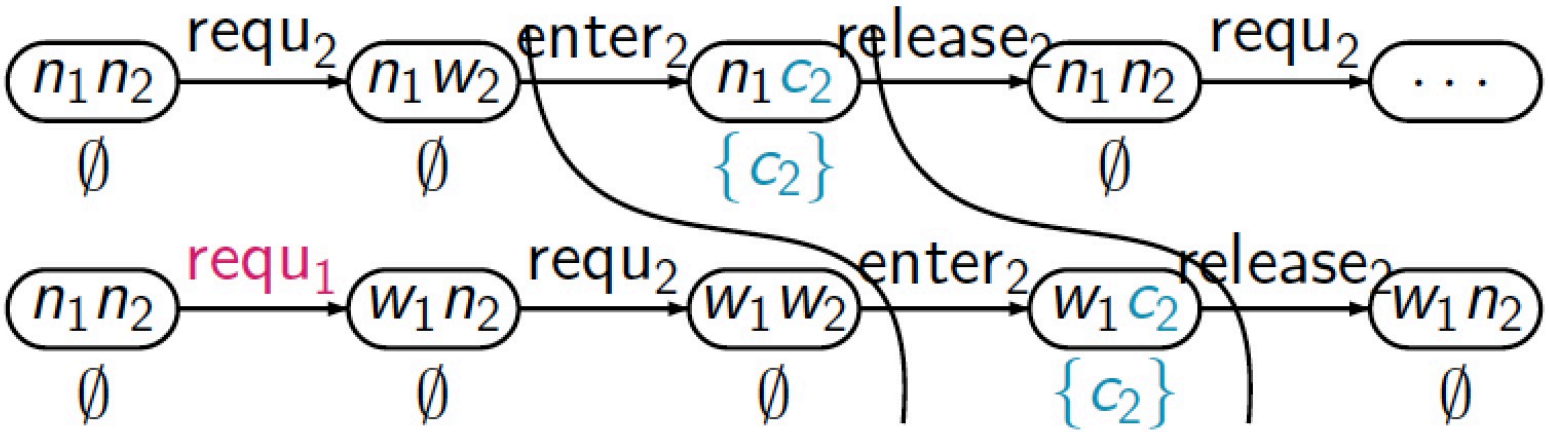
$$\rho' = s \overset{\alpha}{\Longrightarrow} t_0 \xrightarrow{\beta_1} t_1 \xrightarrow{\beta_2} t_2 \xrightarrow{\beta_3} \ldots$$

where $\alpha \in \mathit{ample}(s)$ and $\rho \equiv_{\mathit{sttrace}} \rho'$.

Example: Ample Sets for Semaphore

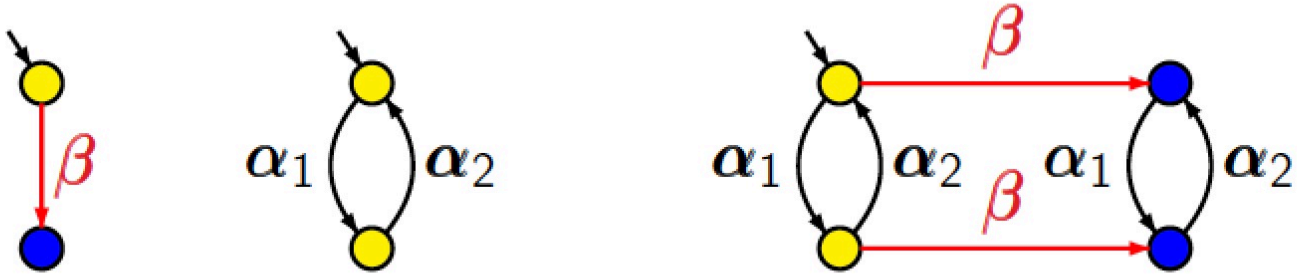


execution where $request_1$ is not executed



The Necessity of Cycle Condition (A4)

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

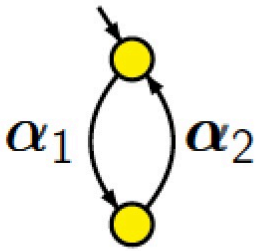


$\mathcal{T} \not\models \Box\neg$blue

β, α_i independent

α_1, α_2 stutter actions

$\mathcal{T}_{\text{red}}$ satisfies (A1), (A2), (A3)



$\mathcal{T}_{\text{red}} \models \Box\neg$blue

Cycle Condition

Cycle condition (A4)

For any cycle $s_0 \ldots s_n$ in TS_{red} and $\underbrace{\alpha \in Act(s_i)}_{\text{in } TS}$, for some $0 < i \leq n$, $\alpha \in ample(s_j)$ for some $j \in \{1, \ldots, n\}$.

Every enabled action in some state on a cycle in TS_{red} must be selected in some state on that cycle.

Ample Set Conditions

(A1) Nonemptiness condition

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

(A2) Dependency condition

Let $s \xrightarrow{\beta_1} \ldots \xrightarrow{\beta_n} s_n \xrightarrow{\alpha} t$ be a finite run in TS such that α depends on $\mathit{ample}(s)$. Then: $\beta_i \in \mathit{ample}(s)$ for some $0 < i \leq n$.

(A3) Stutter condition

If $\mathit{ample}(s) \neq \mathit{Act}(s)$ then any $\alpha \in \mathit{ample}(s)$ is a stutter action.

(A4) Cycle condition

For any cycle $s_0 \ldots s_n$ in TS_{red} and $\alpha \in \mathit{Act}(s_i)$, for some $0 < i \leq n$, $\alpha \in \mathit{ample}(s_j)$ for some $j \in \{1, \ldots, n\}$.

Overview

Correctness

Let TS be a finite, action-deterministic transition system w/o terminal states.

If all ample sets satisfy conditions (A1)–(A4), then $TS_{red} \equiv_{sttrace} TS$.

Complexity Considerations

Let TS be a finite, action-deterministic transition system w/o terminal states.

The worst-case time complexity of checking (A2) in TS equals that of checking $w\,TS' \vDash \exists\Diamond\, a$ for some $a \in AP$ where $size(TS') \in O(size(TS))$.

Proof.

See Appendix □

(A1), (A3) and (A4) can relatively easy be incorporated in a DFS-based state-space generation.

Some Experimental Results

[Clarke, Grumberg, Minea, Peled, 1999]

Algorithm	TS			TS_{red}		
	states	transition	time	states	transitions	time
sieve	10,878	35,594	1.68	157	157	0.08
data transfer protocol	251,049	648,467	32.2	16,459	17,603	1.47
snoopy (cache coherence)	164,258	546,805	33.6	29,796	44,145	3.58
file transfer protocol	514,188	1,138,750	123.4	125,595	191,466	18.6

partial-order reduction works good for
loosely-synchronised multi-threaded systems

Overview

Summary

- POR ignores several interleavings of independent actions
  in an on-the-fly-manner; i.e., during state-space generation
- The ample set method relies on choosing $\mathit{ample}(s) \subseteq \mathit{Act}(s)$ in state s
  actions not in $\mathit{ample}(s)$ are pruned
- (A1) non-emptiness, (A2) dependency, (A3) stutter and (A4) cycle
- Conditions (A1) and (A2) ensure that any run in TS can be turned into an equivalent run in TS_{red} by permuting independent actions (and adding independent actions)
- (A3) and (A4) ensure that these two runs are stutter equivalent
- POR is effective for loosely coupled multi-threaded systems

Next Lecture

Monday July 4, 10:30

Appendix

Algorithmic Difficulty of Checking (A2)

Algorithmic difficulty of checking $(\mathrm{A2})$

unreachability problem	$\leq_{\text{poly}}$	problem of checking $(\mathrm{A2})$

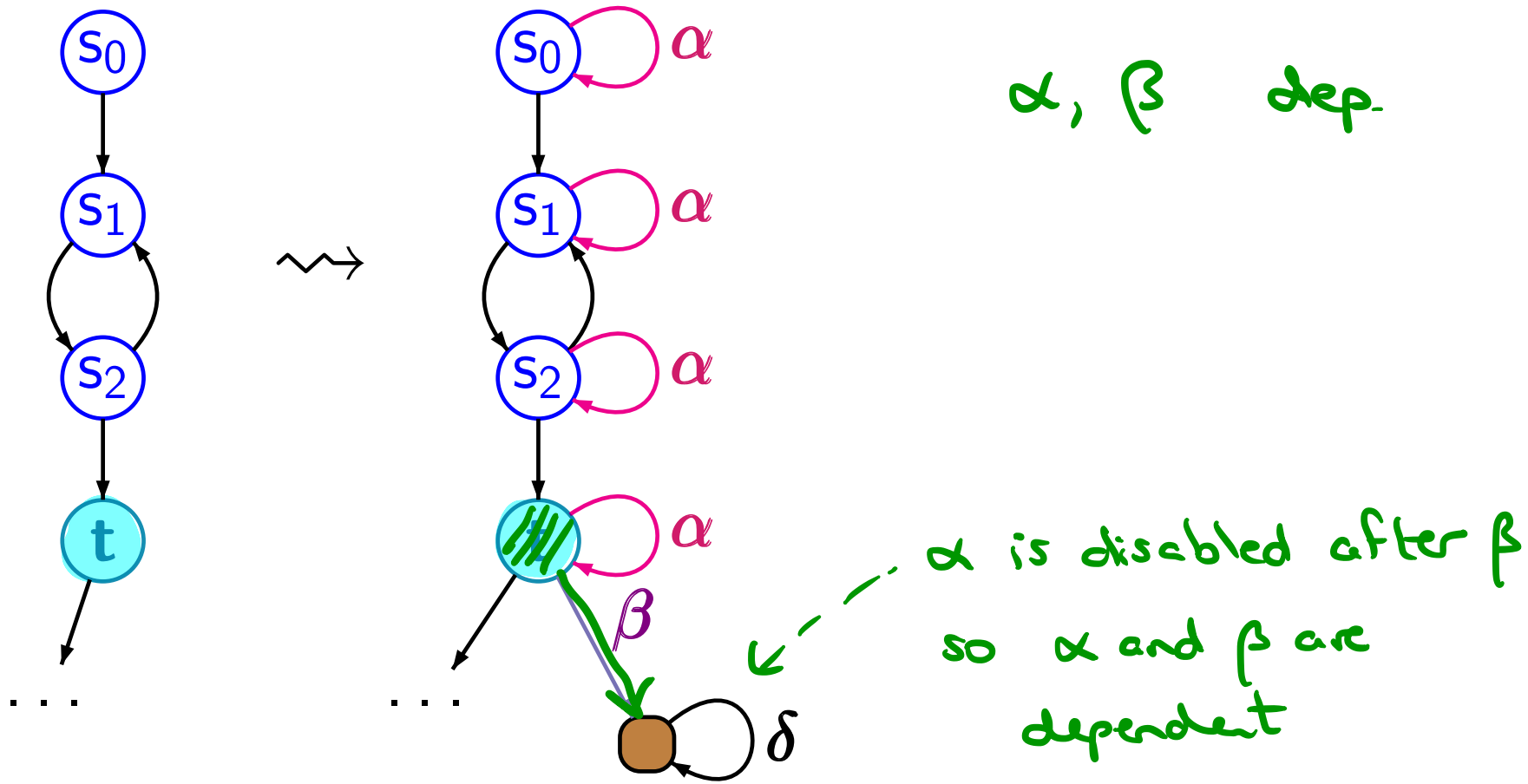
finite TS $\mathcal{T}$ + state s_0 + atomic prop. a $\rightsquigarrow$ finite TS $\mathcal{T}'$ + ample sets

s.t. $s_0 \not\models \exists\Diamond a$ iff $(\mathrm{A2})$ holds

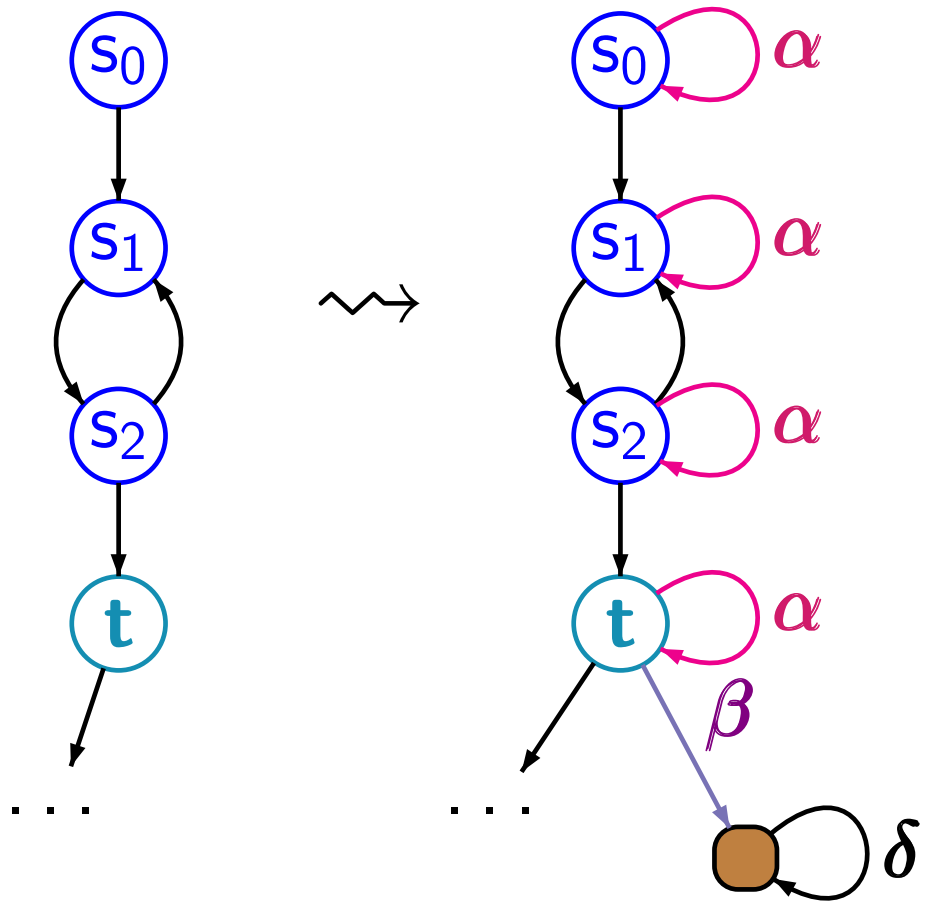
$\mathcal{T}'$ results from $\mathcal{T}$ by adding two fresh actions α, β s.t.

- α are β are dependent
- α is independent from all actions in $\mathcal{T}$
- β is enabled exactly in the states t with $t \models a$

finite TS $\mathcal{T}$ + state s_0 + atomic prop. a		$\rightsquigarrow$	finite TS $\mathcal{T}'$ + ample sets
s.t.	$s_0 \not\models \exists\Diamond a$	iff	(A2) holds



	finite TS $\mathcal{T}$ + state s_0 + atomic prop. **a**	$\rightsquigarrow$	finite TS $\mathcal{T}'$ + ample sets
s.t.	$s_0 \not\models \exists\Diamond \mathbf{a}$	iff	(A2) holds

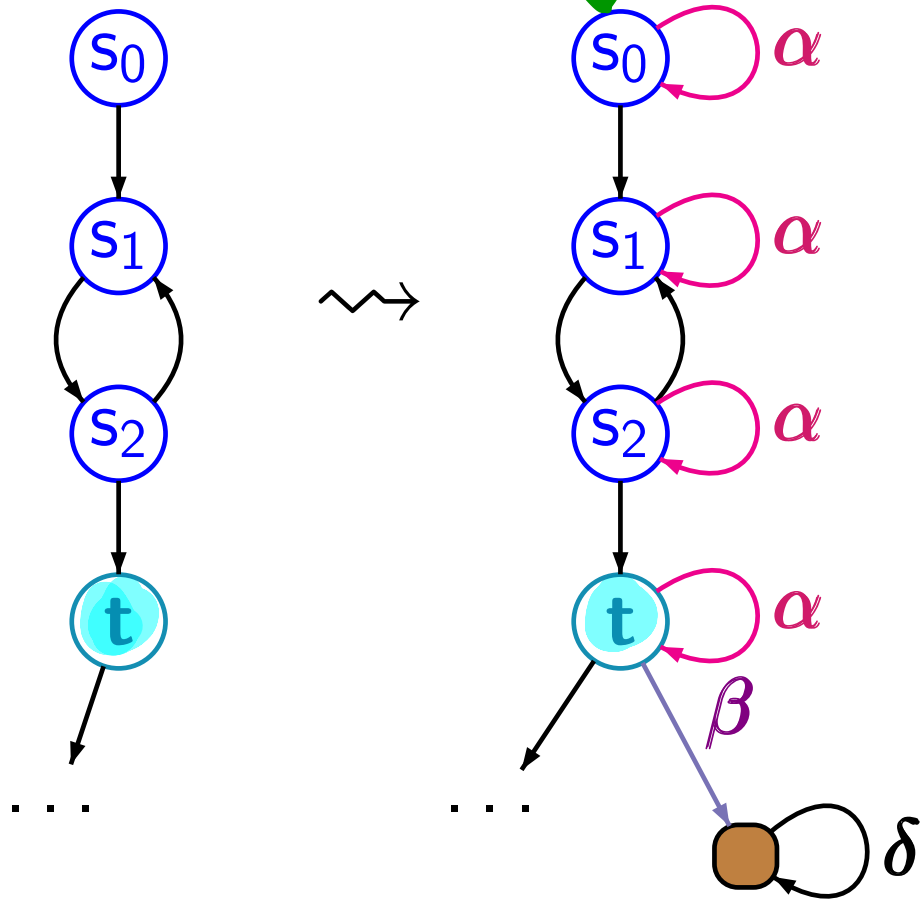


α, β dependent

α independent from all other actions

finite TS $\mathcal{T}$ + state s_0 + atomic prop. a $\rightsquigarrow$ finite TS $\mathcal{T}'$ + ample sets

s.t. $s_0 \not\models \exists\Diamond a$ iff (A2) holds



α, β dependent

α independent from all other actions

$\text{ample}(s_0) = \{\alpha\}$

$\text{ample}(u) = Act(u)$ ✓ for all other states u

- $TS \models \exists\Diamond a$ implies $TS' \not\models$ (A2)

  Let $t \in Reach(TS)$, $t \models a$. Thus

  $$\rho :: s_0 \to \dots \to t \xrightarrow{\beta} trap \in TS'$$

  Since $\alpha \in Act(s_0)$ and α, β dependent in TS' β depends on $ample(s_0) = \{\alpha\}$. Since β is only enabled in t in ρ, and not in any state prior to t, $\rho \not\models$ (A2).

- Assume $TS' \not\models$ (A2). Then for some $v \in Reach(TS')$

  $$v \xrightarrow{t_1} s_1 \xrightarrow{t_2} \dots \xrightarrow{t_n} s_n \xrightarrow{t} s' \in TS'$$

  with t depends on $ample(v)$ and $t_1, \dots, t_n \notin ample(v)$. As s_0 is only "pruned" state $v = s_0$. As $ample(s_0) = \{\alpha\}$ it follows $t = \beta$. Since $t = \beta$ is only enabled in t, $s_n = t$. Thus $TS' \models \exists\Diamond a$. As $v, s_1, \dots, s_n \in TS$, thus $TS \models \exists\Diamond a$. ⊠