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

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

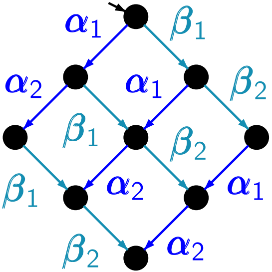
- for asynchronous systems
- analyze representatives of path equivalence classes that represent the same the same behavior up to the interleaving order

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

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

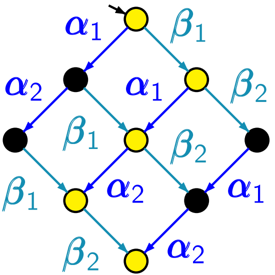
$\mathcal{T} = \mathcal{T}_1 \,|||\, \mathcal{T}_2$



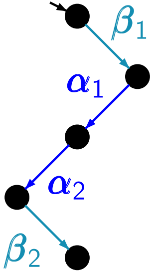
Basic idea of partial order reduction

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

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

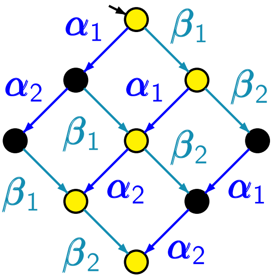


$\mathcal{T}_{\text{red}}$

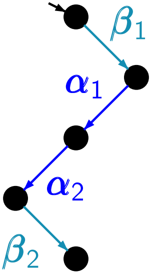


Partial order reduction for $\mathsf{LTL}_{\setminus\bigcirc}$ specifications

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

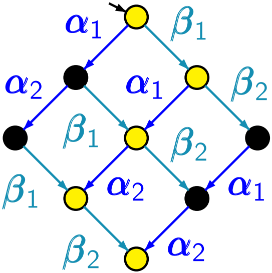


$\mathcal{T}_{\mathsf{red}}$

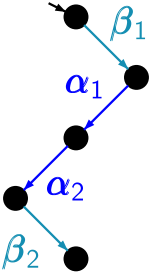


Partial order reduction for $\mathbf{LTL}_{\setminus\bigcirc}$ specifications

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



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

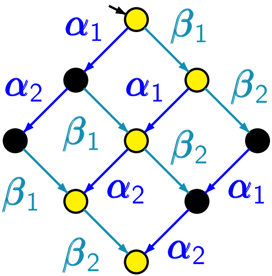


requirement: for all $\mathbf{LTL}_{\setminus\bigcirc}$ formulas φ:

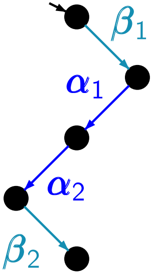
$$\mathcal{T} \models \varphi \text{ iff } \mathcal{T}_{\text{red}} \models \varphi$$

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

The ample set method [Peled '93]

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

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

The ample set method [Peled '93]

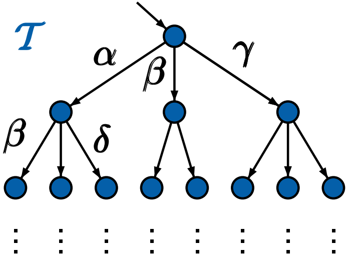
given: syntactical representation of processes of TS $\mathcal{T}$

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

The ample set method [Peled ’93]

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

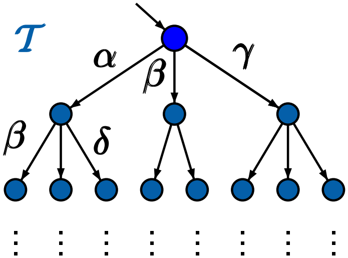
goal: on-the-fly construction of a fragment $\mathcal{T}_{\text{red}}$
by selecting action-sets $\text{ample}(s) \subseteq \text{Act}(s)$
and expanding only the α-successors of s
where $\alpha \in \text{ample}(s)$



The ample set method [Peled '93]

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

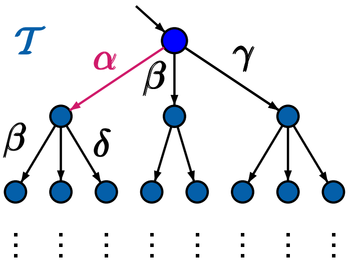
goal: on-the-fly construction of a fragment $\mathcal{T}_{red}$
by selecting action-sets $ample(s) \subseteq Act(s)$
and expanding only the α-successors of s
where $\alpha \in ample(s)$



The ample set method [Peled '93]

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

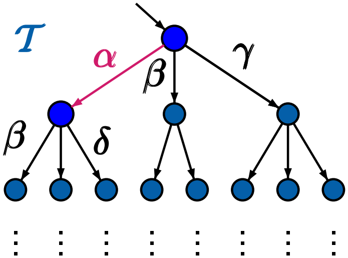
goal: on-the-fly construction of a fragment $\mathcal{T}_{\text{red}}$
by selecting action-sets $\text{ample}(s) \subseteq \text{Act}(s)$
and expanding only the α-successors of s
where $\alpha \in \text{ample}(s)$



The ample set method [Peled '93]

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

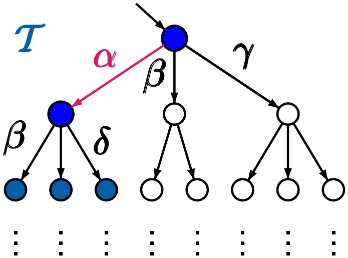
goal: on-the-fly construction of a fragment $\mathcal{T}_{red}$
by selecting action-sets $ample(s) \subseteq Act(s)$
and expanding only the α-successors of s
where $\alpha \in ample(s)$



The ample set method [Peled '93]

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

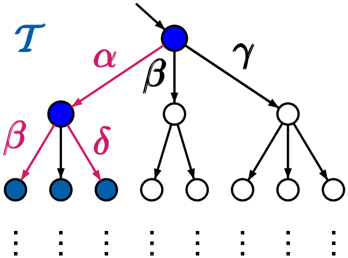
goal: on-the-fly construction of a fragment $\mathcal{T}_{red}$
by selecting action-sets $ample(s) \subseteq Act(s)$
and expanding only the α-successors of s
where $\alpha \in ample(s)$



The ample set method [Peled '93]

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

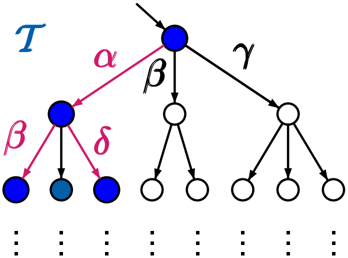
goal: on-the-fly construction of a fragment $\mathcal{T}_{red}$
by selecting action-sets $ample(s) \subseteq Act(s)$
and expanding only the α-successors of s
where $\alpha \in ample(s)$



The ample set method [Peled '93]

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

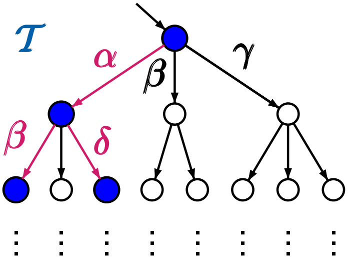
goal: on-the-fly construction of a fragment $\mathcal{T}_{red}$
by selecting action-sets ample(s) $\subseteq$ Act(s)
and expanding only the α-successors of s
where $\alpha \in$ ample(s)



The ample set method [Peled '93]

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

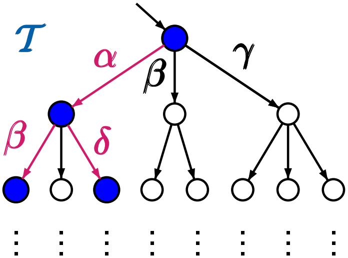
goal: on-the-fly construction of a fragment $\mathcal{T}_{\text{red}}$ by selecting action-sets ample(s) $\subseteq$ Act(s) and expanding only the α-successors of s where $\alpha \in$ ample(s)

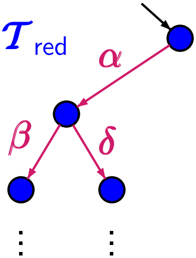


The ample set method [Peled '93]

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

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

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 $\mathcal{T}_{red}$
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 $\mathcal{T}$
- efficient construction of $\mathcal{T}_{red}$ 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:

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

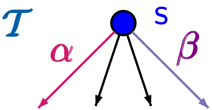
The reduced transition system $\mathcal{T}_{\text{red}}$

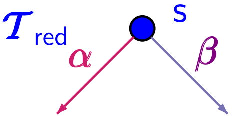
is a fragment of $\mathcal{T}$ that results from $\mathcal{T}$ by

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

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

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





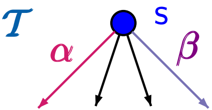
$ample(s) = \{\alpha, \beta\}$

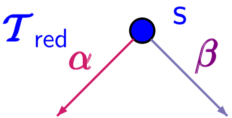
The reduced transition system $\mathcal{T}_{\text{red}}$

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

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

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





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

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

Action-determinism

LTL3.4-11A

Action-determinism

LTL3.4-11A

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

Action-determinism

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

For state $\mathbf{s}$:

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

Action-determinism

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

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 $\alpha \in Act(s)$:

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

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

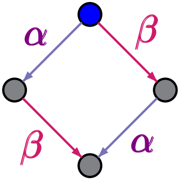
Let $\mathcal{T}$ be an action-deterministic transition system with action-set Act, and $\alpha, \beta \in Act$.

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. $\alpha, \beta \in Act(s)$:

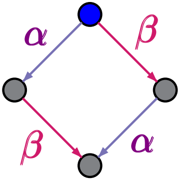


Independence of actions

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

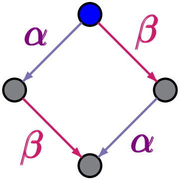


Independence of actions

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

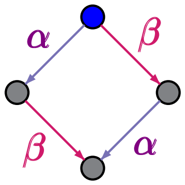


Independence of actions

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

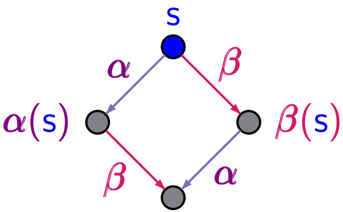


Independence of actions

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



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

Conditions for ample sets

(A1) nonemptiness condition

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

Conditions for ample sets

(A1) nonemptiness condition

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

(A2) dependency condition

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 ample(s)
there is some $i < n$ with

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

(A1) nonemptiness condition

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

(A2) dependency condition

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

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

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

Conditions for ample sets

(A1) nonemptiness condition

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

(A2) dependency condition

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

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

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

(A3) stutter condition

Conditions for ample sets

(A1) nonemptiness condition

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

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

(A3) stutter condition

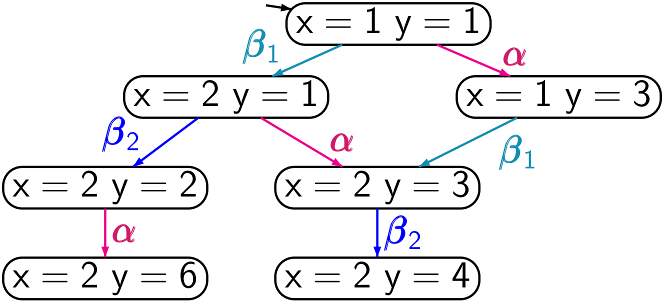
if $\mathsf{ample}(s) \neq \mathit{Act}(s)$ then all actions in ample(s) are stutter actions

Example

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

Example

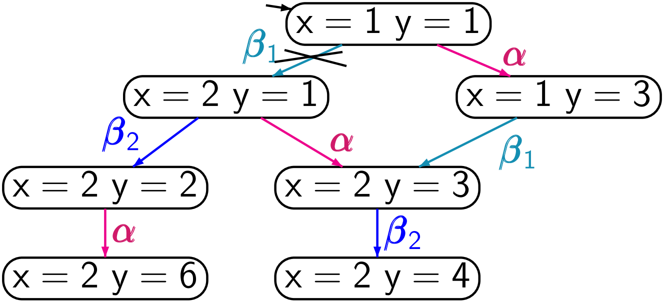
$$\underbrace{x := 2 \cdot x}_{\beta_1} \; ; \; \underbrace{y := y + 1}_{\beta_2} \;\; ||| \;\; \underbrace{y := 3 \cdot y}_{\alpha}$$



Example

LTL3.4-23

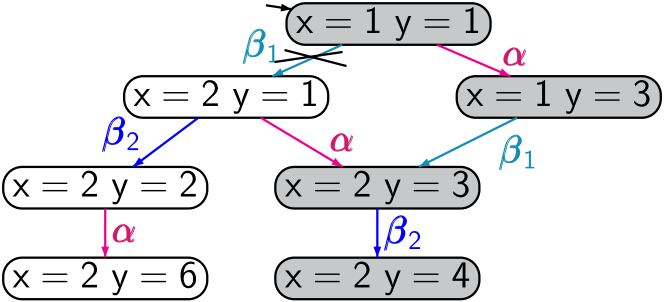
$$\underbrace{x := 2 \cdot x}_{\beta_1} \; ; \; \underbrace{y := y + 1}_{\beta_2} \;\; ||| \;\; \underbrace{y := 3 \cdot y}_{\alpha}$$



Example

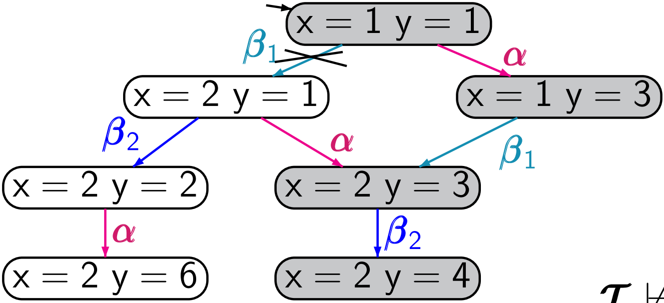
LTL3.4-23

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



Example

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

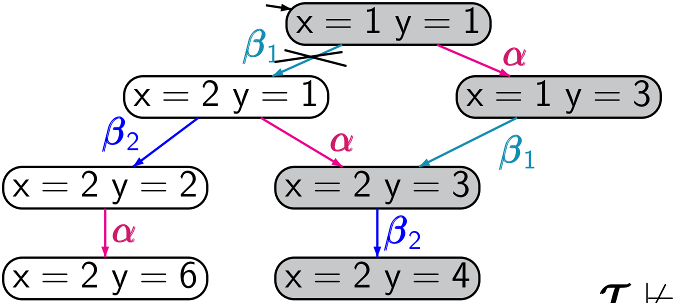


$$\mathcal{T} \not\models \Box(y \neq 6)$$

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

Example

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



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

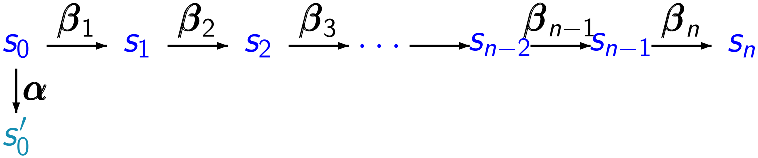
$$\mathcal{T}_{\text{red}} \models \Box(y \neq 6)$$

(A2) violated as β_2, α dependent

Conditions (A2) and (A3)

Suppose

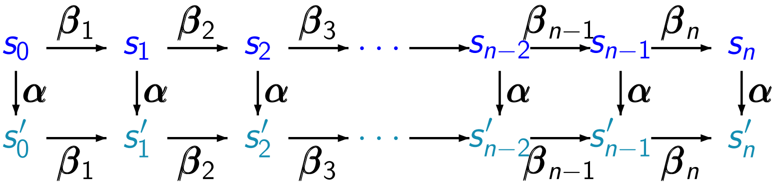
- $\alpha \in \text{ample}(s_0)$, $\beta_i \notin \text{ample}(s_0)$
- α stutter action



Conditions (A2) and (A3)

Suppose

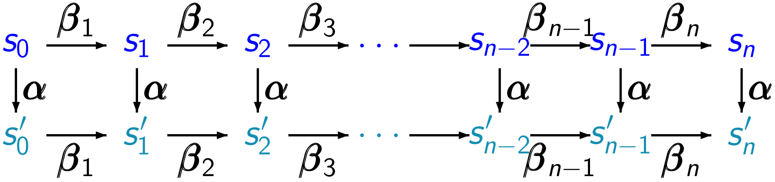
- $\alpha \in ample(s_0)$, $\beta_i \notin ample(s_0)$
- α stutter action



Conditions (A2) and (A3)

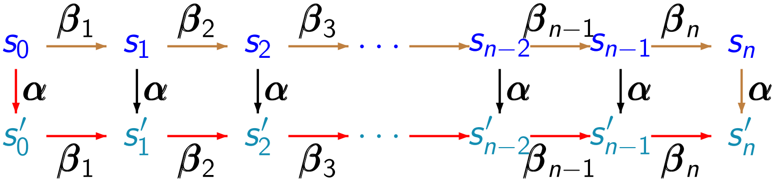
Suppose

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

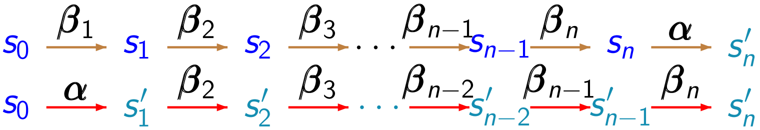


Conditions (A2) and (A3)

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

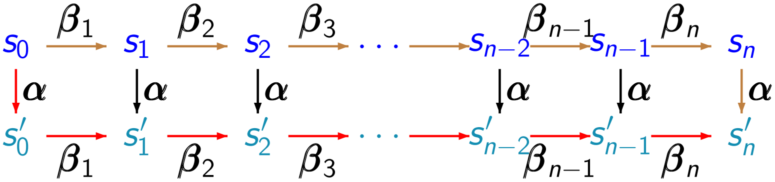


case 1:

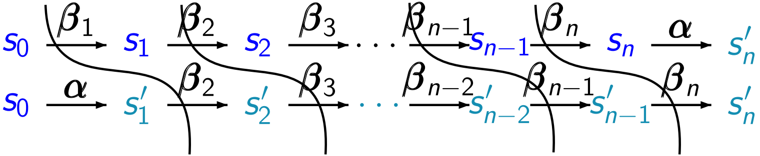


Conditions (A2) and (A3)

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

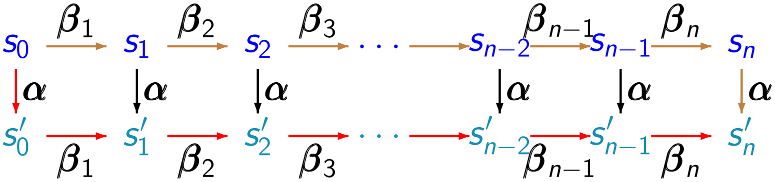


case 1:

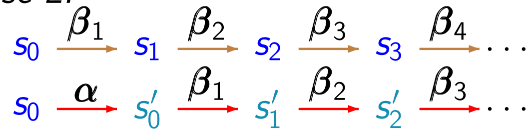


Conditions (A2) and (A3)

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

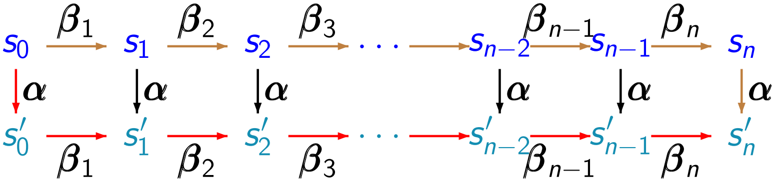


case 2:

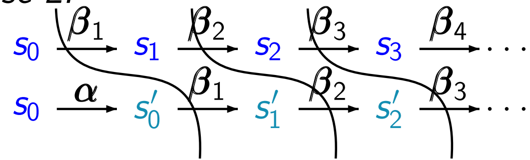


Conditions (A2) and (A3)

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



case 2:



Conditions (A1), (A2), (A3) are not sufficient

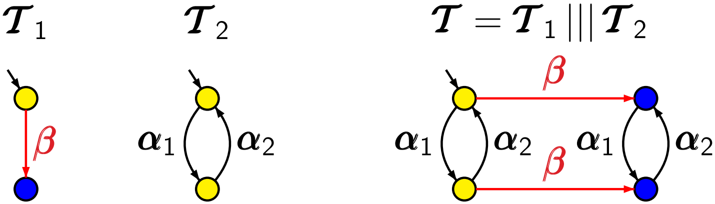
LTL3.4-30

Conditions (A1), (A2), (A3) are not sufficient

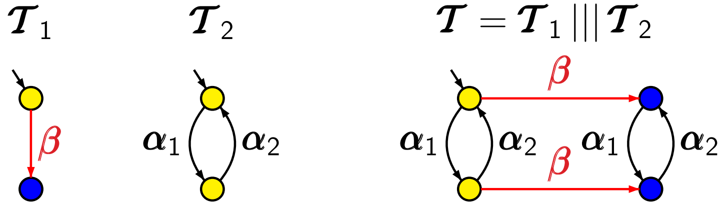
There exists a finite, action-deterministic transition system $\mathcal{T}$ and ample sets for $\mathcal{T}$ such that

$$\mathcal{T} \not\triangleq \mathcal{T}_{\text{red}}$$

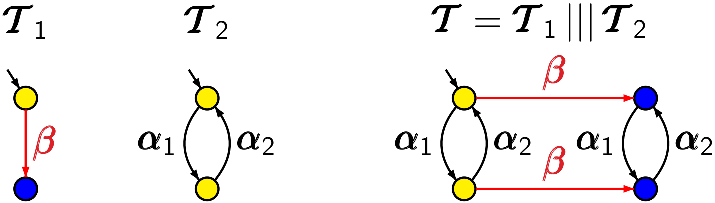
remind: $\triangleq$ stutter trace equivalence



$\mathcal{T}_1$
$\mathcal{T}_2$
$\mathcal{T} = \mathcal{T}_1 \,|||\, \mathcal{T}_2$
β
α_1
α_2
β
α_1
α_2
β
α_1
α_2

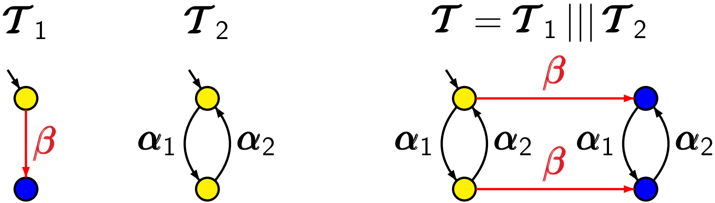


β, α_i independent

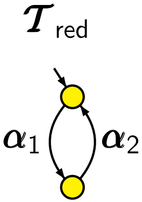


β, α_i independent

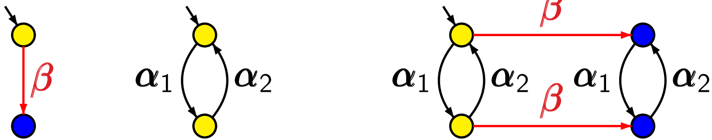
α_1, α_2 stutter actions



β, α_i independent
α_1, α_2 stutter actions



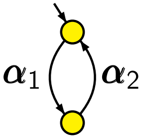
$\mathcal{T}_1$ $\qquad$ $\mathcal{T}_2$ $\qquad$ $\mathcal{T} = \mathcal{T}_1 \,|||\, \mathcal{T}_2$



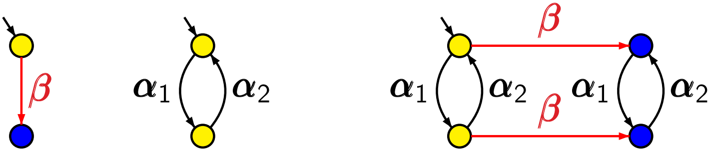
$\boldsymbol{\beta}, \boldsymbol{\alpha}_i$ independent

$\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2$ stutter actions

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



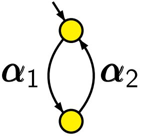
$\mathcal{T}_1$ $\quad$ $\mathcal{T}_2$ $\quad$ $\mathcal{T} = \mathcal{T}_1 \, ||| \, \mathcal{T}_2$



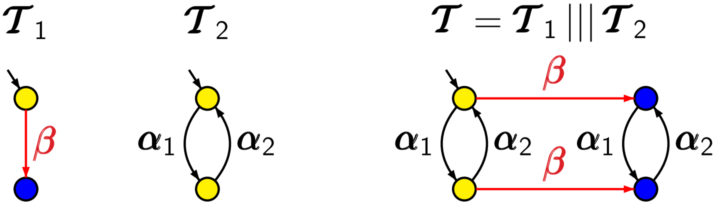
$\mathcal{T} \not\models \Box \neg \text{blue}$

β, α_i independent
α_1, α_2 stutter actions

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



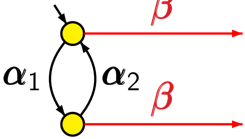
$\mathcal{T}_{\text{red}} \models \Box \neg \text{blue}$



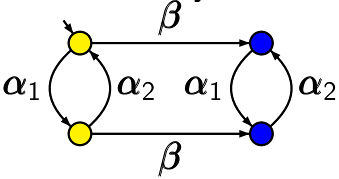
$\mathcal{T} \not\models \Box\neg$blue

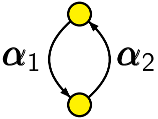
β, α_i independent
α_1, α_2 stutter actions

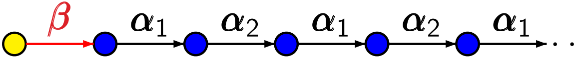
$\mathcal{T}_{\text{red}}$ satisfies (A1), (A2), (A3)

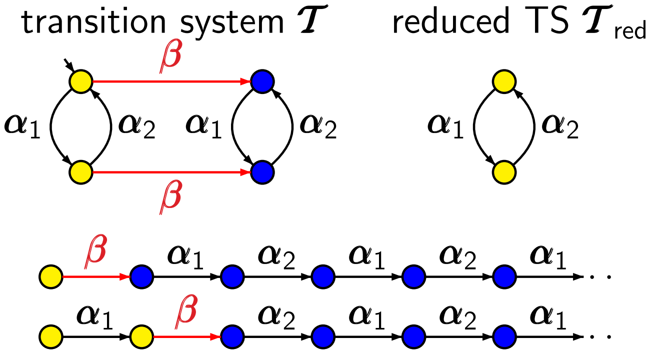


$\mathcal{T}_{\text{red}} \models \Box\neg$blue transition system $\mathcal{T}$ reduced TS $\mathcal{T}_{\text{red}}$



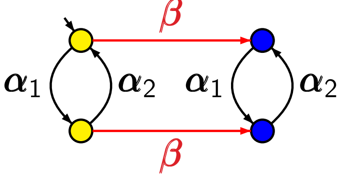




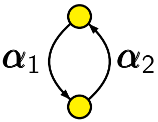


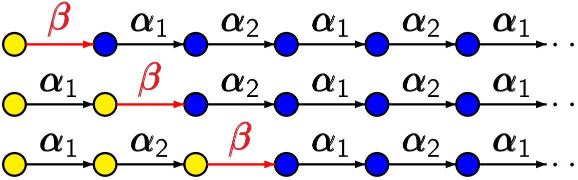
transition system $\mathcal{T}$
reduced TS $\mathcal{T}_{\text{red}}$
β
α_1
α_2
α_1
α_2
β
α_1
α_2
β α_1 α_2 α_1 α_2 α_1 ...
α_1 β α_2 α_1 α_2 α_1 ...

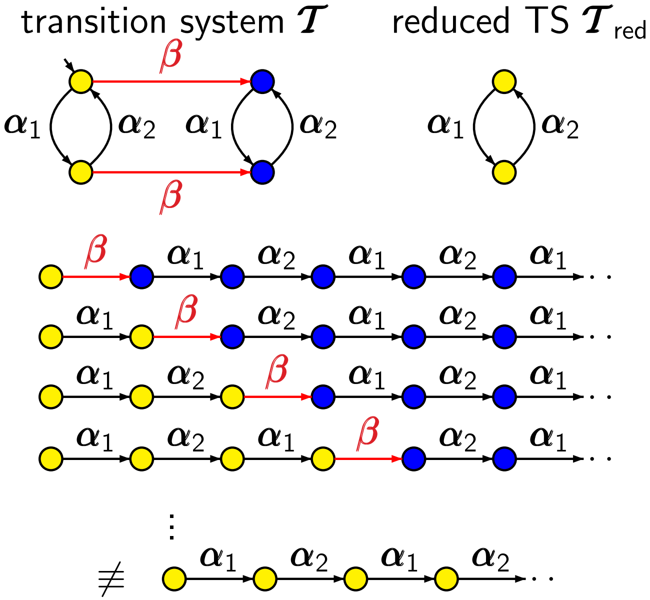
transition system $\mathcal{T}$



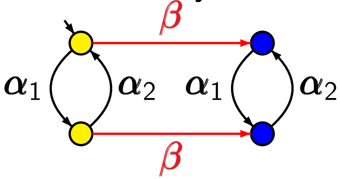
reduced TS $\mathcal{T}_{red}$

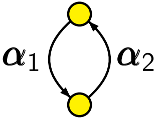


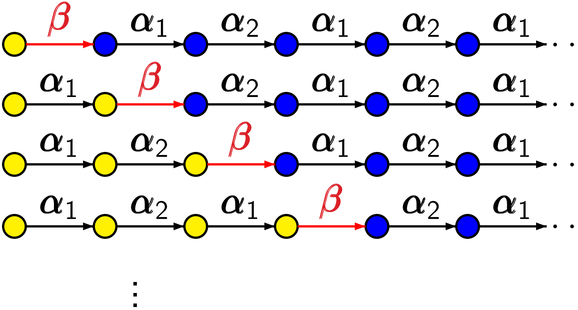



transition system $\mathcal{T}$
reduced TS $\mathcal{T}_{red}$
β
α_1
α_2
$\neq$

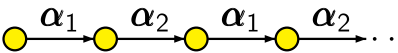
transition system $\mathcal{T}$ reduced TS $\mathcal{T}_{\text{red}}$







$\neq$



$=$ the unique execution of $\mathcal{T}_{\text{red}}$

4 conditions for ample sets LTL3.4-A4

(A1) $\emptyset \neq \mathsf{ample}(s) \subseteq Act(s)$

4 conditions for ample sets LTL3.4-A4

(A1) $\emptyset \neq \mathit{ample}(s) \subseteq \mathit{Act}(s)$

(A2) 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 $\mathit{ample}(s)$ there is some $i < n$ with $\beta_i \in \mathit{ample}(s)$

4 conditions for ample sets

(A1) $\emptyset \neq \mathsf{ample}(\mathsf{s}) \subseteq \mathit{Act}(\mathsf{s})$

(A2) for each execution fragment in $\mathcal{T}$

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

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

(A3) if $\mathsf{ample}(\mathsf{s}) \neq \mathit{Act}(\mathsf{s})$ then all actions in $\mathsf{ample}(\mathsf{s})$ are *stutter actions*

4 conditions for ample sets LTL3.4-A4

(A1) $\emptyset \neq \mathsf{ample}(\mathsf{s}) \subseteq Act(\mathsf{s})$

(A2) for each execution fragment in $\mathcal{T}$

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

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

(A3) if $\mathsf{ample}(\mathsf{s}) \neq Act(\mathsf{s})$ then all actions in $\mathsf{ample}(\mathsf{s})$ are *stutter actions*

(A4) cycle condition

4 conditions for ample sets

(A1) $\emptyset \neq \mathsf{ample}(s) \subseteq Act(s)$

(A2) 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) if $\mathsf{ample}(s) \neq Act(s)$ then all actions in $\mathsf{ample}(s)$ are *stutter actions*

(A4) for each *cycle* $s_0 \Rightarrow s_1 \Rightarrow \ldots \Rightarrow s_n$ in $\mathcal{T}_{red}$ and each action

$$\beta \in \bigcup_{0 \leq i < n} Act(s_i)$$

there is some $i \in \{1, \ldots, n\}$ with $\beta \in \mathsf{ample}(s_i)$

4 conditions for ample sets

(A1) $\emptyset \neq \mathsf{ample}(\mathsf{s}) \subseteq \mathit{Act}(\mathsf{s})$

(A2) for each execution fragment in $\mathcal{T}$

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

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

(A3) if $\mathsf{ample}(\mathsf{s}) \neq \mathit{Act}(\mathsf{s})$ then all actions in $\mathsf{ample}(\mathsf{s})$ are *stutter actions*

(A4) for each *cycle* $\mathsf{s}_0 \Rightarrow \mathsf{s}_1 \Rightarrow \ldots \Rightarrow \mathsf{s}_\mathsf{n}$ in $\mathcal{T}_{\mathsf{red}}$ and each action

$$\beta \in \bigcup_{0 \leq \mathsf{i} < \mathsf{n}} \mathit{Act}(\mathsf{s}_\mathsf{i})$$

there is some $\mathsf{i} \in \{1, \ldots, \mathsf{n}\}$ with $\beta \in \mathsf{ample}(\mathsf{s}_\mathsf{i})$

4 conditions for ample sets

(A1) $\emptyset \neq ample(s) \subseteq Act(s)$

(A2) 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 $\beta_i \in ample(s)$

(A3) if $ample(s) \neq Act(s)$ then all actions in $ample(s)$ are *stutter actions*

(A4) for each *cycle* $s_0 \Rightarrow s_1 \Rightarrow \ldots \Rightarrow s_n$ in $\mathcal{T}_{red}$ and each action

$$\beta \in \bigcup_{0 \leq i < n} Act(s_i)$$

there is some $i \in \{1, \ldots, n\}$ with $\beta \in ample(s_i)$

Soundness of conditions (A1), (A2), (A3), (A4)

LTL3.4-35

Soundness of conditions (A1), (A2), (A3), (A4)

Let $\mathcal{T}$ be a finite, action-deterministic transition system.

Let $\mathcal{T}$ be a finite, action-deterministic transition system.

If the ample sets ample(s) satisfy conditions (A1), (A2), (A3), (A4) then

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

remind: $\triangleq$ stutter trace equivalence

Soundness of conditions (A1), (A2), (A3), (A4)

Let $\mathcal{T}$ be a finite, action-deterministic transition system.

If the ample sets ample(s) satisfy conditions (A1), (A2), (A3), (A4) then

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

hence: for all $\mathbf{LTL}_{\setminus \bigcirc}$ formulas φ:

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

Soundness of conditions (A1), (A2), (A3), (A4)

Let $\mathcal{T}$ be a finite, action-deterministic transition system. If the ample sets ample(s) satisfy conditions (A1), (A2), (A3), (A4) then

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

Proof: show that

$$\mathcal{T} \trianglelefteq \mathcal{T}_{\text{red}} \quad \text{and} \quad \mathcal{T}_{\text{red}} \trianglelefteq \mathcal{T}$$

where $\trianglelefteq$ = stutter trace inclusion

Soundness of conditions (A1), (A2), (A3), (A4)

Let $\mathcal{T}$ be a finite, action-deterministic transition system. If the ample sets ample(s) satisfy conditions (A1), (A2), (A3), (A4) then

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

Proof:

- $\mathcal{T}_{\text{red}} \trianglelefteq \mathcal{T}$: $\surd$

Soundness of conditions $(A1)$, $(A2)$, $(A3)$, $(A4)$

Let $\mathcal{T}$ be a finite, action-deterministic transition system. If the ample sets ample(s) satisfy conditions $(A1)$, $(A2)$, $(A3)$, $(A4)$ then

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

Proof:

- $\mathcal{T}_{\text{red}} \trianglelefteq \mathcal{T}$: $\surd$
- $\mathcal{T} \trianglelefteq \mathcal{T}_{\text{red}}$:
  show that each execution ρ of $\mathcal{T}$ can be transformed into a stutter equivalent execution ρ' of $\mathcal{T}_{\text{red}}$

Proof of $\mathcal{T} \trianglelefteq \mathcal{T}_{\textsf{red}}$

given: infinite execution fragment ρ of $\mathcal{T}$

goal: construction of a stutter equivalent execution fragment ρ' of $\mathcal{T}_{\textsf{red}}$

Proof of $\mathcal{T} \trianglelefteq \mathcal{T}_{\text{red}}$

given: infinite execution fragment ρ of $\mathcal{T}$

goal: construction of a stutter equivalent execution fragment ρ' of $\mathcal{T}_{\text{red}}$

idea: ρ' results from the "limit" of transformations

$$\rho = \rho_0 \leadsto \rho_1 \leadsto \rho_2 \leadsto \rho_3 \leadsto$$

Proof of $\mathcal{T} \trianglelefteq \mathcal{T}_{\text{red}}$

given: infinite execution fragment ρ of $\mathcal{T}$

goal: construction of a stutter equivalent execution fragment ρ' of $\mathcal{T}_{\text{red}}$

idea: ρ' results from the "limit" of transformations

$$\rho = \rho_0 \rightsquigarrow \rho_1 \rightsquigarrow \rho_2 \rightsquigarrow \rho_3 \rightsquigarrow$$

where, for $i > j \geq 0$, the execution fragments ρ_i and ρ_j have a common prefix

- of length j
- consisting of transitions in $\mathcal{T}_{\text{red}}$

Stepwise transformation $\rho_0 \rightsquigarrow \rho_1$

LTL3.4-35A

Stepwise transformation $\rho_0 \leadsto \rho_1$

case 0: $\rho_0 = \quad s_0 \xrightarrow{\alpha} \xrightarrow{\beta_1} \xrightarrow{\beta_2} \xrightarrow{\beta_3} \ldots$ with $\alpha \in \mathit{ample}(s_0)$

Stepwise transformation $\rho_0 \rightsquigarrow \rho_1$

case 0: $\rho_0 = \quad s_0 \xrightarrow{\alpha} \xrightarrow{\beta_1} \xrightarrow{\beta_2} \xrightarrow{\beta_3} \ldots$ with $\alpha \in \text{ample}(s_0)$

$\rho_1 = \quad s_0 \xrightarrow{\alpha} \xrightarrow{\beta_1} \xrightarrow{\beta_2} \xrightarrow{\beta_3} \ldots = \rho_0$

Stepwise transformation $\rho_0 \rightsquigarrow \rho_1$

case 0: $\rho_0 = \quad s_0 \xrightarrow{\alpha} \xrightarrow{\beta_1} \xrightarrow{\beta_2} \xrightarrow{\beta_3} \ldots$ with $\alpha \in \text{ample}(s_0)$

$\rho_1 = \quad s_0 \xrightarrow{\alpha} \xrightarrow{\beta_1} \xrightarrow{\beta_2} \xrightarrow{\beta_3} \ldots = \rho_0$

case 1: $\rho_0 = \quad s_0 \xrightarrow{\beta_1} \xrightarrow{\beta_2} \ldots \xrightarrow{\beta_{n-1}} \xrightarrow{\alpha} \xrightarrow{\beta_{n+1}} \xrightarrow{\beta_{n+2}} \ldots$

where $\beta_1, \ldots, \beta_{n-1} \notin \text{ample}(s_0)$, $\alpha \in \text{ample}(s_0)$

Stepwise transformation $\rho_0 \rightsquigarrow \rho_1$

case 0: $\rho_0 = \quad s_0 \xrightarrow{\alpha} \xrightarrow{\beta_1} \xrightarrow{\beta_2} \xrightarrow{\beta_3} \ldots$ with $\alpha \in \text{ample}(s_0)$

$\rho_1 = \quad s_0 \xrightarrow{\alpha} \xrightarrow{\beta_1} \xrightarrow{\beta_2} \xrightarrow{\beta_3} \ldots = \rho_0$

case 1: $\rho_0 = \quad s_0 \xrightarrow{\beta_1} \xrightarrow{\beta_2} \ldots \xrightarrow{\beta_{n-1}} \xrightarrow{\alpha} \xrightarrow{\beta_{n+1}} \xrightarrow{\beta_{n+2}} \ldots$

where $\beta_1, \ldots, \beta_{n-1} \notin \text{ample}(s_0)$, $\alpha \in \text{ample}(s_0)$

$\rho_1 = \quad s_0 \xrightarrow{\alpha} \xrightarrow{\beta_1} \ldots \xrightarrow{\beta_{n-2}} \xrightarrow{\beta_{n-1}} \xrightarrow{\beta_{n+1}} \xrightarrow{\beta_{n+2}} \ldots$

Stepwise transformation $\rho_0 \rightsquigarrow \rho_1$

case 0: $\rho_0 = \quad s_0 \xrightarrow{\alpha} \xrightarrow{\beta_1} \xrightarrow{\beta_2} \xrightarrow{\beta_3} \ldots$ with $\alpha \in \text{ample}(s_0)$

$\rho_1 = \quad s_0 \xrightarrow{\alpha} \xrightarrow{\beta_1} \xrightarrow{\beta_2} \xrightarrow{\beta_3} \ldots = \rho_0$

case 1: $\rho_0 = \quad s_0 \xrightarrow{\beta_1} \xrightarrow{\beta_2} \ldots \xrightarrow{\beta_{n-1}} \xrightarrow{\alpha} \xrightarrow{\beta_{n+1}} \xrightarrow{\beta_{n+2}} \ldots$

where $\beta_1, \ldots, \beta_{n-1} \notin \text{ample}(s_0)$, $\alpha \in \text{ample}(s_0)$

$\rho_1 = \quad s_0 \xrightarrow{\alpha} \xrightarrow{\beta_1} \ldots \xrightarrow{\beta_{n-2}} \xrightarrow{\beta_{n-1}} \xrightarrow{\beta_{n+1}} \xrightarrow{\beta_{n+2}} \ldots$

case 2: $\rho_0 = \quad s_0 \xrightarrow{\beta_1} \xrightarrow{\beta_2} \xrightarrow{\beta_3} \ldots$

where $\beta_i \notin \text{ample}(s_0)$, $i = 1, 2, \ldots$

Stepwise transformation $\rho_0 \rightsquigarrow \rho_1$

case 0: $\rho_0 = \quad s_0 \xrightarrow{\alpha} \xrightarrow{\beta_1} \xrightarrow{\beta_2} \xrightarrow{\beta_3} \ldots$ with $\alpha \in \text{ample}(s_0)$

$\rho_1 = \quad s_0 \xrightarrow{\alpha} \xrightarrow{\beta_1} \xrightarrow{\beta_2} \xrightarrow{\beta_3} \ldots = \rho_0$

case 1: $\rho_0 = \quad s_0 \xrightarrow{\beta_1} \xrightarrow{\beta_2} \ldots \xrightarrow{\beta_{n-1}} \xrightarrow{\alpha} \xrightarrow{\beta_{n+1}} \xrightarrow{\beta_{n+2}} \ldots$

where $\beta_1, \ldots, \beta_{n-1} \notin \text{ample}(s_0)$, $\alpha \in \text{ample}(s_0)$

$\rho_1 = \quad s_0 \xrightarrow{\alpha} \xrightarrow{\beta_1} \ldots \xrightarrow{\beta_{n-2}} \xrightarrow{\beta_{n-1}} \xrightarrow{\beta_{n+1}} \xrightarrow{\beta_{n+2}} \ldots$

case 2: $\rho_0 = \quad s_0 \xrightarrow{\beta_1} \xrightarrow{\beta_2} \xrightarrow{\beta_3} \ldots$

where $\beta_i \notin \text{ample}(s_0)$, $i = 1, 2, \ldots$

$\rho_1 = \quad s_0 \xrightarrow{\alpha} \xrightarrow{\beta_1} \xrightarrow{\beta_2} \xrightarrow{\beta_3} \ldots$

Stepwise transformation $\rho_0 \rightsquigarrow \rho_1$

case 0: $\rho_0 = \quad s_0 \xrightarrow{\alpha} \xrightarrow{\beta_1} \xrightarrow{\beta_2} \xrightarrow{\beta_3} \ldots$ with $\alpha \in$ ample(s_0)

$\rho_1 = \quad s_0 \xrightarrow{\alpha} \xrightarrow{\beta_1} \xrightarrow{\beta_2} \xrightarrow{\beta_3} \ldots = \rho_0$

case 1: $\rho_0 = \quad s_0 \xrightarrow{\beta_1} \xrightarrow{\beta_2} \ldots \xrightarrow{\beta_{n-1}} \xrightarrow{\alpha} \xrightarrow{\beta_{n+1}} \xrightarrow{\beta_{n+2}} \ldots$

where $\beta_1, \ldots, \beta_{n-1} \notin$ ample(s_0), $\alpha \in$ ample(s_0)

$\rho_1 = \quad s_0 \xrightarrow{\alpha} \xrightarrow{\beta_1} \ldots \xrightarrow{\beta_{n-2}} \xrightarrow{\beta_{n-1}} \xrightarrow{\beta_{n+1}} \xrightarrow{\beta_{n+2}} \ldots$

case 2: $\rho_0 = \quad s_0 \xrightarrow{\beta_1} \xrightarrow{\beta_2} \xrightarrow{\beta_3} \ldots$

where $\beta_i \notin$ ample(s_0), $i = 1, 2, \ldots$

$\rho_1 = \quad s_0 \xrightarrow{\alpha} \xrightarrow{\beta_1} \xrightarrow{\beta_2} \xrightarrow{\beta_3} \ldots$

by (A3): α is a stutter action in cases 1 and 2

Stepwise transformation $\rho_0 \rightsquigarrow \rho_1$

case 0: $\rho_0 = \quad s_0 \xrightarrow{\alpha} \xrightarrow{\beta_1} \xrightarrow{\beta_2} \xrightarrow{\beta_3} \ldots$ with $\alpha \in \mathit{ample}(s_0)$

$\rho_1 = \quad s_0 \xrightarrow{\alpha} \xrightarrow{\beta_1} \xrightarrow{\beta_2} \xrightarrow{\beta_3} \ldots = \rho_0$

case 1: $\rho_0 = \quad s_0 \xrightarrow{\beta_1} \xrightarrow{\beta_2} \ldots \xrightarrow{\beta_{n-1}} \xrightarrow{\alpha} \xrightarrow{\beta_{n+1}} \xrightarrow{\beta_{n+2}} \ldots$

where $\beta_1, \ldots, \beta_{n-1} \notin \mathit{ample}(s_0)$, $\alpha \in \mathit{ample}(s_0)$

$\rho_1 = \quad s_0 \xrightarrow{\alpha} \xrightarrow{\beta_1} \ldots \xrightarrow{\beta_{n-2}} \xrightarrow{\beta_{n-1}} \xrightarrow{\beta_{n+1}} \xrightarrow{\beta_{n+2}} \ldots \triangleq \rho_0$

case 2: $\rho_0 = \quad s_0 \xrightarrow{\beta_1} \xrightarrow{\beta_2} \xrightarrow{\beta_3} \ldots$

where $\beta_i \notin \mathit{ample}(s_0)$, $i = 1, 2, \ldots$

$\rho_1 = \quad s_0 \xrightarrow{\alpha} \xrightarrow{\beta_1} \xrightarrow{\beta_2} \xrightarrow{\beta_3} \ldots \triangleq \rho_0$

by (A3): α is a stutter action in cases 1 and 2

Stepwise transformation $\rho_0 \rightsquigarrow \rho_1$

case 0: $\rho_0 = \quad s_0 \xrightarrow{\alpha} \xrightarrow{\beta_1} \xrightarrow{\beta_2} \xrightarrow{\beta_3} \ldots$ with $\alpha \in \text{ample}(s_0)$

$\rho_1 = \quad s_0 \xrightarrow{\alpha} \xrightarrow{\beta_1} \xrightarrow{\beta_2} \xrightarrow{\beta_3} \ldots = \rho_0$

case 1: $\rho_0 = \quad s_0 \xrightarrow{\beta_1} \xrightarrow{\beta_2} \ldots \xrightarrow{\beta_{n-1}} \xrightarrow{\alpha} \xrightarrow{\beta_{n+1}} \xrightarrow{\beta_{n+2}} \ldots$

where $\beta_1, \ldots, \beta_{n-1} \notin \text{ample}(s_0)$, $\alpha \in \text{ample}(s_0)$

$\rho_1 = \quad s_0 \xrightarrow{\alpha} \xrightarrow{\beta_1} \ldots \xrightarrow{\beta_{n-2}} \xrightarrow{\beta_{n-1}} \xrightarrow{\beta_{n+1}} \xrightarrow{\beta_{n+2}} \ldots \triangleq \rho_0$

case 2: $\rho_0 = \quad s_0 \xrightarrow{\beta_1} \xrightarrow{\beta_2} \xrightarrow{\beta_3} \ldots$

where $\beta_i \notin \text{ample}(s_0)$, $i = 1, 2, \ldots$

$\rho_1 = \quad s_0 \xrightarrow{\alpha} \xrightarrow{\beta_1} \xrightarrow{\beta_2} \xrightarrow{\beta_3} \ldots \triangleq \rho_0$

$\rho_1 \rightsquigarrow \rho_2$:

repeat the same procedure from the 2nd state on

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

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

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. $\rho \triangleq \rho_{\text{red}}$

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. $\rho \overset{\Delta}{=} \rho_{\text{red}}$

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:

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. $\rho \overset{\Delta}{=} \rho_{\text{red}}$

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 $\beta_i \notin ample(s_0)$

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

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. $\rho \overset{\Delta}{=} \rho_{red}$

case 0:	$\rho =$	$s_0 \xrightarrow{\alpha} s_0' \longrightarrow \ldots$ with $\alpha \in ample(s_0)$
		$s_0 \xRightarrow{\alpha} s_0' \longrightarrow \ldots$
case 1:	$\rho =$	$s_0 \xrightarrow{\beta_1} \ldots \xrightarrow{\beta_n} \xrightarrow{\alpha} \longrightarrow \ldots$ with $\alpha \in ample(s_0)$, $\beta_i \notin ample(s_0)$
		$s_0 \xRightarrow{\alpha} \xrightarrow{\beta_1} \ldots \xrightarrow{\beta_n} \longrightarrow \ldots$
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. $\rho \overset{\Delta}{=} \rho_{red}$

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 $\beta_i \notin ample(s_0)$
		$s_0 \overset{\alpha}{\Longrightarrow} \xrightarrow{\beta_1} \xrightarrow{\beta_2} \xrightarrow{\beta_3} \ldots$ 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. $\rho \overset{\Delta}{=} \rho_{\text{red}}$

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

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

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

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

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

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

$$\text{execution } \rho \text{ in } \mathcal{T} \quad \rightsquigarrow \quad \begin{array}{c}\text{execution } \rho_{red} \text{ in } \mathcal{T}_{red} \\ \text{s.t. } \rho \triangleq \rho_{red}\end{array}$$

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

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

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

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

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

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

Transformation $\rho \rightsquigarrow \rho_1$

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

---

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 $\alpha \in \mathit{ample}(s_0)$

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

---

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

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

---

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 $\alpha \in ample(s_0)$

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

---

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 $\beta_i \notin ample(s_0)$

Transformation $\rho \rightsquigarrow \rho_1$

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

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

---

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

$\beta_i \notin ample(s_0)$

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

---

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

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

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'

Transformation according to cases 1 and 2

$$
\begin{array}{llll}
\rho_0 = & s_0 \xrightarrow{\beta_1} \ldots & & \xrightarrow{\beta_{k-1}} \xrightarrow{\beta_k} \xrightarrow{\beta_{k+1}} \ldots \\
\rho_1 = & s_0 \xRightarrow{\alpha_1} s_1 \xrightarrow{\beta_1} \ldots & & \xrightarrow{\beta_{k-1}} \xrightarrow{\beta_k} \xrightarrow{\beta_{k+1}} \ldots \\
\rho_2 = & s_0 \xRightarrow{\alpha_1} \xRightarrow{\alpha_2} s_2 \xrightarrow{\beta_1} \ldots & & \xrightarrow{\beta_{k-1}} \xrightarrow{\beta_k} \xrightarrow{\beta_{k+1}} \ldots \\
\vdots & & & \\
\rho_m = & s_0 \xRightarrow{\alpha_1} \xRightarrow{\alpha_2} \ldots \xRightarrow{\alpha_m} s_m \xrightarrow{\beta_1} & \ldots & \xrightarrow{\beta_k} \xrightarrow{\beta_{k+1}} \ldots
\end{array}
$$

Transformation according to cases 1 and 2

$$
\begin{array}{llll}
\rho_0 = & s_0 \xrightarrow{\beta_1} \ldots & & \xrightarrow{\beta_{k-1}} \xrightarrow{\beta_k} \xrightarrow{\beta_{k+1}} \ldots \\
\rho_1 = & s_0 \overset{\alpha_1}{\Longrightarrow} s_1 \xrightarrow{\beta_1} \ldots & & \xrightarrow{\beta_{k-1}} \xrightarrow{\beta_k} \xrightarrow{\beta_{k+1}} \ldots \\
\rho_2 = & s_0 \overset{\alpha_1}{\Longrightarrow} \overset{\alpha_2}{\Longrightarrow} s_2 \xrightarrow{\beta_1} \ldots & & \xrightarrow{\beta_{k-1}} \xrightarrow{\beta_k} \xrightarrow{\beta_{k+1}} \ldots \\
\vdots & & & \\
\rho_m = & s_0 \overset{\alpha_1}{\Longrightarrow} \overset{\alpha_2}{\Longrightarrow} \ldots \overset{\alpha_m}{\Longrightarrow} s_m \xrightarrow{\beta_1} & \ldots & \xrightarrow{\beta_k} \xrightarrow{\beta_{k+1}} \ldots
\end{array}
$$

α_i stutter action

Transformation according to cases 1 and 2

$$\rho_0 = \quad s_0 \xrightarrow{\beta_1} \ldots \quad \xrightarrow{\beta_{k-1}} \xrightarrow{\beta_k} \xrightarrow{\beta_{k+1}} \ldots$$

$$\rho_1 = \quad s_0 \overset{\alpha_1}{\Longrightarrow} s_1 \xrightarrow{\beta_1} \ldots \quad \xrightarrow{\beta_{k-1}} \xrightarrow{\beta_k} \xrightarrow{\beta_{k+1}} \ldots$$

$$\rho_2 = \quad s_0 \overset{\alpha_1}{\Longrightarrow} \overset{\alpha_2}{\Longrightarrow} s_2 \xrightarrow{\beta_1} \ldots \quad \xrightarrow{\beta_{k-1}} \xrightarrow{\beta_k} \xrightarrow{\beta_{k+1}} \ldots$$

$$\vdots$$

$$\rho_m = \quad s_0 \overset{\alpha_1}{\Longrightarrow} \overset{\alpha_2}{\Longrightarrow} \ldots \overset{\alpha_m}{\Longrightarrow} s_m \xrightarrow{\beta_1} \quad \ldots \xrightarrow{\beta_k} \xrightarrow{\beta_{k+1}} \ldots$$

α_i stutter action $\rightsquigarrow \rho_0 \triangleq \rho_1 \triangleq \rho_2 \triangleq \ldots$

Transformation according to cases 1 and 2

$$
\begin{array}{lll}
\rho_0 = & s_0 \xrightarrow{\beta_1} \ldots & \xrightarrow{\beta_{k-1}} \xrightarrow{\beta_k} \xrightarrow{\beta_{k+1}} \ldots \\
\rho_1 = & s_0 \xRightarrow{\alpha_1} s_1 \xrightarrow{\beta_1} \ldots & \xrightarrow{\beta_{k-1}} \xrightarrow{\beta_k} \xrightarrow{\beta_{k+1}} \ldots \\
\rho_2 = & s_0 \xRightarrow{\alpha_1} \xRightarrow{\alpha_2} s_2 \xrightarrow{\beta_1} \ldots & \xrightarrow{\beta_{k-1}} \xrightarrow{\beta_k} \xrightarrow{\beta_{k+1}} \ldots \\
\vdots & & \\
\rho_m = & s_0 \xRightarrow{\alpha_1} \xRightarrow{\alpha_2} \ldots \xRightarrow{\alpha_m} s_m \xrightarrow{\beta_1} & \ldots \xrightarrow{\beta_k} \xrightarrow{\beta_{k+1}} \ldots
\end{array}
$$

by the cycle condition (A4):

"action β_1 will *not* be postponed forever"

Transformation according to cases 1 and 2

$$
\begin{array}{llll}
\boldsymbol{\rho}_0 = & s_0 \xrightarrow{\beta_1} \ldots & & \xrightarrow{\beta_{k-1}} \xrightarrow{\beta_k} \xrightarrow{\beta_{k+1}} \ldots \\
\boldsymbol{\rho}_1 = & s_0 \xRightarrow{\alpha_1} s_1 \xrightarrow{\beta_1} \ldots & & \xrightarrow{\beta_{k-1}} \xrightarrow{\beta_k} \xrightarrow{\beta_{k+1}} \ldots \\
\boldsymbol{\rho}_2 = & s_0 \xRightarrow{\alpha_1} \xRightarrow{\alpha_2} s_2 \xrightarrow{\beta_1} \ldots & & \xrightarrow{\beta_{k-1}} \xrightarrow{\beta_k} \xrightarrow{\beta_{k+1}} \ldots \\
\vdots & & & \\
\boldsymbol{\rho}_m = & s_0 \xRightarrow{\alpha_1} \xRightarrow{\alpha_2} \ldots \xRightarrow{\alpha_m} s_m \xrightarrow{\beta_1} & \ldots & \xrightarrow{\beta_k} \xrightarrow{\beta_{k+1}} \ldots
\end{array}
$$

by the cycle condition (A4):

"action β_1 will *not* be postponed forever"

i.e., there exists some m such that case 0 applies and $\boldsymbol{\rho}_m = \boldsymbol{\rho}_{m+1}$

Transformation according to cases 1 and 2

$$\begin{array}{lll}
\boldsymbol{\rho}_0 = & s_0 \xrightarrow{\boldsymbol{\beta}_1} \ldots & \xrightarrow{\boldsymbol{\beta}_{k-1}} \xrightarrow{\boldsymbol{\beta}_k} \xrightarrow{\boldsymbol{\beta}_{k+1}} \ldots \\
\boldsymbol{\rho}_1 = & s_0 \xRightarrow{\boldsymbol{\alpha}_1} s_1 \xrightarrow{\boldsymbol{\beta}_1} \ldots & \xrightarrow{\boldsymbol{\beta}_{k-1}} \xrightarrow{\boldsymbol{\beta}_k} \xrightarrow{\boldsymbol{\beta}_{k+1}} \ldots \\
\boldsymbol{\rho}_2 = & s_0 \xRightarrow{\boldsymbol{\alpha}_1} \xRightarrow{\boldsymbol{\alpha}_2} s_2 \xrightarrow{\boldsymbol{\beta}_1} \ldots & \xrightarrow{\boldsymbol{\beta}_{k-1}} \xrightarrow{\boldsymbol{\beta}_k} \xrightarrow{\boldsymbol{\beta}_{k+1}} \ldots \\
\vdots & & \\
\boldsymbol{\rho}_m = & s_0 \xRightarrow{\boldsymbol{\alpha}_1} \xRightarrow{\boldsymbol{\alpha}_2} \ldots \xRightarrow{\boldsymbol{\alpha}_m} s_m \xrightarrow{\boldsymbol{\beta}_1} & \ldots \xrightarrow{\boldsymbol{\beta}_k} \xrightarrow{\boldsymbol{\beta}_{k+1}} \ldots
\end{array}$$

by the cycle condition $(\mathrm{A4})$:

"action $\boldsymbol{\beta}_1$ will *not* be postponed forever"

i.e., there exists some m such that case 0 applies and $\boldsymbol{\rho}_m = \boldsymbol{\rho}_{m+1}$

Transformation according to cases 1 and 2

$$
\begin{array}{lll}
\boldsymbol{\rho}_0 = & s_0 \xrightarrow{\beta_1} \ldots & \xrightarrow{\beta_{k-1}} \xrightarrow{\beta_k} \xrightarrow{\beta_{k+1}} \ldots \\
\boldsymbol{\rho}_1 = & s_0 \xRightarrow{\alpha_1} s_1 \xrightarrow{\beta_1} \ldots & \xrightarrow{\beta_{k-1}} \xrightarrow{\beta_k} \xrightarrow{\beta_{k+1}} \ldots \\
\boldsymbol{\rho}_2 = & s_0 \xRightarrow{\alpha_1} \xRightarrow{\alpha_2} s_2 \xrightarrow{\beta_1} \ldots & \xrightarrow{\beta_{k-1}} \xrightarrow{\beta_k} \xrightarrow{\beta_{k+1}} \ldots \\
\vdots & & \\
\boldsymbol{\rho}_m = & s_0 \xRightarrow{\alpha_1} \xRightarrow{\alpha_2} \ldots \xRightarrow{\alpha_m} s_m \xrightarrow{\beta_1} & \ldots \xrightarrow{\beta_k} \xrightarrow{\beta_{k+1}} \ldots \\
\boldsymbol{\rho}_{m+1} = & s_0 \xRightarrow{\alpha_1} \xRightarrow{\alpha_2} \ldots \xRightarrow{\alpha_m} s_m \xRightarrow{\beta_1} & \ldots \xrightarrow{\beta_k} \xrightarrow{\beta_{k+1}} \ldots
\end{array}
$$

by the cycle condition (A4):

"action β_1 will *not* be postponed forever"

i.e., there exists some m such that case 0 applies and $\boldsymbol{\rho}_m = \boldsymbol{\rho}_{m+1}$

4 conditions for ample sets LTL3.4-FOUR-COND

(A1) $\emptyset \neq \mathit{ample}(s) \subseteq \mathit{Act}(s)$

(A2) 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 $\mathit{ample}(s)$ there is some $i < n$ with $\beta_i \in \mathit{ample}(s)$

(A3) if $\mathit{ample}(s) \neq \mathit{Act}(s)$ then all actions in $\mathit{ample}(s)$ are *stutter actions*

(A4) for each *cycle* $s_0 \Rightarrow s_1 \Rightarrow \ldots \Rightarrow s_n$ in $\mathcal{T}_{red}$ and each action

$$\beta \in \bigcup_{0 \leq i < n} \mathit{Act}(s_i)$$

there is some $i \in \{1, \ldots, n\}$ with $\beta \in \mathit{ample}(s_i)$

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

LTL3.4-37

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

- on-the-fly DFS-based generatation of $\mathcal{T}_{red}$

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

- on-the-fly DFS-based generatation of $\mathcal{T}_{red}$
- *exploration* of state s:

  create the states α(s) for $\alpha \in$ ample(s),

The ample set method for $LTL_{\setminus\bigcirc}$ model checking

- on-the-fly DFS-based generatation of $\mathcal{T}_{red}$
- *exploration* of state s:

  create the states $\alpha(s)$ for $\alpha \in ample(s)$, but ignore the β-successors of s for $\beta \notin ample(s)$

The ample set method for $LTL_{\setminus\bigcirc}$ model checking

- on-the-fly DFS-based generatation of $\mathcal{T}_{red}$
- *exploration* of state s:

  create the states $\alpha(s)$ for $\alpha \in ample(s)$, but ignore the β-successors of s for $\beta \notin ample(s)$
- *interleave* the generation of $\mathcal{T}_{red}$ with the product construction $\mathcal{T}_{red} \otimes \mathcal{A}$

where $\mathcal{A}$ is an NBA for the negation of the formula to be checked

The ample set method for $LTL_{\setminus\bigcirc}$ model checking

- on-the-fly DFS-based generatation of $\mathcal{T}_{red}$
- *exploration* of state s:

  create the states $\alpha(s)$ for $\alpha \in ample(s)$, but ignore the β-successors of s for $\beta \notin ample(s)$
- *interleave* the generation of $\mathcal{T}_{red}$ with the product construction $\mathcal{T}_{red} \otimes \mathcal{A}$ and nested DFS

where $\mathcal{A}$ is an NBA for the negation of the formula to be checked

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

- on-the-fly DFS-based generatation of $\mathcal{T}_{red}$
- *exploration* of state s:

  create the states α(s) for $\alpha \in$ ample(s), but ignore the β-successors of s for $\beta \notin$ ample(s)
- *interleave* the generation of $\mathcal{T}_{\text{red}}$ with the product construction $\mathcal{T}_{\text{red}} \otimes \mathcal{A}$ and nested DFS

where $\mathcal{A}$ is an NBA for the negation of the formula to be checked

here: only explanations for reachability analysis

The ample set method for reachability

given: finite transition system $\mathcal{T}$
atomic proposition a

goal: on-the-fly construction of $\mathcal{T}_{red}$
abort as soon as a state s with
$s \not\models a$ has been generated

The ample set method for reachability

given: finite transition system $\mathcal{T}$
atomic proposition **a**

goal: on-the-fly construction of $\mathcal{T}_{red}$
abort as soon as a state s with
s $\not\models$ **a** has been generated

uses

- **V** $=$ set of states that have been generated so far (organized as a hash table)

The ample set method for reachability

given: finite transition system $\mathcal{T}$
atomic proposition **a**

goal: on-the-fly construction of $\mathcal{T}_{red}$
abort as soon as a state s with
s $\not\models$ **a** has been generated

uses

- **V** = set of states that have been generated so far (organized as a hash table)
- DFS-stack π

The ample set method for reachability

given: finite transition system $\mathcal{T}$ for $P_1 \| \ldots \| P_n$
atomic proposition **a**

goal: on-the-fly construction of $\mathcal{T}_{\text{red}}$
abort as soon as a state s with
$s \not\models$ **a** has been generated

uses

- **V** = set of states that have been generated so far (organized as a hash table)
- DFS-stack π
- "local" criteria to compute ample(s) from a syntactic representation of the processes P_i

Ample set method (full generation of $\mathcal{T}_{\textsf{red}}$)

$\pi := \emptyset$; $\mathbf{V} := \emptyset$

WHILE $\mathbf{S}_0 \not\subseteq \mathbf{V}$ **DO**

select an initial state $\mathbf{s} \in \mathbf{S}_0 \setminus \mathbf{V}$; add $\mathbf{s}$ to $\mathbf{V}$;

Push(π, $\mathbf{s}$);

OD

Ample set method (full generation of $\mathcal{T}_{\text{red}}$)

$\pi := \emptyset$; $\mathbf{V} := \emptyset$

WHILE $\mathbf{S}_0 \not\subseteq \mathbf{V}$ **DO**

select an initial state $\mathbf{s} \in \mathbf{S}_0 \setminus \mathbf{V}$; add $\mathbf{s}$ to $\mathbf{V}$;

Push(π, $\mathbf{s}$); compute ample($\mathbf{s}$);

OD

Ample set method (full generation of $\mathcal{T}_{\mathsf{red}}$)

$\pi := \emptyset$; $\mathsf{V} := \emptyset$

WHILE $\mathsf{S}_0 \not\subseteq \mathsf{V}$ **DO**

  select an initial state $\mathsf{s} \in \mathsf{S}_0 \setminus \mathsf{V}$; add s to V;

  $\mathsf{Push}(\pi, \mathsf{s})$; compute $\mathsf{ample}(\mathsf{s})$;

  WHILE $\pi \neq \emptyset$ **DO**

    $\mathsf{s} := \mathsf{Top}(\pi)$;

  OD

OD

$\pi := \emptyset$; $\mathbf{V} := \emptyset$

WHILE $\mathbf{S}_0 \not\subseteq \mathbf{V}$ **DO**

  select an initial state $\mathbf{s} \in \mathbf{S}_0 \setminus \mathbf{V}$; add $\mathbf{s}$ to $\mathbf{V}$;

  Push(π, $\mathbf{s}$); compute ample($\mathbf{s}$);

  WHILE $\pi \neq \emptyset$ **DO**

    $\mathbf{s}$:= Top(π);

    IF $\exists \alpha \in$ ample($\mathbf{s}$) with $\alpha(\mathbf{s}) \notin \mathbf{V}$

    FI

  OD

OD

Ample set method (full generation of $\mathcal{T}_{red}$)

$\pi := \emptyset$; $\mathbf{V} := \emptyset$

WHILE $\mathbf{S}_0 \not\subseteq \mathbf{V}$ **DO**

  select an initial state $\mathbf{s} \in \mathbf{S}_0 \setminus \mathbf{V}$; add $\mathbf{s}$ to $\mathbf{V}$;

  Push(π, $\mathbf{s}$); compute ample($\mathbf{s}$);

  WHILE $\pi \neq \emptyset$ **DO**

    $\mathbf{s}$:= Top(π);

    IF $\exists \alpha \in$ ample($\mathbf{s}$) with $\alpha(\mathbf{s}) \notin \mathbf{V}$

      THEN select such α; add $\mathbf{s}' := \alpha(\mathbf{s})$ to $\mathbf{V}$;

        Push(π, $\mathbf{s}'$);

    FI

  OD

OD

```
π := ∅; V := ∅
WHILE S₀ ⊈ V DO
   select an initial state s ∈ S₀ \ V; add s to V;
   Push(π, s); compute ample(s);
   WHILE π ≠ ∅ DO
      s := Top(π);
      IF ∃α ∈ ample(s) with α(s) ∉ V
        THEN select such α; add s′ := α(s) to V;
                Push(π, s′); compute ample(s′);

      FI
   OD
OD
```

Ample set method (full generation of $\mathcal{T}_{red}$)

$\pi := \emptyset$; $V := \emptyset$

WHILE $S_0 \not\subseteq V$ **DO**

  select an initial state $s \in S_0 \setminus V$; add s to V;

  $Push(\pi, s)$; compute $ample(s)$;

  WHILE $\pi \neq \emptyset$ **DO**

    $s := Top(\pi)$;

    IF $\exists \alpha \in ample(s)$ with $\alpha(s) \notin V$

      THEN select such α; add $s' := \alpha(s)$ to V;

        $Push(\pi, s')$; compute $ample(s')$;

      ELSE $Pop(\pi)$

    FI

  OD

OD

The ample set method for reachability

$\pi := \emptyset$; $V := \emptyset$
WHILE $S_0 \not\subseteq V$ **DO**
  select an initial state $s \in S_0 \setminus V$; add s to V;
  $\mathrm{Push}(\pi, s)$; compute $\mathrm{ample}(s)$;
  WHILE $\pi \neq \emptyset$ **DO**
    $s := \mathrm{Top}(\pi)$;
    IF $\exists \alpha \in \mathrm{ample}(s)$ with $\alpha(s) \notin V$
      THEN select such α; add $s' := \alpha(s)$ to V;
        $\mathrm{Push}(\pi, s')$; compute $\mathrm{ample}(s')$;
      ELSE $\mathrm{Pop}(\pi)$
    FI
  OD
OD

$\pi := \emptyset$; $V := \emptyset$
WHILE $S_0 \not\subseteq V$ **DO**
  select an initial state $s \in S_0 \setminus V$; add s to V;
  $Push(\pi, s)$; compute $ample(s)$;
  WHILE $\pi \neq \emptyset$ **DO**
    $s := Top(\pi)$;
    IF $\exists \alpha \in ample(s)$ with $\alpha(s) \notin V$
      THEN select such α; add $s' := \alpha(s)$ to V;
        $Push(\pi, s')$; compute $ample(s')$;
      ELSE $Pop(\pi)$
    FI
  OD
OD

Does $\mathcal{T} \models \Box a$ hold?


```
π := ∅; V := ∅
WHILE S₀ ⊄ V DO
  select an initial state s ∈ S₀ \ V; add s to V;
  Push(π, s); compute ample(s);
  WHILE π ≠ ∅ DO
    s := Top(π);
    IF s ⊭ a THEN return "NO"                FI;
    IF ∃α ∈ ample(s) with α(s) ∉ V
      THEN ....
      ELSE  Pop(π)
    FI
  OD
OD
```

Does $\mathcal{T} \models \Box a$ hold?

$\pi := \emptyset$; $V := \emptyset$

WHILE $S_0 \not\subseteq V$ **DO**

  select an initial state $s \in S_0 \setminus V$; add s to V;

  $Push(\pi, s)$; compute $ample(s)$;

  WHILE $\pi \neq \emptyset$ **DO**

    $s := Top(\pi)$;

    IF $s \not\models a$ **THEN** return "NO" + counterexample **FI**;

    IF $\exists \alpha \in ample(s)$ with $\alpha(s) \notin V$

      THEN ....

      ELSE $Pop(\pi)$

    FI

  OD

OD

Example: ample set method

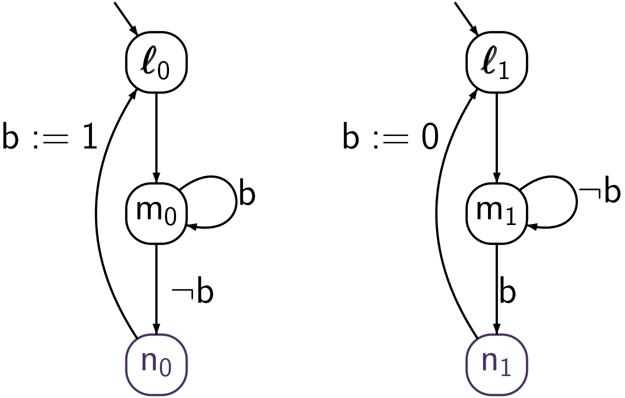
full generation of $\mathcal{T}_{\mathsf{red}}$ for $\mathcal{T} = \mathcal{T}_{\mathsf{P}_1 \,|||\, \mathsf{P}_2}$ where

- P_1, P_2 are program graphs with shared variable $\mathsf{b} \in \{0, 1\}$

Example: ample set method

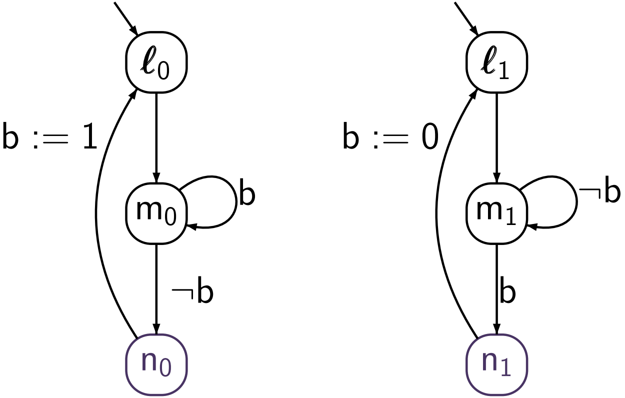
full generation of $\mathcal{T}_{red}$ for $\mathcal{T} = \mathcal{T}_{P_1 \,|||\, P_2}$ where

- P_1, P_2 are program graphs with shared variable $b \in \{0, 1\}$



full generation of $\mathcal{T}_{\text{red}}$ for $\mathcal{T} = \mathcal{T}_{P_1 \,|||\, P_2}$ where

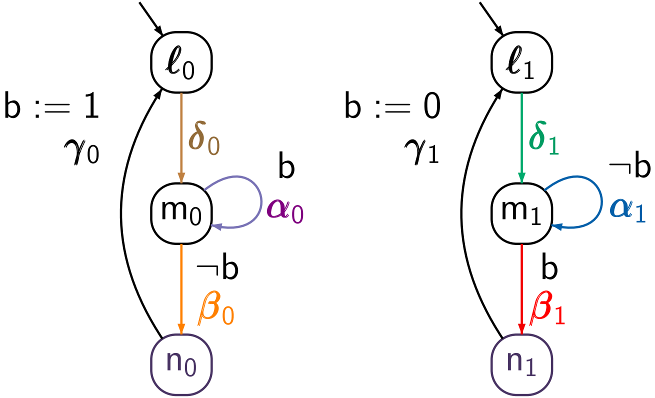
- P_1, P_2 are program graphs with shared variable $b \in \{0, 1\}$
- $AP = \{n_0, n_1\}$

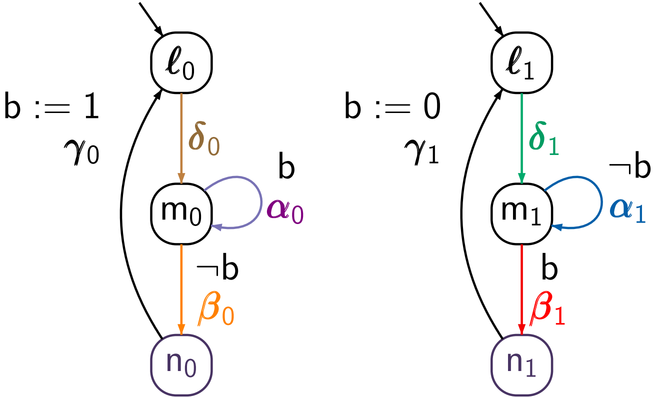


Example: ample set method

full generation of $\mathcal{T}_{\text{red}}$ for $\mathcal{T} = \mathcal{T}_{P_1 \,|||\, P_2}$ where

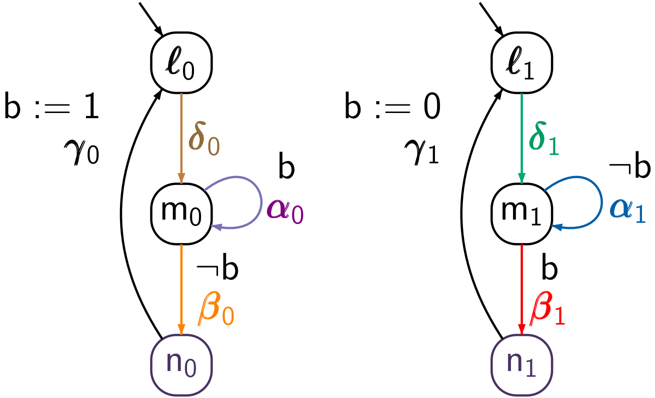
- P_1, P_2 are program graphs with shared variable $b \in \{0, 1\}$
- $AP = \{n_0, n_1\}$





independent actions:

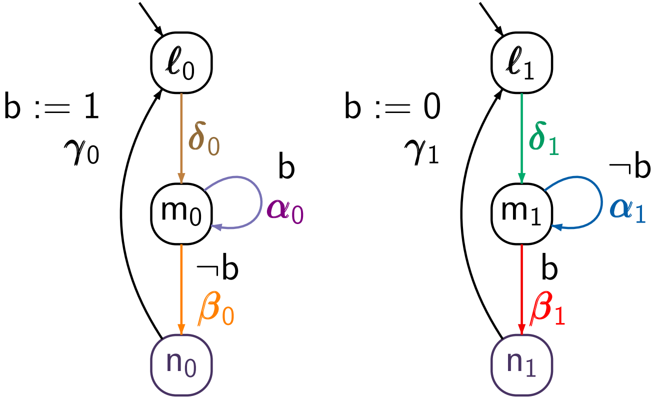
$\delta_0\ \delta_1$ $\quad$ $\delta_0\ \alpha_1$ $\quad$ $\delta_0\ \beta_1$ $\quad$ $\delta_0\ \gamma_1$



independent actions:

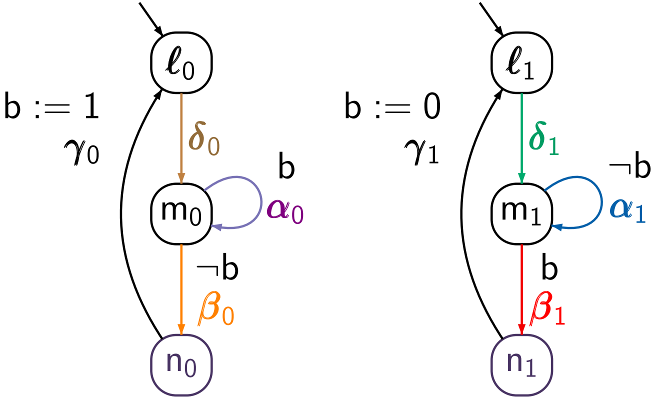
$\delta_0\ \delta_1$ $\quad$ $\delta_0\ \alpha_1$ $\quad$ $\delta_0\ \beta_1$ $\quad$ $\delta_0\ \gamma_1$

$\alpha_0\ \delta_1$ $\quad$ $\alpha_0\ \beta_1$



independent actions:

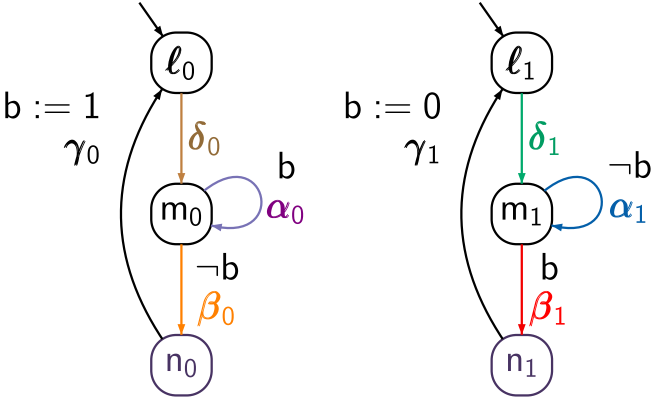
$\delta_0\ \delta_1$	$\delta_0\ \alpha_1$	$\delta_0\ \beta_1$	$\delta_0\ \gamma_1$
$\alpha_0\ \delta_1$		$\alpha_0\ \beta_1$	
$\beta_0\ \delta_1$	$\beta_0\ \alpha_1$	$\beta_0\ \beta_1$	$\beta_0\ \gamma_1$



independent actions:

$\delta_0\ \delta_1$	$\delta_0\ \alpha_1$	$\delta_0\ \beta_1$	$\delta_0\ \gamma_1$
$\alpha_0\ \delta_1$		$\alpha_0\ \beta_1$	
$\beta_0\ \delta_1$	$\beta_0\ \alpha_1$	$\beta_0\ \beta_1$	$\beta_0\ \gamma_1$

β_0 and β_1 are never enabled simultaneously



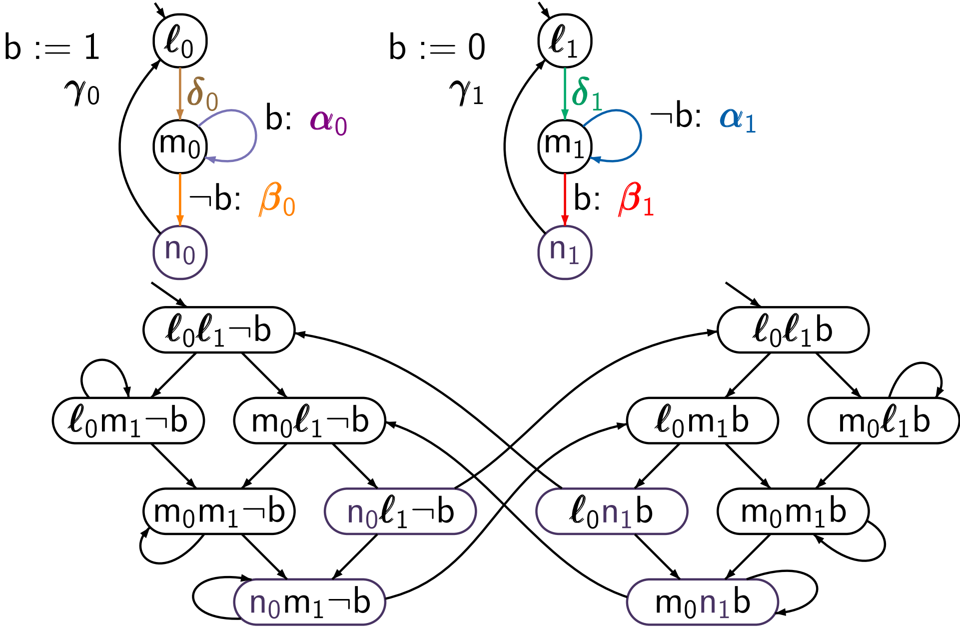
independent actions:

$\delta_0\ \delta_1$ $\delta_0\ \alpha_1$ $\delta_0\ \beta_1$ $\delta_0\ \gamma_1$

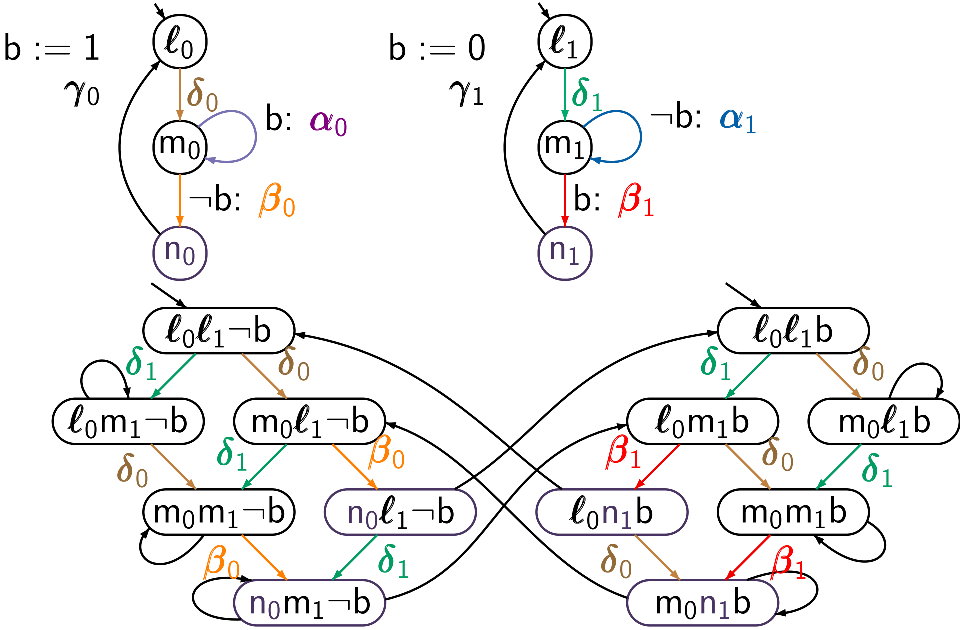
$\alpha_0\ \delta_1$ $\alpha_0\ \beta_1$

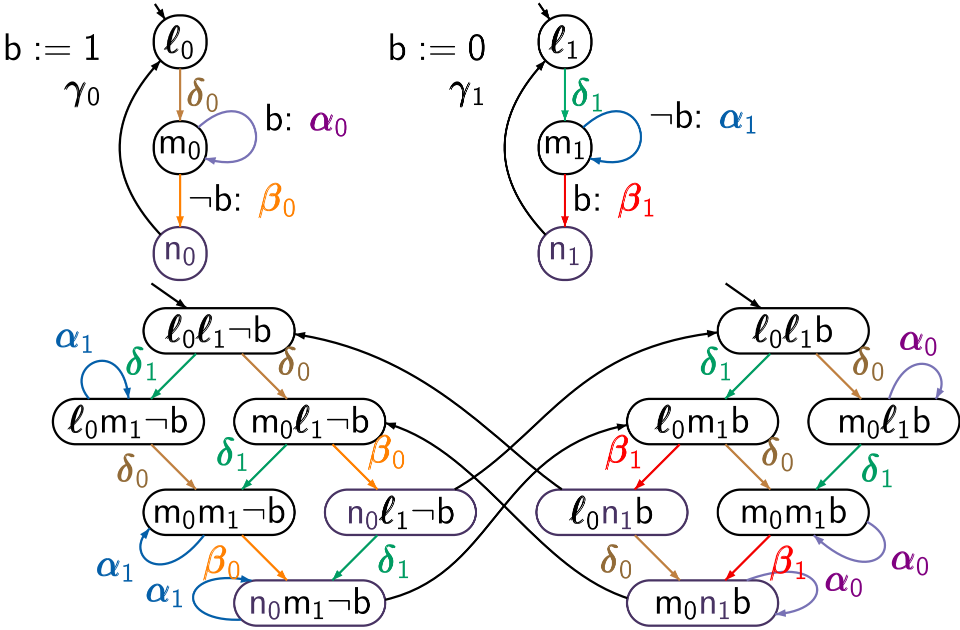
$\beta_0\ \delta_1$ $\beta_0\ \alpha_1$ $\beta_0\ \beta_1$ $\beta_0\ \gamma_1$

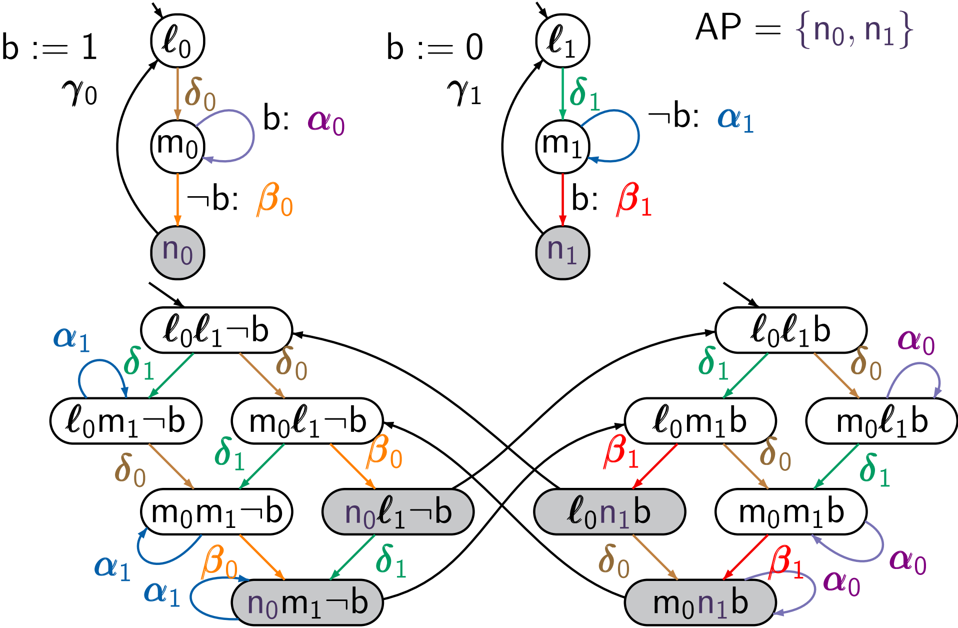
$\gamma_0\ \delta_1$ $\gamma_0\ \beta_1$



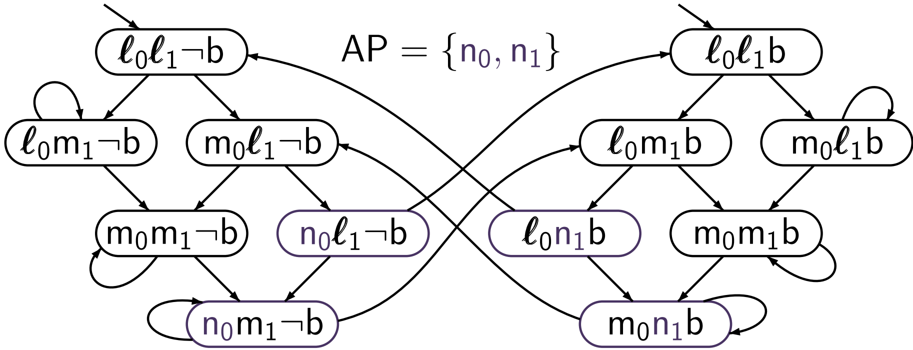
b := 1
γ0
ℓ0
δ0
m0
b: α0
¬b: β0
n0
b := 0
γ1
ℓ1
δ1
m1
¬b: α1
b: β1
n1
ℓ0ℓ1¬b
ℓ0m1¬b
m0ℓ1¬b
m0m1¬b
n0ℓ1¬b
n0m1¬b
ℓ0ℓ1b
ℓ0m1b
m0ℓ1b
ℓ0n1b
m0m1b
m0n1b


b := 1
γ0
ℓ0
δ0
m0
b: α0
¬b: β0
n0
b := 0
γ1
ℓ1
δ1
m1
¬b: α1
b: β1
n1
ℓ0ℓ1¬b
ℓ0m1¬b
m0ℓ1¬b
m0m1¬b
n0ℓ1¬b
n0m1¬b
ℓ0ℓ1b
ℓ0m1b
m0ℓ1b
ℓ0n1b
m0m1b
m0n1b

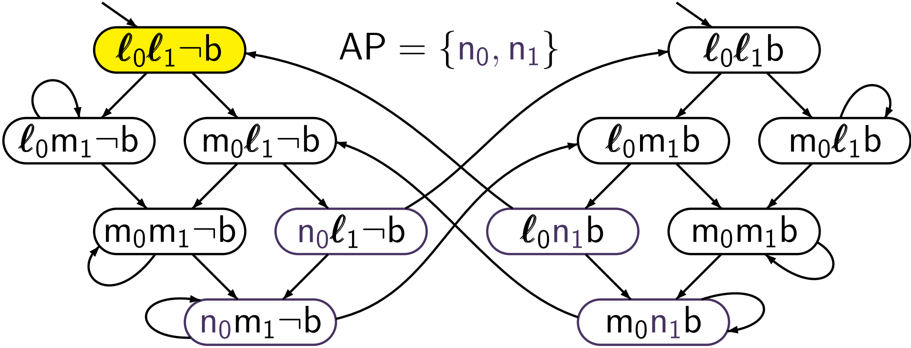

b := 1
γ0
ℓ0
δ0
m0
b: α0
¬b: β0
n0
b := 0
γ1
ℓ1
δ1
m1
¬b: α1
b: β1
n1
ℓ0ℓ1¬b
ℓ0m1¬b
m0ℓ1¬b
m0m1¬b
n0ℓ1¬b
n0m1¬b
ℓ0ℓ1b
ℓ0m1b
m0ℓ1b
ℓ0n1b
m0m1b
m0n1b


b := 1
γ_0
ℓ_0
δ_0
m_0
b: α_0
$\neg$b: β_0
n_0
b := 0
γ_1
ℓ_1
δ_1
m_1
$\neg$b: α_1
b: β_1
n_1
AP = $\{n_0, n_1\}$
$\ell_0\ell_1\neg$b
$\ell_0 m_1\neg$b
$m_0\ell_1\neg$b
$m_0 m_1\neg$b
$n_0\ell_1\neg$b
$n_0 m_1\neg$b
$\ell_0\ell_1$b
$\ell_0 m_1$b
$m_0\ell_1$b
$\ell_0 n_1$b
$m_0 m_1$b
$m_0 n_1$b

Example: on-the-fly generation of $\mathcal{T}_{\text{red}}$

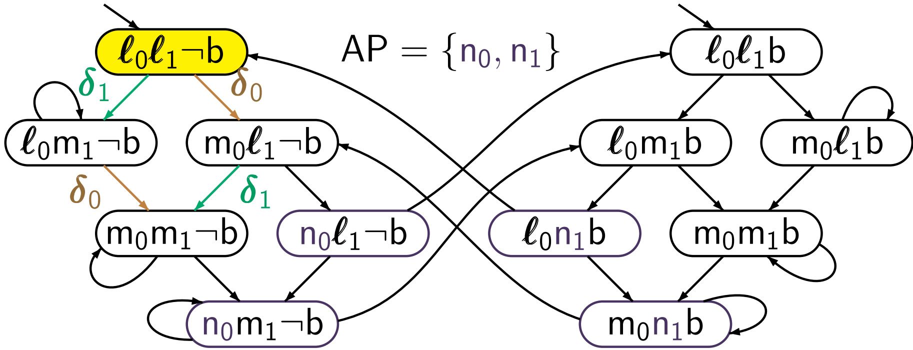


Example: on-the-fly generation of $\mathcal{T}_{red}$



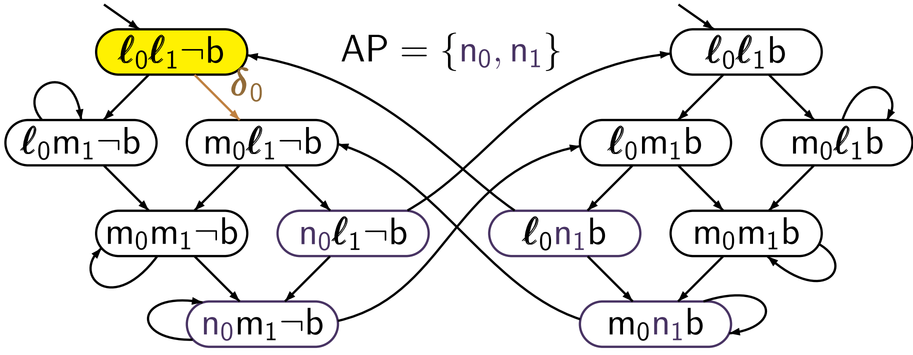
$AP = \{n_0, n_1\}$

$\text{ample}(\ell_0\ell_1\neg b) =$



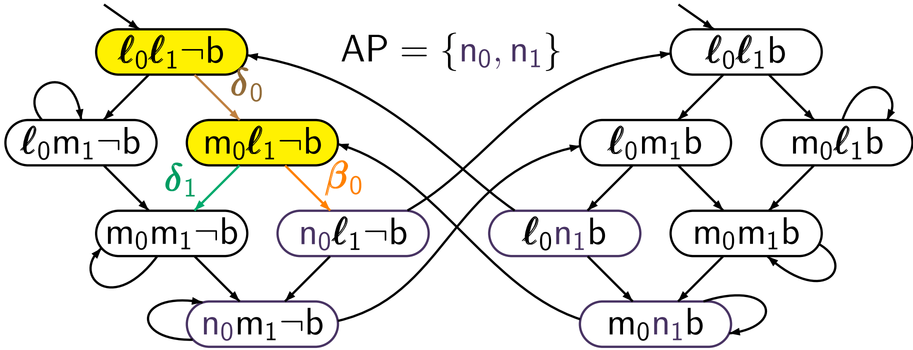
$\mathsf{ample}(\ell_0\ell_1\neg b) =$

Example: on-the-fly generation of $\mathcal{T}_{\text{red}}$



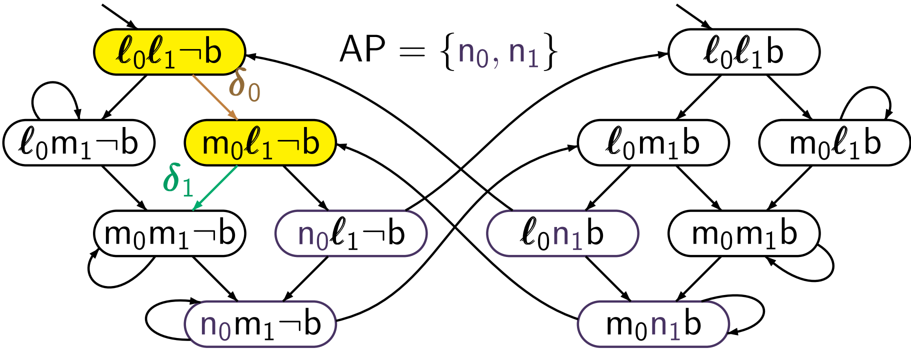
$\text{ample}(\ell_0\ell_1\neg b) = \{\delta_0\},$

Example: on-the-fly generation of $\mathcal{T}_{\text{red}}$



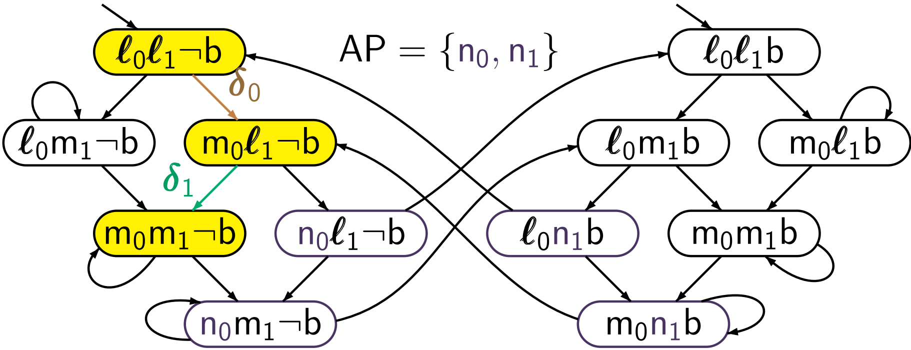
$\text{ample}(\ell_0\ell_1\neg b) = \{\delta_0\}, \quad \text{ample}(m_0\ell_1\neg b) =$

Example: on-the-fly generation of $\mathcal{T}_{\text{red}}$



$$\text{ample}(\ell_0\ell_1\neg b) = \{\delta_0\}, \quad \text{ample}(m_0\ell_1\neg b) = \{\delta_1\}$$

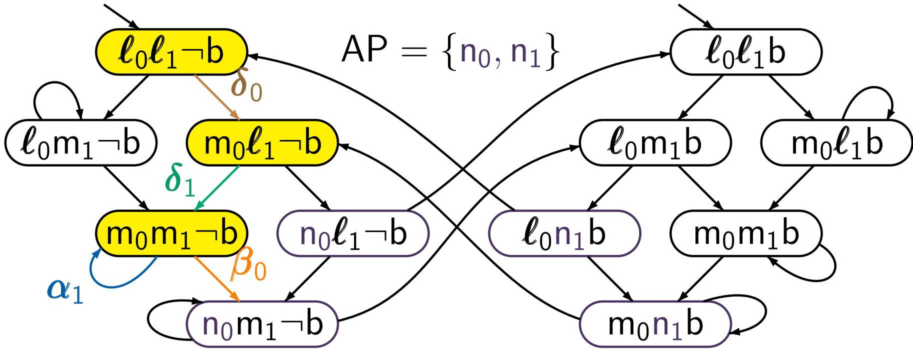
Example: on-the-fly generation of $\mathcal{T}_{\text{red}}$



$\text{ample}(\ell_0\ell_1\neg b) = \{\delta_0\}, \quad \text{ample}(m_0\ell_1\neg b) = \{\delta_1\}$

$\text{ample}(m_0 m_1 \neg b) =$

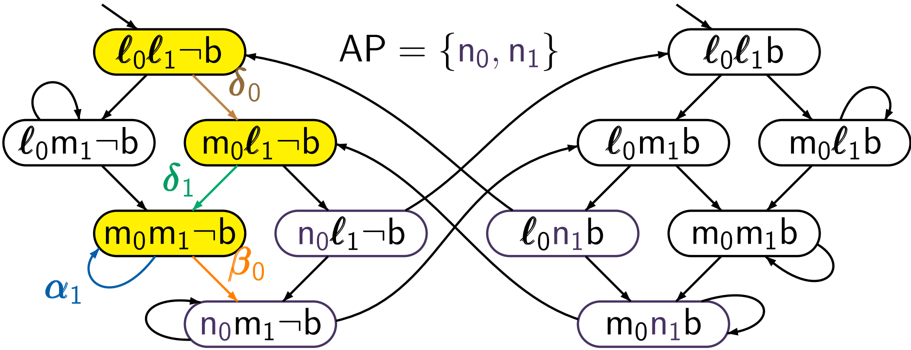
Example: on-the-fly generation of $\mathcal{T}_{\mathsf{red}}$



$\text{ample}(\ell_0\ell_1\neg b) = \{\delta_0\}, \quad \text{ample}(m_0\ell_1\neg b) = \{\delta_1\}$

$\text{ample}(m_0m_1\neg b) =$

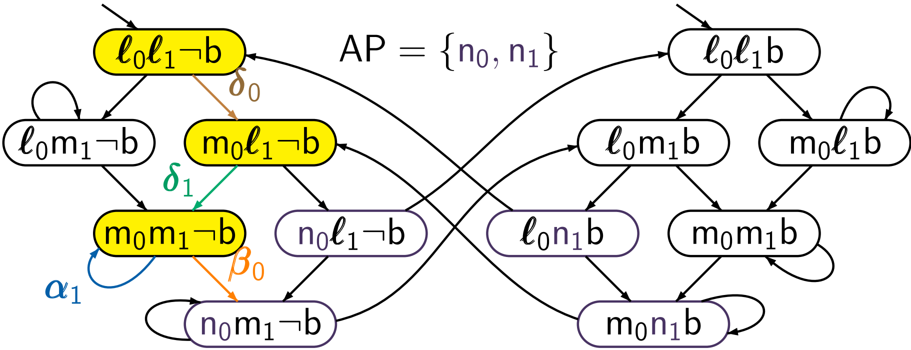
Example: on-the-fly generation of $\mathcal{T}_{\text{red}}$



$\text{ample}(\ell_0\ell_1\neg b) = \{\delta_0\}, \quad \text{ample}(m_0\ell_1\neg b) = \{\delta_1\}$

$\text{ample}(m_0 m_1\neg b) = \{\alpha_1, \beta_0\}$

Example: on-the-fly generation of $\mathcal{T}_{\text{red}}$



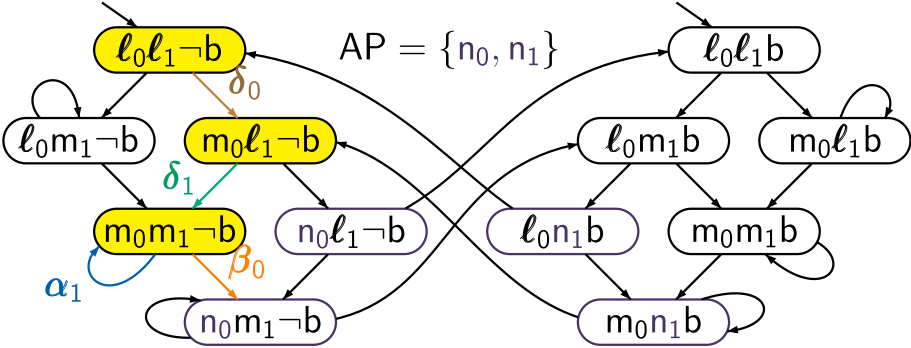
$\text{ample}(\ell_0\ell_1\neg b) = \{\delta_0\}$, $\text{ample}(m_0\ell_1\neg b) = \{\delta_1\}$

$\text{ample}(m_0m_1\neg b) = \{\alpha_1, \beta_0\}$

note:

α_1 closes cycle (A4),

Example: on-the-fly generation of $\mathcal{T}_{\text{red}}$



$\text{ample}(\ell_0\ell_1\neg b) = \{\delta_0\}$, $\text{ample}(m_0\ell_1\neg b) = \{\delta_1\}$

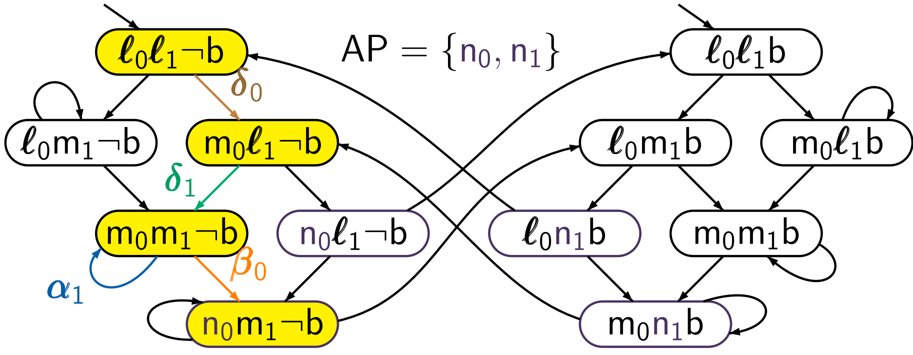
$\text{ample}(m_0m_1\neg b) = \{\alpha_1, \beta_0\}$

note:

α_1 closes cycle (A4),

β_0 no stutter action (A3)

Example: on-the-fly generation of $\mathcal{T}_{\text{red}}$

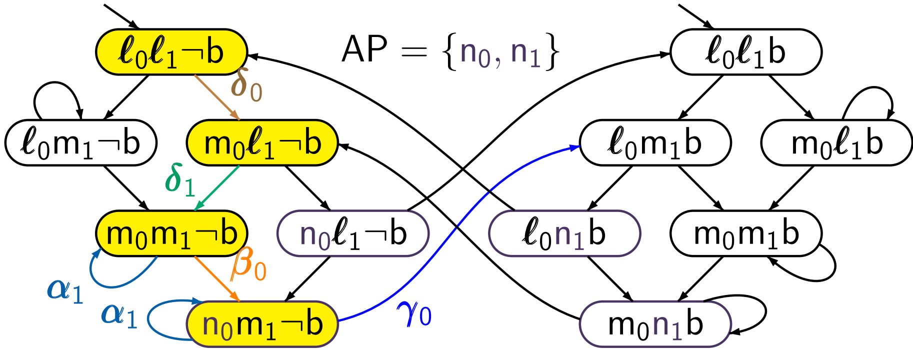


$\text{ample}(\ell_0\ell_1\neg b) = \{\delta_0\}, \quad \text{ample}(m_0\ell_1\neg b) = \{\delta_1\}$

$\text{ample}(m_0 m_1\neg b) = \{\alpha_1, \beta_0\}$

$\text{ample}(n_0 m_1\neg b) =$

Example: on-the-fly generation of $\mathcal{T}_{red}$

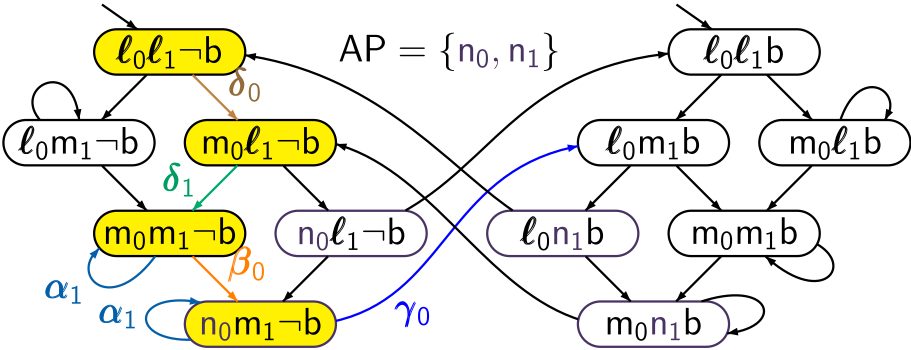


$\text{ample}(\ell_0\ell_1\neg b) = \{\delta_0\}, \quad \text{ample}(m_0\ell_1\neg b) = \{\delta_1\}$

$\text{ample}(m_0m_1\neg b) = \{\alpha_1, \beta_0\}$

$\text{ample}(n_0m_1\neg b) =$

Example: on-the-fly generation of $\mathcal{T}_{\text{red}}$

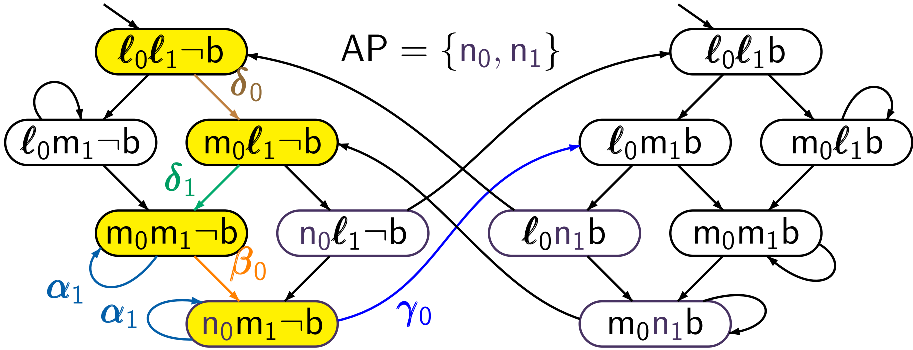


$\text{ample}(\ell_0\ell_1\neg b) = \{\delta_0\}, \quad \text{ample}(m_0\ell_1\neg b) = \{\delta_1\}$

$\text{ample}(m_0m_1\neg b) = \{\alpha_1, \beta_0\}$

$\text{ample}(n_0m_1\neg b) = \{\alpha_1, \gamma_0\}$

Example: on-the-fly generation of $\mathcal{T}_{red}$



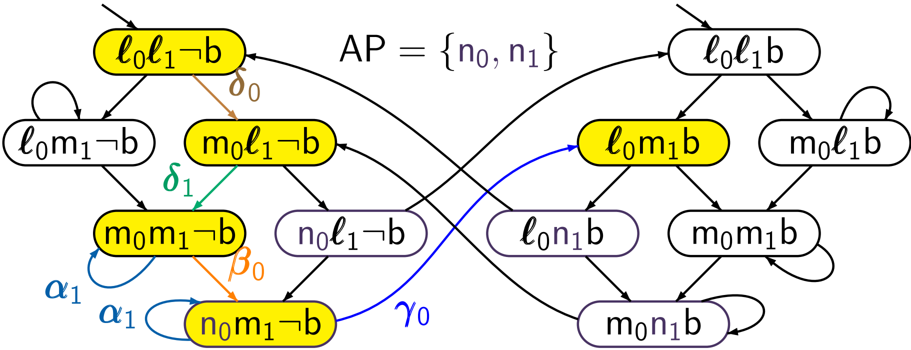
$\text{ample}(\ell_0\ell_1\neg b) = \{\delta_0\}$, $\quad \text{ample}(m_0\ell_1\neg b) = \{\delta_1\}$

$\text{ample}(m_0 m_1\neg b) = \{\alpha_1, \beta_0\}$

$\text{ample}(n_0 m_1\neg b) = \{\alpha_1, \gamma_0\}$

note: α_1, γ_0 are dependent (A2)

Example: on-the-fly generation of $\mathcal{T}_{\text{red}}$



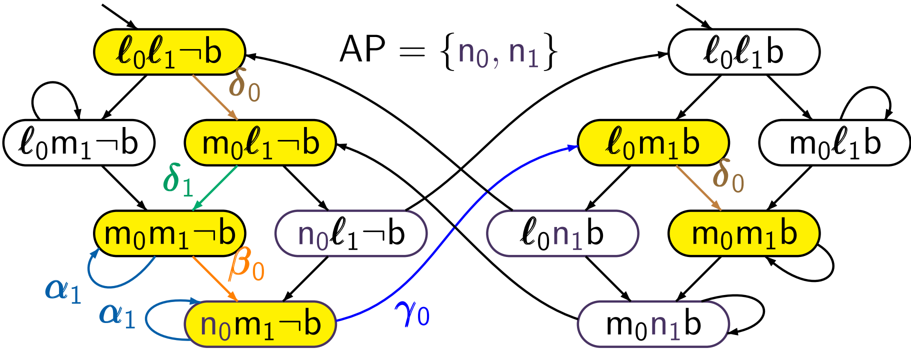
$\text{ample}(\ell_0\ell_1\neg b) = \{\delta_0\}$, $\text{ample}(m_0\ell_1\neg b) = \{\delta_1\}$

$\text{ample}(m_0m_1\neg b) = \{\alpha_1, \beta_0\}$

$\text{ample}(n_0m_1\neg b) = \{\alpha_1, \gamma_0\}$

note: α_1, γ_0 are dependent (A2)

Example: on-the-fly generation of $\mathcal{T}_{\text{red}}$



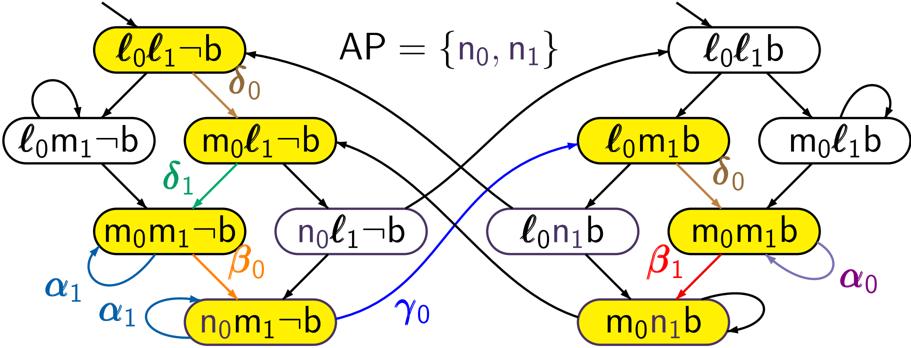
$$\text{ample}(\ell_0\ell_1\neg b) = \{\delta_0\}, \quad \text{ample}(m_0\ell_1\neg b) = \{\delta_1\}$$

$$\text{ample}(m_0 m_1\neg b) = \{\alpha_1, \beta_0\}$$

$$\text{ample}(n_0 m_1\neg b) = \{\alpha_1, \gamma_0\}$$

note: α_1, γ_0 are dependent (A2)

Example: on-the-fly generation of $\mathcal{T}_{red}$



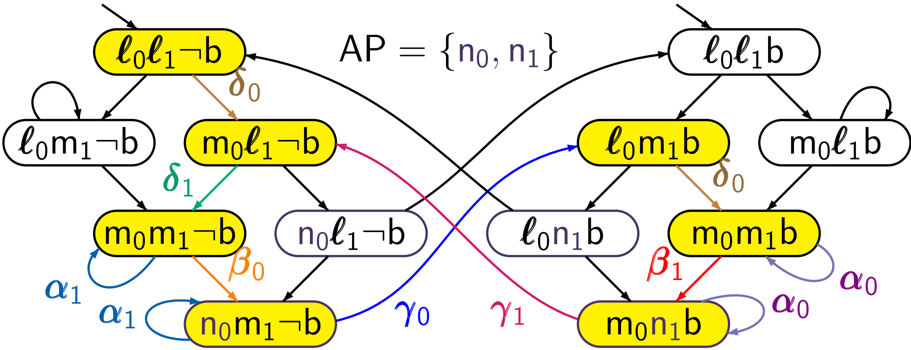
$\text{ample}(\ell_0\ell_1\neg b) = \{\delta_0\}$, $\text{ample}(m_0\ell_1\neg b) = \{\delta_1\}$

$\text{ample}(m_0 m_1\neg b) = \{\alpha_1, \beta_0\}$

$\text{ample}(n_0 m_1\neg b) = \{\alpha_1, \gamma_0\}$

note: α_1, γ_0 are dependent (A2)

Example: on-the-fly generation of $\mathcal{T}_{\text{red}}$



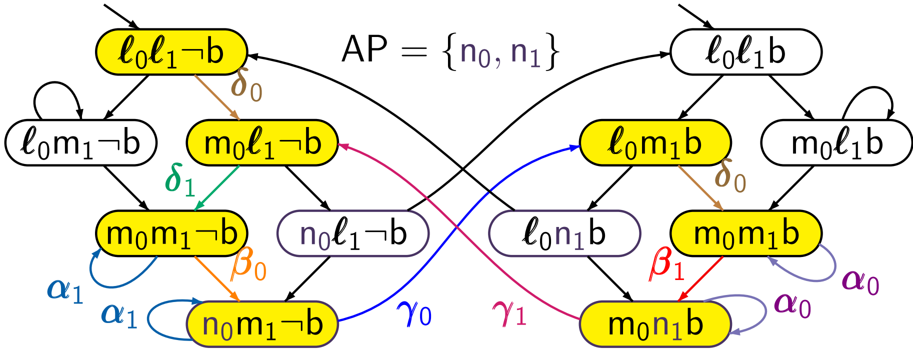
$\text{ample}(\ell_0\ell_1\neg b) = \{\delta_0\}, \quad \text{ample}(m_0\ell_1\neg b) = \{\delta_1\}$

$\text{ample}(m_0 m_1 \neg b) = \{\alpha_1, \beta_0\}$

$\text{ample}(n_0 m_1 \neg b) = \{\alpha_1, \gamma_0\}$

$\text{ample}(m_0 n_1 b) =$

Example: on-the-fly generation of $\mathcal{T}_{\text{red}}$



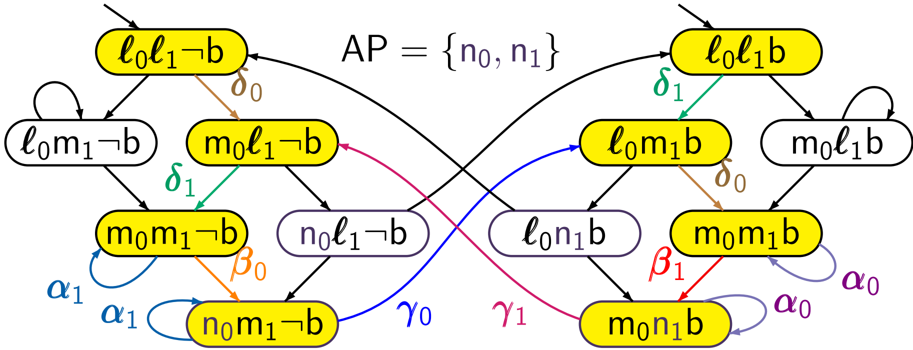
$\text{ample}(\ell_0\ell_1\neg b) = \{\delta_0\}$, $\text{ample}(m_0\ell_1\neg b) = \{\delta_1\}$

$\text{ample}(m_0m_1\neg b) = \{\alpha_1, \beta_0\}$

$\text{ample}(n_0m_1\neg b) = \{\alpha_1, \gamma_0\}$

$\text{ample}(m_0n_1b) = \{\alpha_0, \gamma_1\}$: cycle condition (A4)

Example: on-the-fly generation of $\mathcal{T}_{\text{red}}$



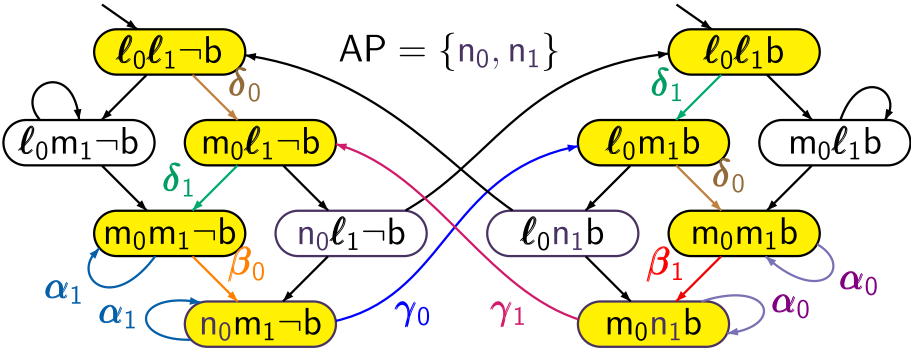
$\text{ample}(\ell_0\ell_1\neg b) = \{\delta_0\}$, $\text{ample}(m_0\ell_1\neg b) = \{\delta_1\}$

$\text{ample}(m_0m_1\neg b) = \{\alpha_1, \beta_0\}$

$\text{ample}(n_0m_1\neg b) = \{\alpha_1, \gamma_0\}$

$\text{ample}(m_0n_1b) = \{\alpha_0, \gamma_1\}$: cycle condition (A4)

reduction: 8 out of 12 states



$\text{ample}(\ell_0\ell_1\neg b) = \{\delta_0\}$, $\text{ample}(m_0\ell_1\neg b) = \{\delta_1\}$

$\text{ample}(m_0m_1\neg b) = \{\alpha_1, \beta_0\}$

$\text{ample}(n_0m_1\neg b) = \{\alpha_1, \gamma_0\}$

$\text{ample}(m_0n_1b) = \{\alpha_0, \gamma_1\}$: cycle condition (A4)

Nested DFS with POR

Nested DFS (standard approach)

remind: nested DFS for checking "$\mathcal{T} \models \Diamond\Box a$?" uses:

outer DFS: visits all reachable states

inner DFS: CYCLE_CHECK(s) searches for a backward edge $s' \rightarrow s$

Nested DFS (standard approach)

remind: nested DFS for checking "$\mathcal{T} \models \Diamond\Box a$?" uses:

outer DFS: visits all reachable states

inner DFS: CYCLE_CHECK(s) searches for a backward edge $s' \to s$

CYCLE_CHECK(s)

- is called for each state s that violates the persistence condition a

Nested DFS (standard approach)

remind: nested DFS for checking "$\mathcal{T} \models \Diamond\Box a$?" uses:

outer DFS: visits all reachable states

inner DFS: CYCLE_CHECK(s) searches for a backward edge $s' \rightarrow s$

CYCLE_CHECK(s)

- is called for each state s that violates the persistence condition a
- must not be started before the outer DFS is finished for s

Nested DFS

outer DFS: visits all reachable states

inner DFS: CYCLE_CHECK(s) searches for a backward edge $s' \longrightarrow s$

CYCLE_CHECK(s)

- is called for each state s that violates the persistence condition a
- must not be started before the outer DFS is finished for s
- early termination

Nested DFS

outer DFS: visits all reachable states

inner DFS: CYCLE_CHECK(s) searches for a backward edge $s' \to s$

CYCLE_CHECK(s)

- is called for each state s that violates the persistence condition a
- must not be started before the outer DFS is finished for s
- early termination, e.g., abort with the answer

  $$\text{CYCLE\_CHECK}(s) = \textit{true}$$

  as soon as the inner DFS visits a state in the DFS-stack of the outer DFS

Nested DFS with POR

requirement for the nested DFS in the ample set approach:

Nested DFS with POR

requirement for the nested DFS in the ample set approach:

outer DFS and inner DFS must use the same ample-sets

Nested DFS with POR

requirement for the nested DFS in the ample set approach:

outer DFS and inner DFS must use the same ample-sets

implementation: uses a hash-table for the set of states that have been visited in the outer DFS

Implementation of the nested DFS with POR

use *hash-table* for the set of states that have been visited in the outer DFS

Implementation of the nested DFS with POR

use *hash-table* for the set of states that have been visited in the outer DFS

entries in the hash-table have the form

$$\langle s, b, c, a_1, \ldots, a_k \rangle$$

where s is a state and $b, c, a_1, \ldots, a_k$ are bits

Implementation of the nested DFS with POR

use *hash-table* for the set of states that have been visited in the outer DFS

entries in the hash-table have the form

$$\langle s, b, c, a_1, \ldots, a_k \rangle$$

where s is a state and $b, c, a_1, \ldots, a_k$ are bits

- $b = 1$ iff s has been visited in inner DFS use *hash-table* for the set of states that have been visited in the outer DFS

entries in the hash-table have the form

$$\langle s, b, c, a_1, \ldots, a_k \rangle$$

where s is a state and $b, c, a_1, \ldots, a_k$ are bits

- $b = 1$ iff s has been visited in inner DFS
- $c = 1$ iff s is in the DFS stack

Implementation of the nested DFS with POR

use *hash-table* for the set of states that have been visited in the outer DFS

entries in the hash-table have the form

$$\langle s, b, c, a_1, \ldots, a_k \rangle$$

where s is a state and $b, c, a_1, \ldots, a_k$ are bits

- $b = 1$ iff s has been visited in inner DFS
- $c = 1$ iff s is in the DFS stack
- for $Act(s) = \{\alpha_1, \ldots, \alpha_k\}$:

$$a_i = 1 \text{ iff } \alpha_i \in ample(s)$$

On-the-fly construction of $\mathcal{T}_{\mathsf{red}}$

On-the-fly construction of $\mathcal{T}_{\mathsf{red}}$

starting point: syntactic description of the processes $P_1, \ldots, P_n$ of a parallel system

On-the-fly construction of $\mathcal{T}_{\text{red}}$

starting point: syntactic description of the processes
$P_1, \ldots, P_n$ of a parallel system
e.g., PROMELA-specification

On-the-fly construction of $\mathcal{T}_{\text{red}}$ in DFS-manner

starting point: syntactic description of the processes $P_1, \ldots, P_n$ of a parallel system e.g., PROMELA-specification

method: generate the reachable fragment of $\mathcal{T}_{\text{red}}$ in DFS-manner by generating ample sets by means of local conditions that ensure (A1)-(A4)

On-the-fly construction of $\mathcal{T}_{red}$ in DFS-manner

starting point: syntactic description of the processes $P_1, \ldots, P_n$ of a parallel system e.g., PROMELA-specification

method: generate the reachable fragment of $\mathcal{T}_{red}$ in DFS-manner by generating ample sets by means of local conditions that ensure (A1)-(A4)

idea: check whether

$$\text{ample}(s) = \text{set of enabled actions of process } P_i$$

fulfills (A1), (A2), (A3)

On-the-fly construction of $\mathcal{T}_{\text{red}}$ in DFS-manner

starting point: syntactic description of the processes $P_1, \ldots, P_n$ of a parallel system e.g., PROMELA-specification

method: generate the reachable fragment of $\mathcal{T}_{\text{red}}$ in DFS-manner by generating ample sets by means of local conditions that ensure (A1)-(A4)

idea: check whether

$$\text{ample}(s) = \text{set of enabled actions of process } P_i$$

fulfills (A1), (A2), (A3) and ensure (A4) by searching for backward edges in $\mathcal{T}_{\text{red}}$

Computing the ample set for state s

Computing the ample set for state s

REPEAT

select a process P_i not considered before

Computing the ample set for state s

REPEAT

select a process P_i not considered before

$A :=$ action set of $P_i \cap Act(s)$

Computing the ample set for state s

REPEAT

select a process P_i not considered before

$A :=$ action set of $P_i \cap Act(s)$

IF $A \neq \emptyset$ and (A2) is not violated

Computing the ample set for state s

<u>REPEAT</u>

select a process P_i not considered before

$A := \text{action set of } P_i \cap Act(s)$

<u>IF</u> $A \neq \emptyset$ and $(A2)$ is not violated

and all actions of A are stutter actions

Computing the ample set for state s

REPEAT

select a process P_i not considered before

$A :=$ action set of $P_i \cap Act(s)$

IF $A \neq \emptyset$ and (A2) is not violated

and all actions of A are stutter actions

THEN $ample(s) := A$ **FI**

REPEAT

select a process P_i not considered before

$A := \text{action set of } P_i \cap Act(s)$

IF $A \neq \emptyset$ and (A2) is not violated

and all actions of A are stutter actions

THEN $ample(s) := A$ **FI**

UNTIL all processes have been considered

or $ample(s)$ is defined;

Computing the ample set for state s

REPEAT

select a process P_i not considered before

$A :=$ action set of $P_i \cap Act(s)$

IF $A \neq \emptyset$ and $(A2)$ is not violated

and all actions of A are stutter actions

THEN $ample(s) := A$ **FI**

UNTIL all processes have been considered

or $ample(s)$ is defined;

IF $ample(s)$ is not yet defined

THEN $ample(s) := Act(s)$ **FI**

Computing the ample set for state s

<u>REPEAT</u>

select a process P_i not considered before

$A :=$ action set of $P_i \cap Act(s)$

<u>IF</u> (A1), (A2), (A3) hold **<u>THEN</u>** ample(s) $:= A$ **<u>FI</u>**

<u>UNTIL</u> all processes have been considered

or ample(s) is defined;

<u>IF</u> ample(s) is not yet defined

<u>THEN</u> ample(s) $:= Act(s)$ **<u>FI</u>**

... consider state $\alpha(s)$ for some $\alpha \in$ ample(s) ...

REPEAT

select a process P_i not considered before

$A :=$ action set of $P_i \cap Act(s)$

IF (A1), (A2), (A3) hold **THEN** $ample(s) := A$ **FI**

UNTIL all processes have been considered

or $ample(s)$ is defined;

IF $ample(s)$ is not yet defined

THEN $ample(s) := Act(s)$ **FI**

... consider state $\alpha(s)$ for some $\alpha \in ample(s)$ ...

IF the expansion of s finds a backwards edge $s' \Longrightarrow s$

Computing the ample set for state s

<u>REPEAT</u>

select a process P_i not considered before

A := action set of $P_i \cap Act(s)$

<u>IF</u> (A1), (A2), (A3) hold **<u>THEN</u>** ample(s) := A **<u>FI</u>**

<u>UNTIL</u> all processes have been considered

or ample(s) is defined;

<u>IF</u> ample(s) is not yet defined

<u>THEN</u> ample(s) := Act(s) **<u>FI</u>**

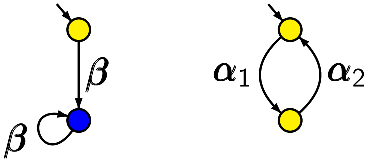
... consider state α(s) for some $\alpha \in$ ample(s) ...

<u>IF</u> the expansion of s finds a backwards edge $s' \Longrightarrow s$

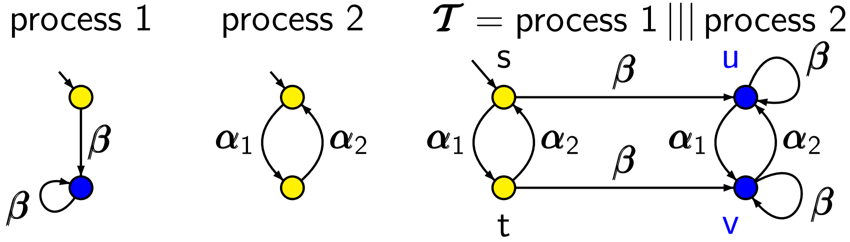
<u>THEN</u> ample(s) := Act(s) **<u>FI</u>**

Example: construction of $\mathcal{T}_{\text{red}}$

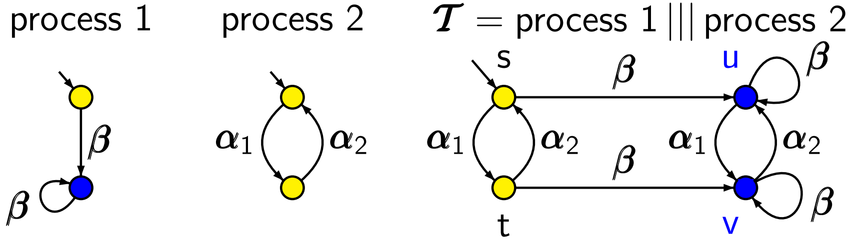
process 1 process 2



Example: construction of $\mathcal{T}_{\mathsf{red}}$



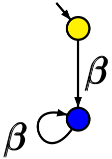
Example: construction of $\mathcal{T}_{\text{red}}$



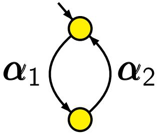
DFS(s)

Example: construction of $\mathcal{T}_{\text{red}}$

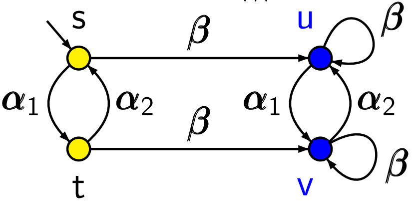
process 1



process 2

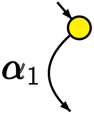


$\mathcal{T}$ = process 1 ||| process 2



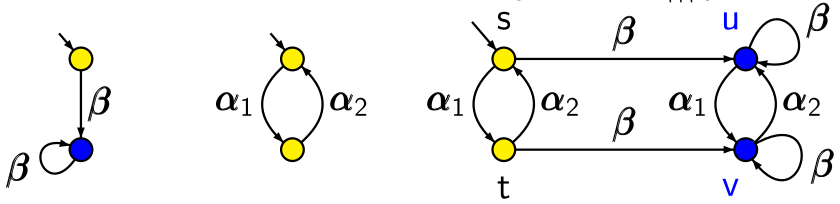
DFS(s)

ample(s) = $\{\alpha_1\}$

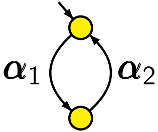


Example: construction of $\mathcal{T}_{\text{red}}$

process 1 process 2 $\mathcal{T}$ = process 1 ||| process 2

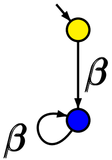


DFS(s)
  ample(s) = $\{\alpha_1\}$
  DFS(t)
    ample(t) = $\{\alpha_2\}$

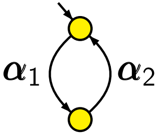


Example: construction of $\mathcal{T}_{\mathsf{red}}$

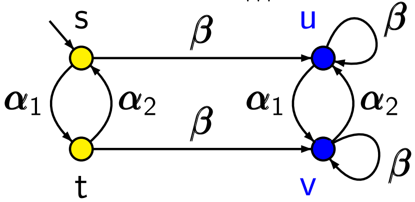
process 1



process 2



$\mathcal{T}$ = process 1 ||| process 2



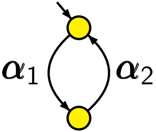
DFS(s)

ample(s) $= \{\alpha_1\}$

DFS(t)

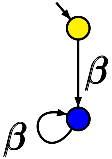
ample(t) $= \{\alpha_2\}$

backward edge t $\longrightarrow$ s

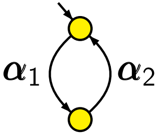


Example: construction of $\mathcal{T}_{\text{red}}$

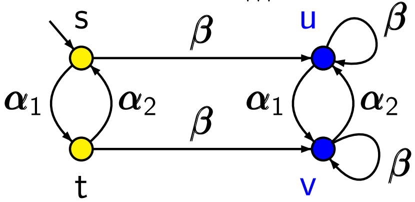
process 1



process 2



$\mathcal{T}$ = process 1 ||| process 2



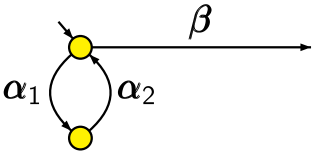
DFS(s)

ample(s) = $\{\alpha_1\} \cup \{\beta\}$

DFS(t)

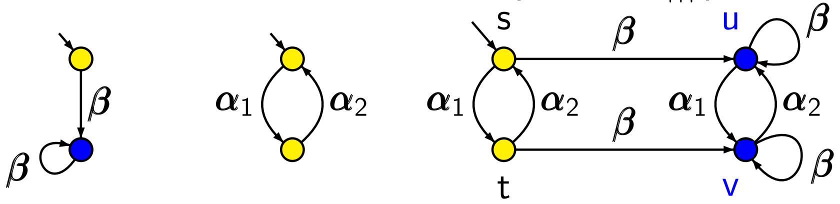
ample(t) = $\{\alpha_2\}$

backward edge t $\longrightarrow$ s



Example: construction of $\mathcal{T}_{\text{red}}$

process 1   process 2 $\mathcal{T}$ = process 1 ||| process 2



DFS(s)

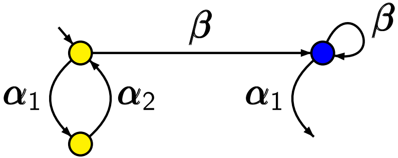
ample(s) = $\{\alpha_1\} \cup \{\beta\}$

DFS(t)

ample(t) = $\{\alpha_2\}$

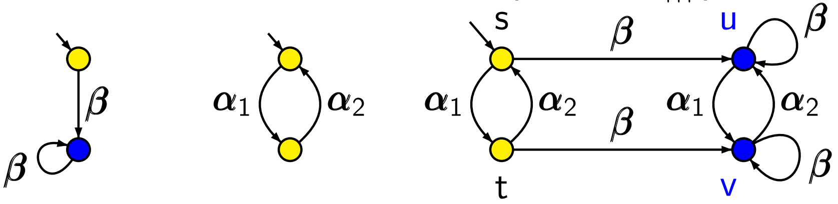
backward edge t $\longrightarrow$ s

DFS(u) ...

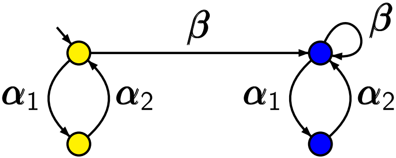


Example: construction of $\mathcal{T}_{red}$

process 1 process 2 $\mathcal{T}$ = process 1 ||| process 2



DFS(s)
- ample(s) $= \{\alpha_1\} \cup \{\beta\}$
- DFS(t)
  - ample(t) $= \{\alpha_2\}$
  - backward edge t $\longrightarrow$ s
- DFS(u) ...
  - DFS(v) ...



Computing the ample set for state s

REPEAT

select a process P_i not considered before

$A :=$ action set of $P_i \cap Act(s)$

IF $A \neq \emptyset$ and $(A2)$ holds

and all actions of A are stutter actions

THEN ample(s) $:= A$ **FI**

UNTIL all processes have been considered

or ample(s) is defined;

IF ample(s) is not yet defined

THEN ample(s) $:= Act(s)$ **FI**

Checking the dependence condition (A2)?

REPEAT

select a process P_i not considered before

$A := \text{action set of } P_i \cap \mathit{Act}(s)$

IF $A \neq \emptyset$ and (A2) holds

and all actions of A are stutter actions

THEN $\text{ample}(s) := A$ **FI**

UNTIL all processes have been considered

or $\text{ample}(s)$ is defined;

IF $\text{ample}(s)$ is not yet defined

THEN $\text{ample}(s) := \mathit{Act}(s)$ **FI**

Checking the dependence condition (A2)?

(A1) nonemptiness condition

(A2) dependence 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 $\beta_i \in$ ample(s)

(A3) stutter condition

(A4) cycle condition

Checking the dependence condition (A2)?

(A1) nonemptiness condition

(A2) dependence 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 $\beta_i \in$ ample(s)

(A3) stutter condition

(A4) cycle condition

checking (A2) is as hard as the reachability problem

Checking the dependence condition (A2)?

(A1) nonemptiness condition

(A2) dependence 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 $\beta_i \in$ ample(s)

(A3) stutter condition

(A4) cycle condition

checking (A2) is as hard as the unreachability problem

given: finite transition system $\mathcal{T}$, $a \in AP$

question: does $\mathcal{T} \not\models \exists\Diamond a$ hold?

(A1) nonemptiness condition

(A2) dependence condition: ⟵ global 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 $\beta_i \in$ ample(s)

(A3) stutter condition

(A4) cycle condition

checking (A2) is as hard as the unreachability problem

given: finite transition system $\mathcal{T}$, $a \in AP$

question: does $\mathcal{T} \not\models \exists \Diamond a$ hold?

Algorithmic difficulty of checking (A2)

show that the unreachability problem

given: finite transition system $\mathcal{T}$
$\mathbf{a} \in \mathsf{AP}$

question: does $\mathcal{T} \not\models \exists\Diamond \mathbf{a}$ hold?

is polynomially reducible to the problem of checking (A2)

Algorithmic difficulty of checking (A2)

show that the unreachability problem

given: finite transition system $\mathcal{T}$

$a \in AP$

question: does $\mathcal{T} \not\models \exists\Diamond a$ hold?

is polynomially reducible to the problem of checking (A2)

given: finite transition system $\mathcal{T}'$, ample sets for $\mathcal{T}'$

question: does (A2) hold?

i.e., does 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 $\beta_i \in$ ample(s)?

Algorithmic difficulty of checking $(A2)$

show that the unreachability problem

given: finite transition system $\mathcal{T}$ and initial state s_0
$\mathbf{a} \in AP$

question: does $s_0 \not\models \exists\Diamond \mathbf{a}$ hold?

is polynomially reducible to the problem of checking $(A2)$

given: finite transition system $\mathcal{T}'$, ample sets for $\mathcal{T}'$

question: does $(A2)$ hold?
i.e., does 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 $\beta_i \in$ ample(s)?

Algorithmic difficulty of checking (A2)

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

Algorithmic difficulty of checking (A2)

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

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

Algorithmic difficulty of checking (A2)

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

Algorithmic difficulty of checking $(A2)$

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

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	$(A2)$ holds

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

Algorithmic difficulty of checking $(A2)$

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

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 $(A2)$ holds

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

- α are β are dependent

Algorithmic difficulty of checking (A2)

unreachability problem	$\leq_{\text{poly}}$	problem of checking (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	(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 $\mathcal{T}$

Algorithmic difficulty of checking (A2)

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

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

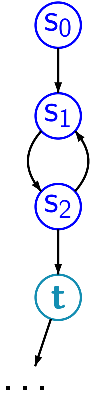
s.t. $s_0 \not\models \exists\Diamond a$ iff (A2) holds

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

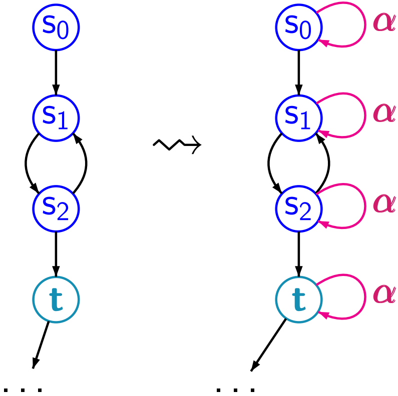
- α are β are dependent
- α is independent from all actions in $\mathcal{T}$
- β is enabled exactly in the states t with $t \models a$

	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

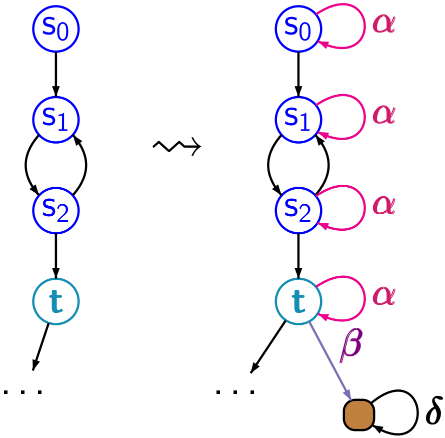
	finite TS $\mathcal{T}$ + state s_0 + atomic prop. a	$\rightsquigarrow$	finite TS $\mathcal{T}'$ + ample sets
s.t.	$s_0 \not\models \exists\Diamond a$	iff	(A2) holds



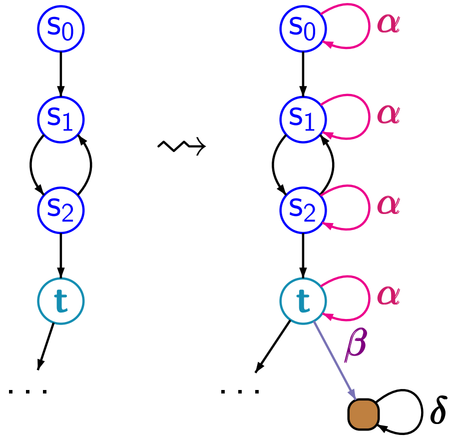
	finite TS $\mathcal{T}$ + state s_0 + atomic prop. a	$\rightsquigarrow$	finite TS $\mathcal{T}'$ + ample sets
s.t.	$s_0 \not\models \exists\Diamond a$	iff	$(A2)$ holds



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

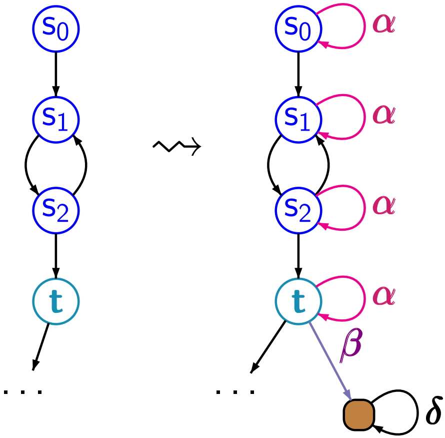


	finite TS $\mathcal{T}$ + state s_0		finite TS $\mathcal{T}'$
	+ atomic prop. a	$\rightsquigarrow$	+ ample sets
s.t.	$s_0 \not\models \exists\Diamond a$	iff	(A2) holds



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

Local criterion for condition $(\mathrm{A2})$

idea: replace the global dependency condition $(\mathrm{A2})$ by a stronger local condition

Local criterion for condition (A2)

idea: replace the global dependency condition (A2) by a stronger local condition that can be derived from the syntactic description for the processes P_i of the given parallel system

$$P_1 \| \ldots \| P_n$$

Local criterion for condition (A2)

idea: replace the global dependency condition (A2) by a stronger local condition that can be derived from the syntactic description for the processes P_i of the given parallel system

$$P_1 \| \ldots \| P_n$$

e.g., the P_i's are given as program graphs of a channel system.

Local criterion for condition (A2)

idea: replace the global dependency condition (A2) by a stronger local condition that can be derived from the syntactic description for the processes P_i of the given parallel system

$$P_1 \| \ldots \| P_n$$

e.g., the P_i's are given as program graphs of a channel system. Then: each state s has the form

$$s = \langle \ell_1, \ldots, \ell_n, \eta, \xi \rangle$$

where ℓ_i is a location of process P_i, η a variable evaluation, ξ a channel evaluation

Local criterion for condition (A2)

Let Act_i denote the set of actions of process P_i.

Local criterion for condition (A2)

Let Act_i denote the set of actions of process P_i.
For state s:

$$\begin{aligned} Act_i(s) &= Act_i \cap Act(s) \\ &= \text{set of actions of process } P_i \\ &\quad\ \text{that are enabled in } s \end{aligned}$$

Local criterion for condition (A2)

Let Act_i denote the set of actions of process P_i.
For state s:

$$\begin{aligned} Act_i(s) &= Act_i \cap Act(s) \\ &= \text{set of actions of process } P_i \\ &\quad\ \text{that are enabled in } s \end{aligned}$$

Provide local criteria such that $ample(s) = Act_i(s)$ fulfills the dependency condition (A2)

Local criterion for condition (A2)

Let $s = \langle \boldsymbol{\ell}_1, \ldots, \boldsymbol{\ell}_{i-1}, \boldsymbol{\ell}_i, \boldsymbol{\ell}_{i+1}, \ldots, \boldsymbol{\ell}_n, \ldots \rangle$.
Suppose that

Local criterion for condition (A2)

Let $s = \langle \boldsymbol{\ell}_1, \ldots, \boldsymbol{\ell}_{i-1}, \boldsymbol{\ell}_i, \boldsymbol{\ell}_{i+1}, \ldots, \boldsymbol{\ell}_n, \ldots \rangle$.
Suppose that

(A2.1) all actions of P_j, $j \neq i$, are independent from $Act_i(s)$

Local criterion for condition $(A2)$

Let $s = \langle \boldsymbol{\ell}_1, \ldots, \boldsymbol{\ell}_{i-1}, \boldsymbol{\ell}_i, \boldsymbol{\ell}_{i+1}, \ldots, \boldsymbol{\ell}_n, \ldots \rangle$.
Suppose that

$(A2.1)$ all actions of P_j, $j \neq i$, are independent from $Act_i(s)$, i.e., if $\gamma \in Act_j$ for some $j \neq i$, and $\alpha \in Act_i(s)$ then α and γ are independent

Local criterion for condition (A2)

Let $s = \langle \ell_1, \ldots, \ell_{i-1}, \ell_i, \ell_{i+1}, \ldots, \ell_n, \ldots \rangle$.
Suppose that

(A2.1) all actions of P_j, $j \neq i$, are independent from $Act_i(s)$, i.e., if $\gamma \in Act_j$ for some $j \neq i$, and $\alpha \in Act_i(s)$ then α and γ are independent

(A2.2) there is no action γ of a process P_j where $j \neq i$ s.t.

Local criterion for condition (A2)

Let $s = \langle \ell_1, \ldots, \ell_{i-1}, \ell_i, \ell_{i+1}, \ldots, \ell_n, \ldots \rangle$.
Suppose that

(A2.1) all actions of P_j, $j \neq i$, are independent from $Act_i(s)$, i.e., if $\gamma \in Act_j$ for some $j \neq i$, and $\alpha \in Act_i(s)$ then α and γ are independent

(A2.2) there is <u>no</u> action γ of a process P_j where $j \neq i$ s.t. γ can enable an action $\beta \in Act_i \setminus Act(s)$ from some state s' with location ℓ_i for process P_i

Local criterion for condition (A2)

Let $s = \langle \ldots, \ell_j, \ldots, \ell_i, \ldots, \ell_n, \ldots \rangle$.
Suppose that

(A2.1) all actions of P_j, $j \neq i$, are independent from $Act_i(s)$, i.e., if $\gamma \in Act_j$ for some $j \neq i$, and $\alpha \in Act_i(s)$ then α and γ are independent

(A2.2) there is no action γ of a process P_j where $j \neq i$ s.t.

$$\langle \ldots h_j \ldots \ell_i \ldots \rangle \xrightarrow{\gamma} \langle \ldots k_j \ldots \ell_i \ldots \rangle \xrightarrow{\beta}$$
$$\beta \not\downarrow$$

for some $\beta \in Act_i \setminus Act(s)$

Local criterion for condition (A2)

Let $s = \langle \ldots, \ell_j, \ldots, \ell_i, \ldots, \ell_n, \ldots \rangle$.
Suppose that

(A2.1) all actions of P_j, $j \neq i$, are independent from $Act_i(s)$, i.e., if $\gamma \in Act_j$ for some $j \neq i$, and $\alpha \in Act_i(s)$ then α and γ are independent

(A2.2) there is no action γ of a process P_j where $j \neq i$ s.t.

$$\langle \ldots h_j \ldots \ell_i \ldots \rangle \xrightarrow{\gamma} \langle \ldots k_j \ldots \ell_i \ldots \rangle \xrightarrow{\beta}$$
$$\beta \not\downarrow$$

for some $\beta \in Act_i \setminus Act(s)$

Then (A2) holds for $ample(s) = Act_i(s)$.

Heuristics for condition (A2)

⋮ expansion of state $s = \langle \ldots \ell_j \ldots \ell_i \ldots \rangle$

Heuristics for condition (A2)

LTL3.4-45

$\vdots$ expansion of state $s = \langle \ldots \ell_j \ldots \ell_i \ldots \rangle$

$A := Act_i(s)$

Heuristics for condition (A2)

$\vdots$ expansion of state $s = \langle \ldots \ell_j \ldots \ell_i \ldots \rangle$

$A := Act_i(s)$

$\vdots$

check if for all other processes P_j the following holds:

Heuristics for condition (A2)

$\vdots$ expansion of state $s = \langle \ldots \ell_j \ldots \ell_i \ldots \rangle$

$A := Act_i(s)$

$\vdots$

check if for all other processes P_j the following holds:

(A2.1) all actions of P_j are independent from A

Heuristics for condition $(A2)$

$\vdots$ expansion of state $s = \langle \ldots \ell_j \ldots \ell_i \ldots \rangle$

$A := Act_i(s)$

$\vdots$

check if for all other processes P_j the following holds:

$(A2.1)$ all actions of P_j are independent from A

$(A2.2)$ there is no action γ of P_j such that

Heuristics for condition (A2)

$\vdots$ expansion of state $s = \langle \ldots \ell_j \ldots \ell_i \ldots \rangle$

$A := Act_i(s)$

$\vdots$

check if for all other processes P_j the following holds:

(A2.1) all actions of P_j are independent from A

(A2.2) there is no action γ of P_j such that

$$\langle \ldots h_j \ldots \ell_i \ldots \rangle \xrightarrow{\gamma} \langle \ldots k_j \ldots \ell_i \ldots \rangle \xrightarrow{\beta}$$

$$\beta \not\downarrow$$

for some $\beta \in Act_i \setminus A$

Heuristics for condition (A2)

$\vdots$ expansion of state $s = \langle \dots \ell_j \dots \ell_i \dots \rangle$

$A := Act_i(s)$

$\vdots$

check if for all other processes P_j the following holds:

(A2.1) all actions of P_j are independent from A

(A2.2) there is no action γ of P_j such that

$$\langle \dots h_j \dots \ell_i \dots \rangle \xrightarrow{\gamma} \langle \dots k_j \dots \ell_i \dots \rangle \xrightarrow{\beta}$$

$$\beta \not\downarrow$$

for some $\beta \in Act_i \setminus A$

if yes then set $ample(s) := A$

$\vdots$

Correct or wrong?

Let $\mathcal{T}_1, \mathcal{T}_2$ be transition systems with $\mathcal{T}_1 \triangleq \mathcal{T}_2$, and let fair be an **LTL** fairness assumption.

Remind: $\triangleq$ denotes stutter trace equivalence.
E.g., $\mathcal{T}_1 = \mathcal{T}$, $\mathcal{T}_2 = \mathcal{T}_{\text{red}}$

Then, for all $\textbf{LTL}_{\setminus\bigcirc}$ formulas φ:

$$\mathcal{T}_1 \models_{\text{fair}} \varphi \quad \text{iff} \quad \mathcal{T}_2 \models_{\text{fair}} \varphi$$

Correct or wrong?

Let $\mathcal{T}_1, \mathcal{T}_2$ be transition systems with $\mathcal{T}_1 \triangleq \mathcal{T}_2$, and let fair be an **LTL** fairness assumption.

Remind: $\triangleq$ denotes stutter trace equivalence.
E.g., $\mathcal{T}_1 = \mathcal{T}, \mathcal{T}_2 = \mathcal{T}_{\text{red}}$

Then, for all $\textbf{LTL}_{\setminus \bigcirc}$ formulas φ:

$$\mathcal{T}_1 \models_{\text{fair}} \varphi \quad \text{iff} \quad \mathcal{T}_2 \models_{\text{fair}} \varphi$$

correct

Correct or wrong?

Let $\mathcal{T}_1$, $\mathcal{T}_2$ be transition systems with $\mathcal{T}_1 \overset{\Delta}{=} \mathcal{T}_2$, and let fair be an **LTL** fairness assumption.

Remind: $\overset{\Delta}{=}$ denotes stutter trace equivalence.
E.g., $\mathcal{T}_1 = \mathcal{T}$, $\mathcal{T}_2 = \mathcal{T}_{red}$

Then, for all $\mathbf{LTL}_{\setminus \bigcirc}$ formulas φ:

$$\mathcal{T}_1 \models_{fair} \varphi \quad \text{iff} \quad \mathcal{T}_2 \models_{fair} \varphi$$

correct, as we have:

$$\mathcal{T}_i \models_{fair} \varphi \quad \text{iff} \quad \mathcal{T}_i \models \mathit{fair} \rightarrow \varphi$$

Correct or wrong?

Let $\mathcal{T}_1, \mathcal{T}_2$ be transition systems with $\mathcal{T}_1 \triangleq \mathcal{T}_2$, and let fair be an **LTL** fairness assumption.

Remind: $\triangleq$ denotes stutter trace equivalence.
E.g., $\mathcal{T}_1 = \mathcal{T}, \mathcal{T}_2 = \mathcal{T}_{red}$

Then, for all $\mathbf{LTL}_{\setminus\bigcirc}$ formulas φ:

$$\mathcal{T}_1 \models_{fair} \varphi \quad \text{iff} \quad \mathcal{T}_2 \models_{fair} \varphi$$

correct, as we have:

$$\mathcal{T}_i \models_{fair} \varphi \;\; \text{iff} \;\; \mathcal{T}_i \models \underbrace{fair \rightarrow \varphi}_{\mathbf{LTL}_{\setminus\bigcirc} \text{ formula}}$$