Partial-Order Reduction

Lecture #7 of Principles of Model Checking

Joost-Pieter Katoen

Software Modeling and Verification Group

affiliated to University of Twente, Formal Methods and Tools

University of Twente, October 3, 2012

Content of this lecture

- Independence of actions
  - definition, permuting and adding independent (stutter) actions
- Ample set constraints
  - definition, examples, justification, correctness
- Dynamic partial-order reduction
  - nested depth-first search + integrated POR
- Branching-time ample set approach
  - ample set constraints, correctness

Content of this lecture

⇒ Independence of actions

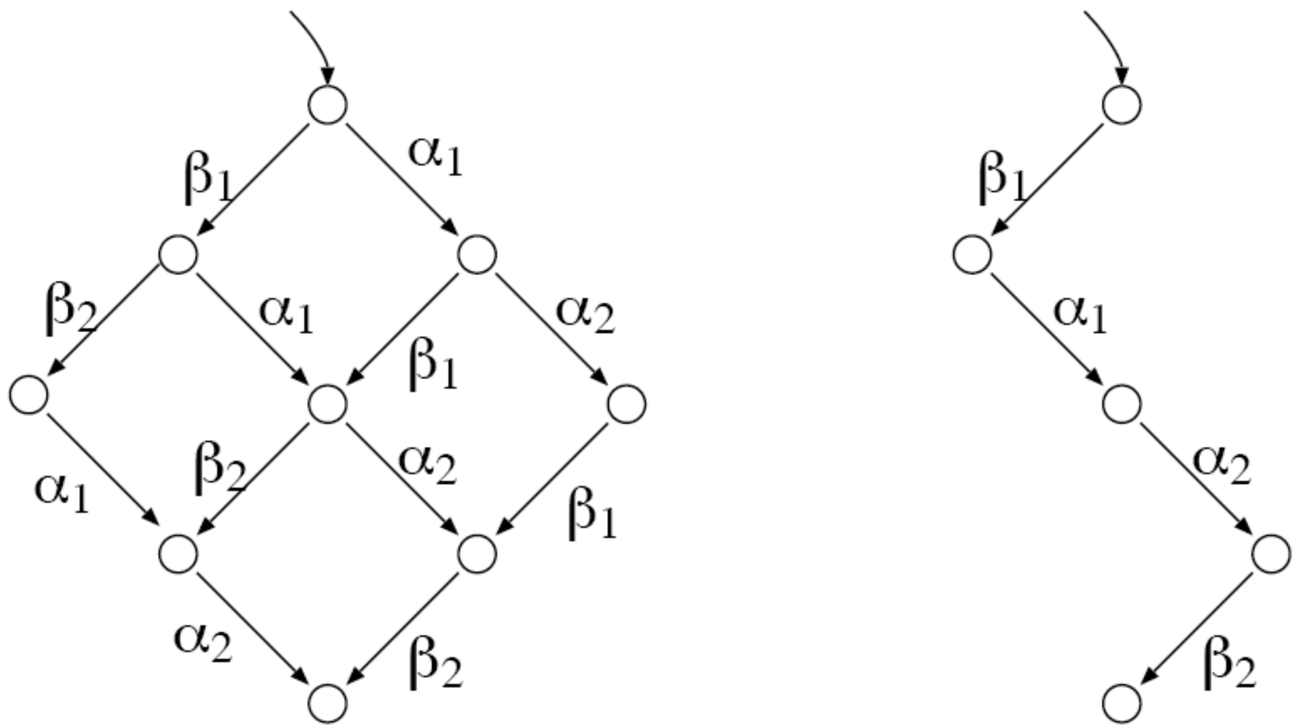
  – definition, permuting and adding independent (stutter) actions

- Ample set constraints
  – definition, examples, justification, correctness
- Dynamic partial-order reduction
  – nested depth-first search + integrated POR
- Branching-time ample set approach
  – ample set constraints, correctness

State space explosion

- Interleaving semantics
  - independent concurrent actions are interleaved
  - a run is defined by a totally ordered sequence of states
- Modeling concurrency by interleaving
  - may enforce an order of actions that has no real "meaning"
  - state space size = product of number of states of components (= explosion)
- Partial-order reduction
  - group runs for which the order of "independent" actions is irrelevant
  - consider only one representative run for equivalent runs

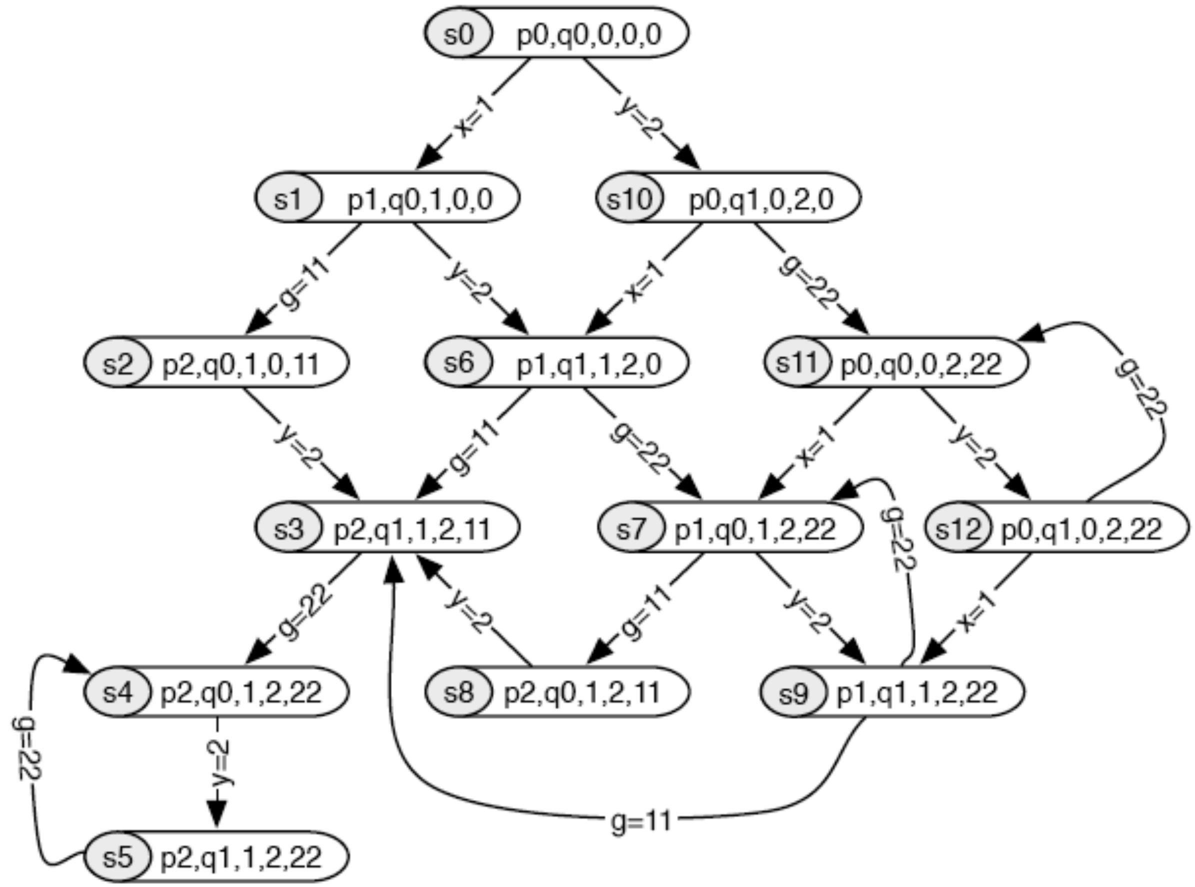
Two independent processes



A simple concurrent program

- Let x, y be local variables, g a shared variable, initially all zero
- Let φ be an LTL formula over *AP* $= \{ x > 0 \}$
- Process $P \; = \; x := 1; g := 11$
- Process $Q \; = \;$ **while** true **do** $y := 2; g := 22$ **od**
- Consider $P \parallel Q$

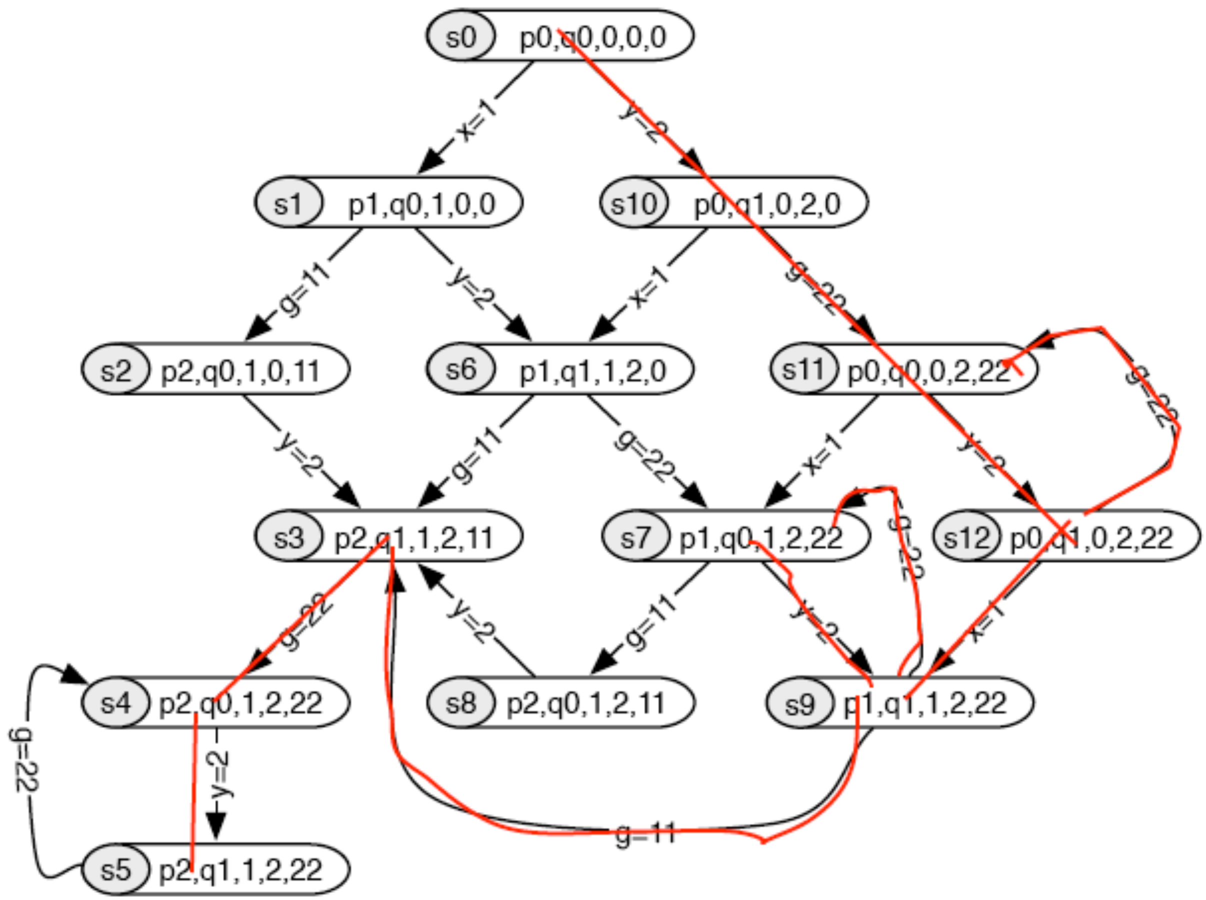
The program's transition system



Action dependencies

- Assume
  - x and y are local variables
  - g is a shared variable
- Dependent
  - $g := 11$ and $g := 22$ as they both operate on a shared variable
  - $x := 1$ and $g := 11$ as they are both executed by the same process
  - $y := 1$ and $g := 22$ as they are both executed by the same process
- Independent
  - $x := 1$ and $y := 1$
  - $x := 1$ and $g := 22$
  - $y := 1$ and $g := 11$

Reduced transition system



Outline of partial-order reduction

- During state space generation obtain reduced $\widehat{TS}$ with $\widehat{TS} \triangleq TS$

  $\Rightarrow$ this preserves all $\text{LTL}_{\setminus \bigcirc}$ formulas
  - at state s select *a (small) subset* of enabled actions in s
  - which actions to select: fulfill *ample set* constraints

- *Static* partial-order reduction

  - obtain a high-level description of $\widehat{TS}$ (without generating TS)

  $\Rightarrow$ POR is preprocessing phase of model checking

- *Dynamic (or: on-the-fly)* partial-order reduction

  - construct $\widehat{TS}$ "during" model checking
  - if accept cycle is found, there is no need to generate entire $\widehat{TS}$

Stutter equivalence

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

- Paths π_1 and π_2 are stutter equivalent, denoted $\pi_1 \triangleq \pi_2$:
  - if there exists an infinite sequence $A_0 A_1 A_2 \ldots$ with $A_i \subseteq AP$ and
  - natural numbers $n_0, n_1, n_2, \ldots, m_0, m_1, m_2, \ldots > 0$ such that:

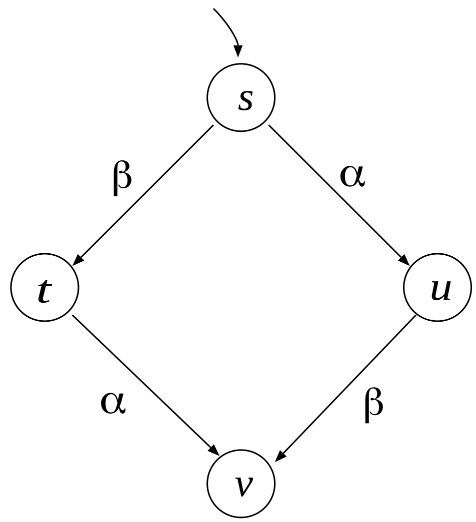
$$\begin{aligned} trace(\pi_1) &= \underbrace{A_0 \ldots A_0}_{n_0\text{-times}} \underbrace{A_1 \ldots A_1}_{n_1\text{-times}} \underbrace{A_2 \ldots A_2}_{n_2\text{-times}} \ldots \\ trace(\pi_2) &= \underbrace{A_0 \ldots A_0}_{m_0\text{-times}} \underbrace{A_1 \ldots A_1}_{m_1\text{-times}} \underbrace{A_2 \ldots A_2}_{m_2\text{-times}} \ldots \end{aligned}$$

$\Rightarrow$ $\pi_1 \triangleq \pi_2$ if both their traces are of the form $A_0{}^+ A_1{}^+ A_2{}^+ \ldots$ for $A_i \subseteq AP$

Preliminaries

- Assume from now on: *TS* is *action-deterministic*
  - for any s and action α it holds $s \xrightarrow{\alpha} u$ and $s \xrightarrow{\alpha} t$ implies $u = t$
  - action-determinism is not a severe restriction: actions can always be renamed
- *Act*(s) is the set of *enabled* actions in state s
  - *Act*$(s) = \{ \alpha \in \mathit{Act} \mid \exists s' \in S.\, s \xrightarrow{\alpha} s' \}$
- $\alpha(s)$ denotes the unique α*-successor* of s, i.e., $s \xrightarrow{\alpha} \alpha(s)$

Action independence



- performing α does not disable β, and β does not disable α
- if $\alpha, \beta \in$ *Act*(s) then $\alpha\,\beta$ and $\beta\,\alpha$ executed in s yield the same state

Action independence

Let *TS* $= (S, \mathit{Act}, \rightarrow, I, \mathit{AP}, L)$ be action-deterministic and $\alpha \neq \beta \in \mathit{Act}$

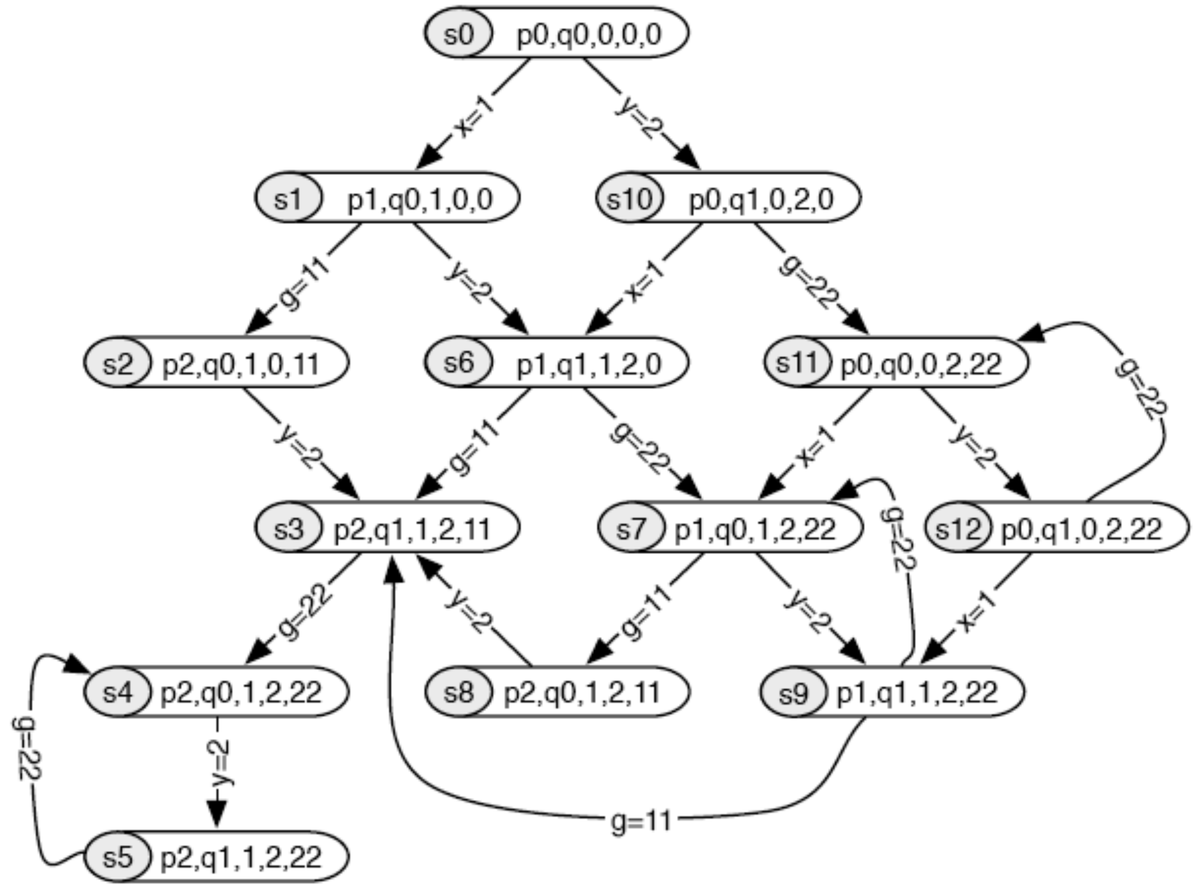
- An *independence relation* *Ind* $\subseteq S \times S$ is a irreflexive and symmetric satisfying: for any $s \in S$ with α, $\beta \in \mathit{Act}(s)$:

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

- α and β are independent if $(\alpha, \beta) \in \mathit{Ind}$, *dependent* otherwise

- For $A \subseteq \mathit{Act}$ and $\beta \in \mathit{Act} \setminus A$:
  - β is independent of A if for any $\alpha \in A$, β is independent of α
  - β depends on A otherwise

Example


s0 p0,q0,0,0,0
x=1
y=2
s1 p1,q0,1,0,0
s10 p0,q1,0,2,0
g=11
y=2
x=1
g=22
s2 p2,q0,1,0,11
s6 p1,q1,1,2,0
s11 p0,q0,0,2,22
g=22
y=2
g=11
g=22
x=1
y=2
s3 p2,q1,1,2,11
s7 p1,q0,1,2,22
g=22
s12 p0,q1,0,2,22
g=22
y=2
g=11
y=2
x=1
s4 p2,q0,1,2,22
s8 p2,q0,1,2,11
s9 p1,q1,1,2,22
g=22
y=2
s5 p2,q1,1,2,22
g=11

Permuting independent actions

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

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

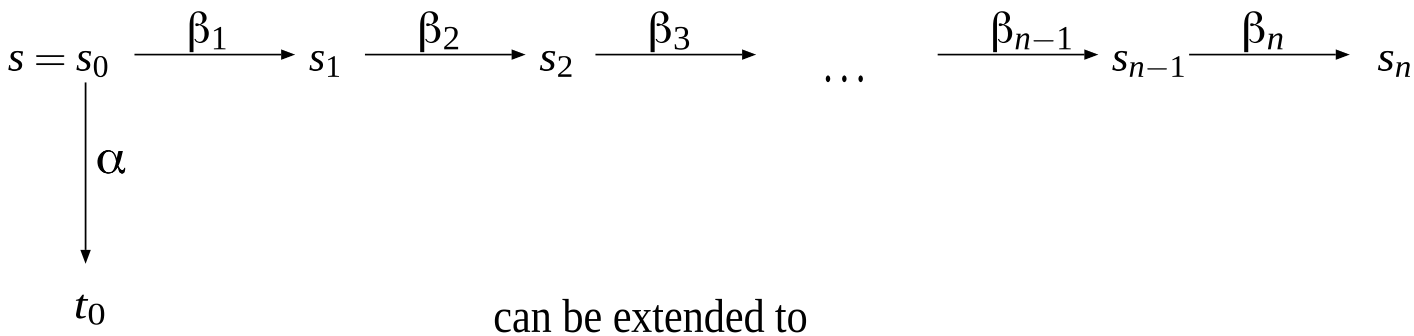
be a finite run in *TS* from s with action sequence $\beta_1 \ldots \beta_n$

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

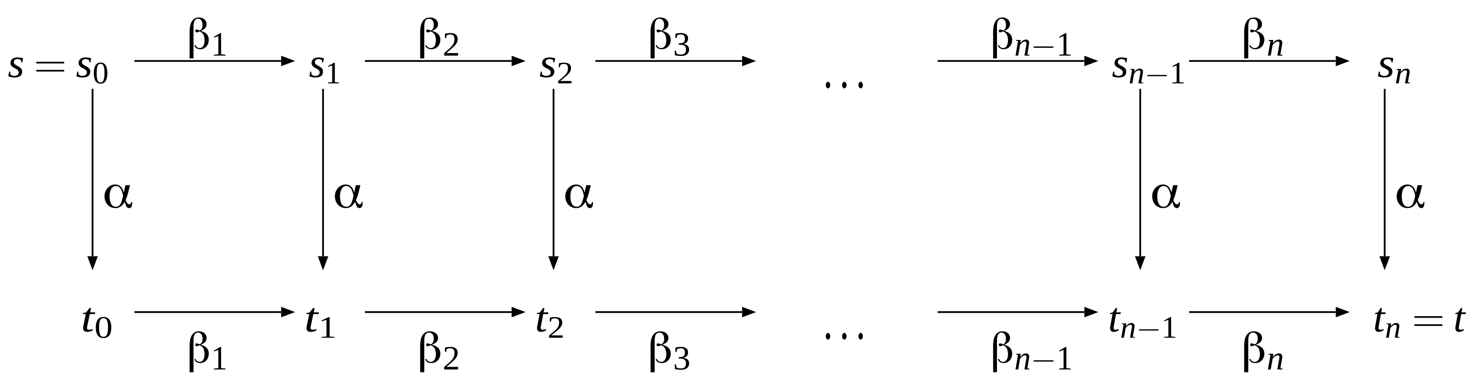
$$s = s_0 \xrightarrow{\alpha} \alpha(s_0) \xrightarrow{\beta_1} \alpha(s_1) \xrightarrow{\beta_2} \ldots \xrightarrow{\beta_{n-1}} \alpha(s_{n-1}) \xrightarrow{\beta_n} \alpha(s_n)$$

is a run in *TS* from s with action sequence $\alpha \, \beta_1 \ldots \beta_n$

Pictorially



can be extended to



Adding an independent action

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

$$s = s_0 \xrightarrow{\beta_1} s_1 \xrightarrow{\beta_2} s_2 \xrightarrow{\beta_3} \ldots$$

an infinite run in *TS* with action sequence $\beta_1\,\beta_2\,\beta_3\ldots$

Then, for $\alpha \in$ *Act*(s) independent of $\{\,\beta_1, \beta_2, \ldots\,\}$: $\forall i.\, \alpha \in$ *Act*(s_i) and:

$$s = s_0 \xrightarrow{\alpha} \alpha(s_0) \xrightarrow{\beta_1} \alpha(s_1) \xrightarrow{\beta_2} \alpha(s_2) \xrightarrow{\beta_3} \ldots$$

is an infinite run in *TS* with action sequence $\alpha\,\beta_1\,\beta_2\,\beta_3\ \ldots$

Stutter actions

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

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

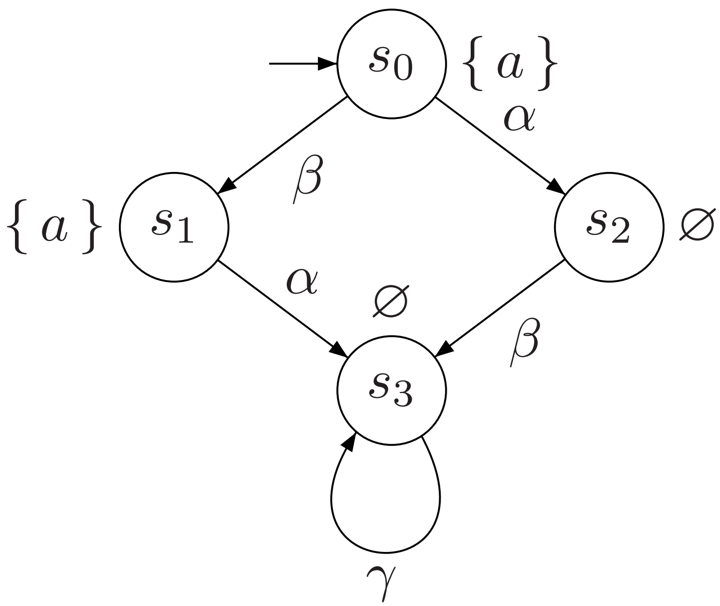
$$\rho' \;=\; s_0 \xrightarrow{\alpha} t_0 \xrightarrow{\beta_1} \ldots \xrightarrow{\beta_{n-1}} t_{n-1} \xrightarrow{\beta_n} t$$

  will be distinct!

- If α does not affect the state-labelling (= “invisible”), then $\rho \triangleq \rho'$

- $\alpha \in$ *Act* is a ***stutter action*** if for each $s \xrightarrow{\alpha} s'$ in *TS*: $L(s) = L(s')$
  - α is a stutter action whenever all transitions $s \xrightarrow{\alpha} s'$ are stutter steps

Example



s_0
$\{ a \}$
α
β
$\{ a \}$
s_1
s_2
$\varnothing$
α
$\varnothing$
β
s_3
γ

Permuting independent stutter actions

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

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

Then:

if α is a stutter action independent of $\{ \beta_1, \ldots, \beta_n \}$ then $\varrho \triangleq \varrho'$

Adding an independent stutter action

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

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

Then:

> if α is a stutter action independent of $\{ \beta_1, \beta_2, \ldots \}$ then $\rho \triangleq \rho'$

Content of this lecture

- Independence of actions
  - definition, permuting and adding independent (stutter) actions

$\Rightarrow$ Ample set constraints
  - definition, examples, justification, correctness

- Dynamic partial-order reduction
  - nested depth-first search + integrated POR

- Branching-time ample set approach
  - ample set constraints, correctness

The ample-set approach

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

- *Reduced* system $\widehat{TS} = (\widehat{S}, \mathit{Act}, \Longrightarrow, I, \mathit{AP}, L')$ where:
  - $\widehat{S}$ contains the states that are reachable (under $\Longrightarrow$) from some $s_0 \in I$
  -
$$\frac{s \xrightarrow{\alpha} s' \;\wedge\; \alpha \in \mathit{ample}(s)}{s \overset{\alpha}{\Longrightarrow} s'}$$
  - $L'(s) = L(s)$ for any $s \in \widehat{S}$

- *Constraints*: correctness ($\triangleq$), effectivity and efficiency

Which actions to select in *ample*(s)?

(A1) **Nonemptiness condition**

Select in any state in $\widehat{TS}$ at least one action.

(A2) **Dependency condition**

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

(A3) **Stutter condition**

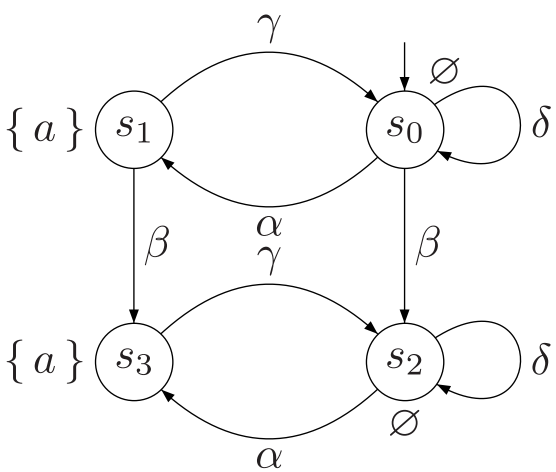
If not all actions in s are selected, then only select stutter actions in s.

(A4) **Cycle condition**

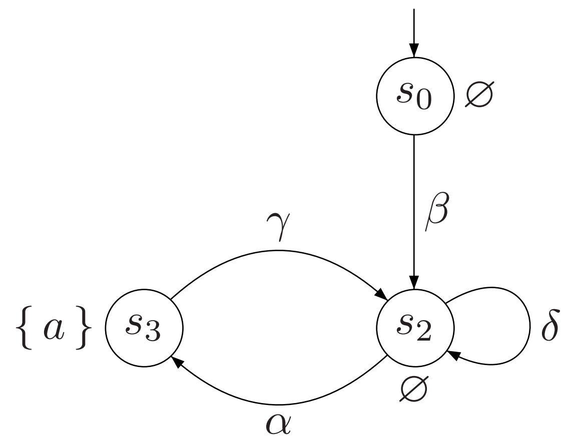
Any action in *ample*(s_i) with s_i on a cycle in $\widehat{TS}$ must be selected in some s_j on that cycle.

(A1) through (A3) apply to states in $\widehat{S}$; (A4) to cycles in $\widehat{TS}$

Example



γ
$\emptyset$
$\{a\}$
s_1
s_0
δ
α
β
γ
β
$\{a\}$
s_3
s_2
δ
$\emptyset$
α



s_0
$\emptyset$
β
γ
$\{a\}$
s_3
s_2
δ
$\emptyset$
α

Nonemptiness condition (A1)

$$\forall s \in \widehat{S}.\ (\varnothing \neq \textit{ample}(s) \subseteq \textit{Act}(s))$$

- If a state has at least one direct successor in *TS*, then it has least at one direct successor in $\widehat{TS}$

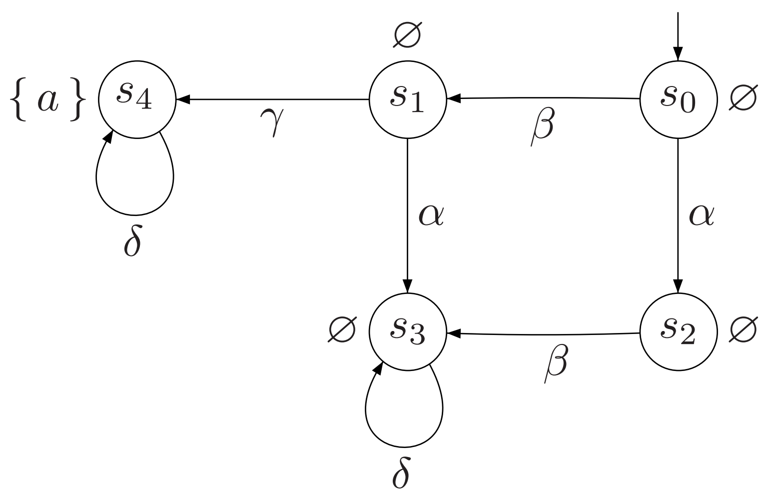
$\Rightarrow$ As *TS* has no terminal states, $\widehat{TS}$ has no terminal states

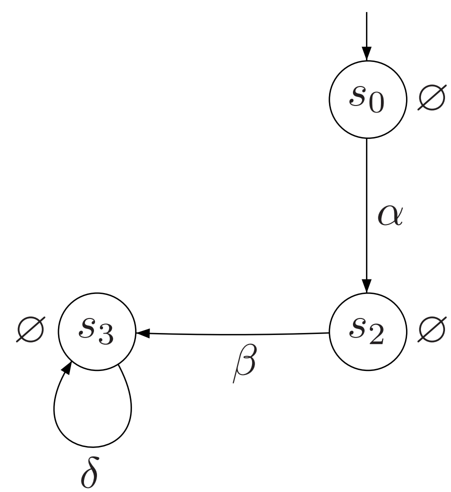
A naive dependency condition (A2')

For any $s \in \widehat{S}$, *ample*$(s) \neq$ *Act*(s)

implies $\alpha \in$ *ample*(s) is independent of *Act*$(s) \setminus$ *ample*(s)

A naive dependency condition (2)





TS $\not\models \Box \neg a$ whereas $\widehat{TS} \models \Box \neg a$, so *TS* $\not\triangleq$ $\widehat{TS}$

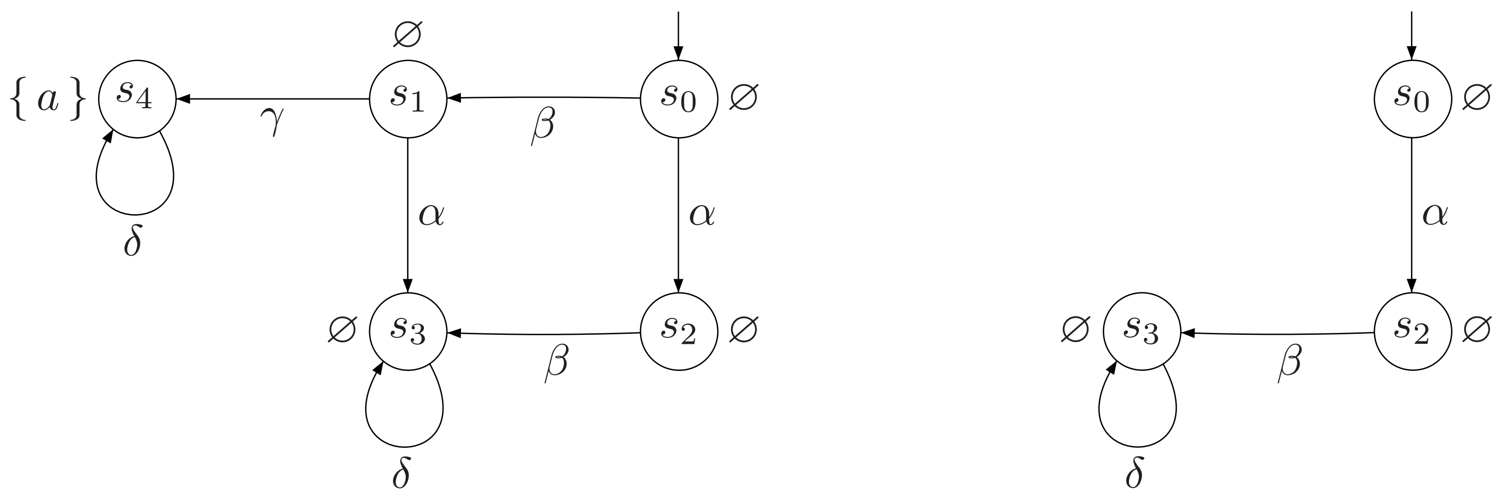
Dependency condition (A2)

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

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

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

Example



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

Properties

- (A2) guarantees that any finite run in *TS* is of the form:

$$\varrho \;=\; s_1 \xrightarrow{\beta_1} s_2 \xrightarrow{\beta_2} \ldots \xrightarrow{\beta_n} s_n \xrightarrow{\alpha} t \quad \text{with} \quad \alpha \in \textit{ample}(s)$$

  and β_i independent of *ample*(s) for $0 < i \leqslant n$.

  – if α is a stutter action: shifting α to the beginning yields an equivalent run

  $\Rightarrow$ if ϱ is pruned in *TS*, then a run is obtained by first taking α in s

- (A2) guarantees that any infinite run in *TS* is of the form:

$$s_1 \xrightarrow{\beta_1} s_2 \xrightarrow{\beta_2} \ldots \quad \text{with } \beta_i \text{ independent of } \textit{ample}(s) \text{ for } 0 < i \leqslant n.$$

  – performing stutter action $\alpha \in$ *ample*(s) in s yields an equivalent run

Properties

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

if $\textit{ample}(s)$ satisfies (A2) then α is independent of $\textit{Act}(s) \setminus \textit{ample}(s)$

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

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

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

Stutter condition (A3)

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

- All ample actions of a non-fully expanded state are stutter actions
- (A3) ensures that:
  - changing $\beta_1 \, , \ldots \, \beta_n \, \alpha$ into $\alpha \, \beta_1 \, \ldots \, \beta_n$, and
  - changing $\beta_1 \, \beta_2 \, \beta_3 \ldots$ into $\alpha \, \beta_1 \, \beta_2 \, \beta_3 \ldots$

  yields stutter-equivalent runs

Consequence of (A1) through (A3)

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

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

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

If *ample*(s) satisfies (A1) through (A3), then there exists an run ϱ':

$$s \xRightarrow{\alpha} t_0 \xrightarrow{\beta_1} t_1 \xrightarrow{\beta_2} \ldots \xrightarrow{\beta_{n-1}} t_{n-1} \xrightarrow{\beta_n} t$$

such that $\boxed{\varrho \triangleq \varrho'}$

Consequence of (A1) through (A3)

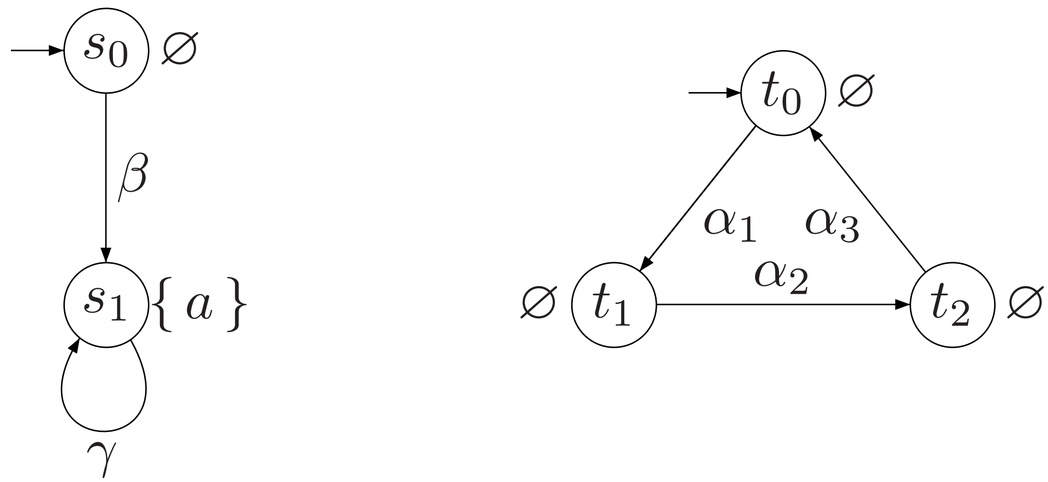
Let $\rho = s \xrightarrow{\beta_1} s_1 \xrightarrow{\beta_2} s_2 \xrightarrow{\beta_3} \ldots$ be an infinite run in *Reach*$($*TS*$)$ where $\beta_i \notin$ *ample*(s), for $i > 0$.

If *ample*(s) satisfies (A1) through (A3), then there exists a run ρ':

$$s \stackrel{\alpha}{\Longrightarrow} t_0 \xrightarrow{\beta_1} t_1 \xrightarrow{\beta_2} t_2 \xrightarrow{\beta_3} \ldots$$

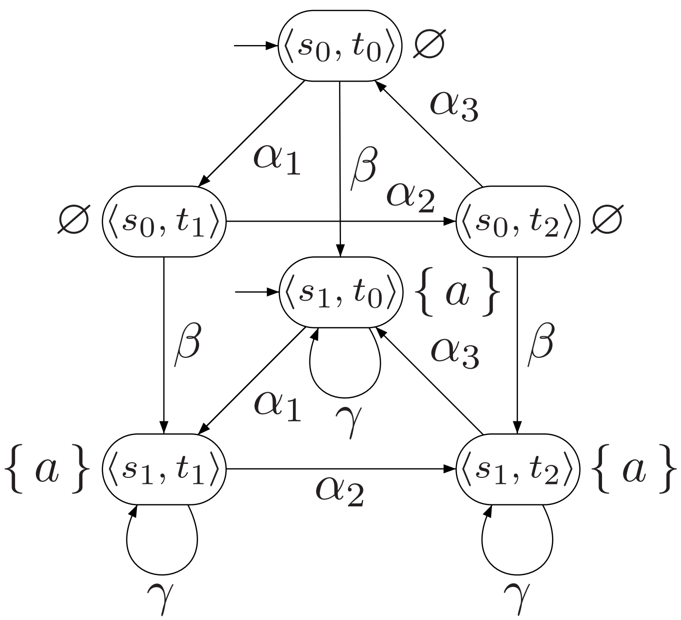
where $\alpha \in$ *ample*(s) and $\boxed{\rho \triangleq \rho'}$

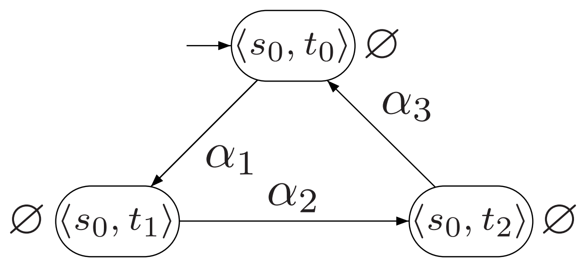
Necessity of cycle condition: example (1)



transition systems TS_1 and TS_2

Necessity of cycle condition: example (2)





$$\underbrace{TS_1 \,|||\, TS_2}_{\textit{left}} \not\models \Box \neg a \;\; \textit{but} \;\; \underbrace{\widehat{TS_1 \,|||\, TS_2}}_{\textit{right}} \models \Box \neg a$$

Cycle condition (A4)

For any cycle $s_0 \, s_1 \, \ldots \, s_n$ in $\widehat{TS}$ and $\alpha \in Act(s_i)$, for some $0 < i \leqslant n$, there exists $j \in \{ 1, \ldots, n \}$ such that $\alpha \in ample(s_j)$.

any enabled action in some state on a cycle must be selected in some state on that cycle

Overview of ample-set conditions

(A1) **Nonemptiness condition**

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

(A2) **Dependency condition**

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

(A3) **Stutter condition**

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

(A4) **Cycle condition**

For any cycle $s_0\, s_1\, \ldots\, s_n$ in $\widehat{TS}$ and $\alpha \in \textit{Act}(s_i)$, for some $0 < i \leqslant n$, there exists $j \in \{ 1, \ldots, n \}$ such that $\alpha \in \textit{ample}(s_j)$.

Correctness theorem

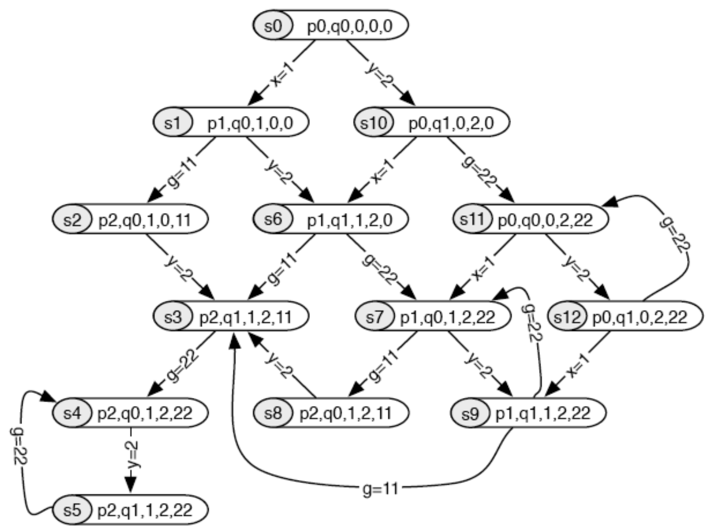
For action-deterministic, finite TS without terminal states:

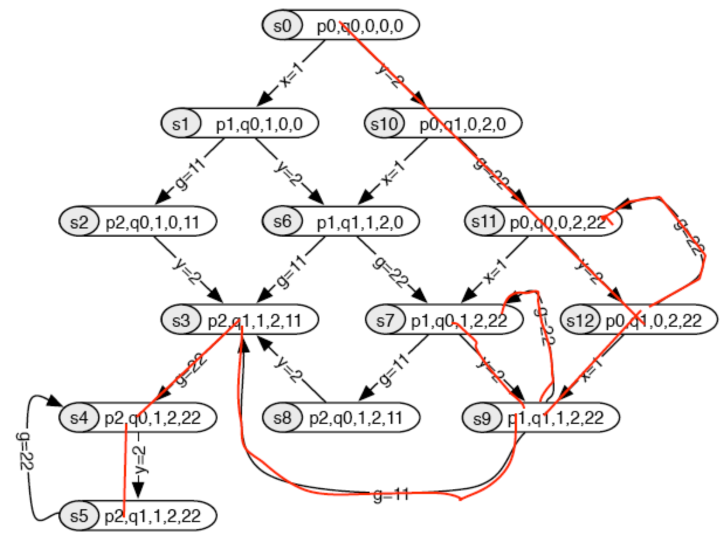
if conditions (A1) through (A4) are satisfied, then $\widehat{TS} \triangleq TS$.

as $Traces(\widehat{TS}) \subseteq Traces(TS)$, it follows $\widehat{TS} \trianglelefteq TS$

proof sketch of reverse direction in lecture notes

Reduction satisfies ample set constraints





Content of this lecture

- Independence of actions
  - definition, permuting and adding independent (stutter) actions
- Ample set constraints
  - definition, examples, justification, correctness

$\Rightarrow$ Dynamic partial-order reduction
  - nested depth-first search + integrated POR

- Branching-time ample set approach
  - ample set constraints, correctness

Strong cycle condition (A4')

On any cycle $s_0\, s_1\, \ldots\, s_n$ in $\widehat{\mathit{TS}}$,
there exists $j \in \{\, 1, \ldots, n \,\}$ such that $\mathit{ample}(s_j) = \mathit{Act}(s_j)$.

- If (A1) through (A3) hold: (A4') implies the cycle condition (A4)
- (A4') can be checked easily in DFS when backward edge is found

Invariant checking with POR

- Invariant checking
  - on state space generation, check whether each state satisfies prop. formula Φ
  - on finding a refuting state, (reversed) stack content yields counterexample

- Incorporating partial order reduction
  - on encountering a new state, compute ample set satisfying (A1) through (A3)
  - e.g., *ample*$(s) =$ *Act*(P_i), enabled actions of a concurrent process
  - enlarge *ample*(s) on demand using the strong cycle condition (A4')
  - mark actions to keep track of which actions have been taken

Example

Process 0:

ℓ_0 : **while** true $\{$

skip;

m_0 : $\textbf{wait until } (\neg b) \ \{$

n_0 : ... critical section ...$\}$

$b :=$ true;

$\}$

Process 1:

ℓ_1 : **while** true $\{$

skip;

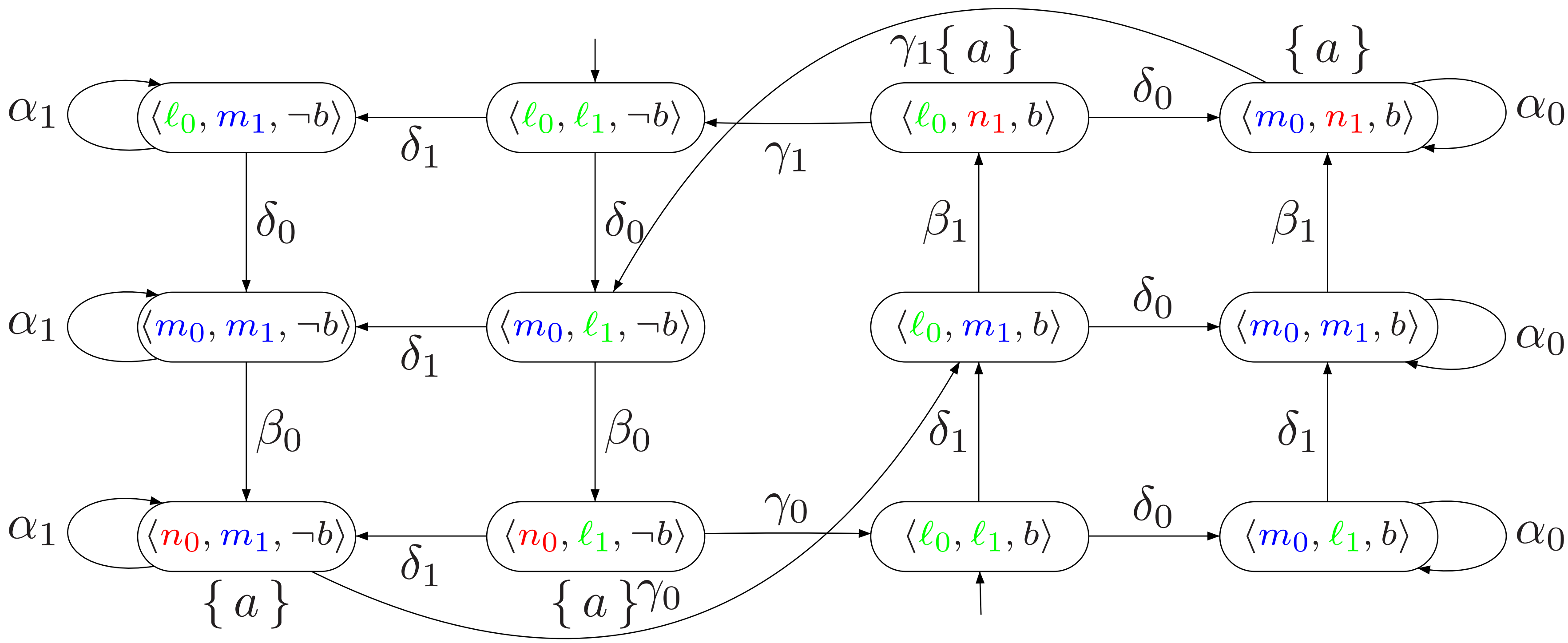
m_1 : $\textbf{wait until } (b) \ \{$

n_1 : ... critical section ...$\}$

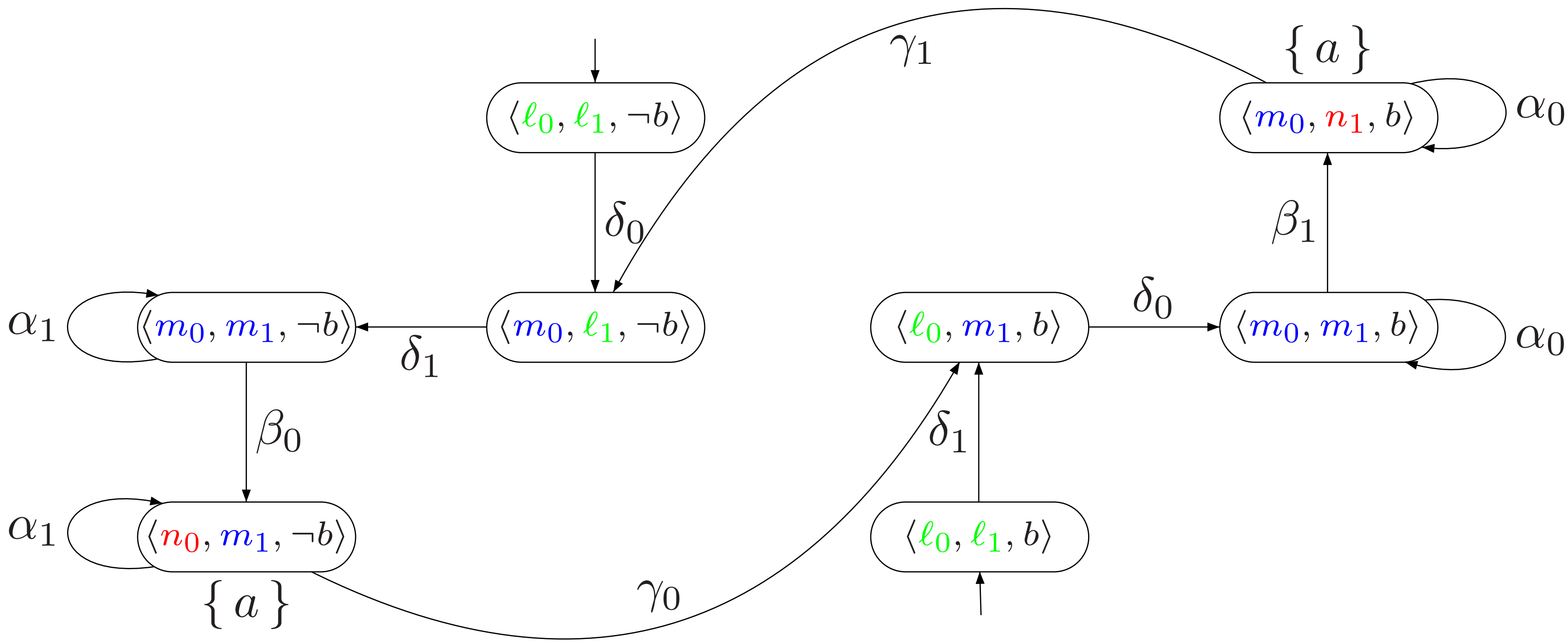
$b :=$ false;

$\}$

Transition system



Reduced transition system



Experimental results

Benchmark	TS			$\widehat{TS}$		
	states	transition	ver. time	states	transitions	ver. time
sieve	10878	35594	1.68	157	157	0.08
data transfer protocol	251049	648467	32.2	16459	17603	1.47
snoopy (cache coherence)	164258	546805	33.6	29796	44145	3.58
file transfer protocol	514188	1138750	123.4	125595	191466	18.6

partial-order reduction works fine for asynchronous systems

Checking ample set conditions

- Nonemptiness condition (A1):
  - check whether process $\mathcal{P}_i$ can perform an action in state s
- Stutter condition (A3):
  - α is a stutter action if the atomic propositions of s and $\alpha(s)$
  - do not refer to a variable that is modified by α, nor
- Strong cycle condition (A4'):
  - fully expand s if during its inner DFS a backward edge is found
- Dependency condition (A2): Hard!

Complexity of checking (A2)

The worst case time complexity of checking (A2) in finite, action-deterministic *TS* equals that of checking $TS' \models \exists \Diamond a$ for some $a \in AP$ where $size(TS') \in \mathcal{O}(size(TS))$

Overapproximating dependencies

- Actions that refer to the same variable are dependent
  - but $x := y + 1$ and $x := y + z$ are not
- Actions that modify the same variable are dependent
  - but $x := z + y$ and $x := z$ are not, if they are never enabled when $y \neq 0$
- Actions that belong to the same process are dependent
- Handshake actions depend on all actions in both processes

this yields a (conservative) dependency relation $D \subseteq$ Act $\times$ Act

Content of this lecture

- Independence of actions
  - definition, permuting and adding independent (stutter) actions
- Ample set constraints
  - definition, examples, justification, correctness
- Dynamic partial-order reduction
  - nested depth-first search + integrated POR

$\Rightarrow$ Branching-time ample set approach

  - ample set constraints, correctness

The branching-time ample approach

- Linear-time ample approach:
  - during state space generation obtain $\widehat{TS}$ such that $\widehat{TS} \triangleq TS$

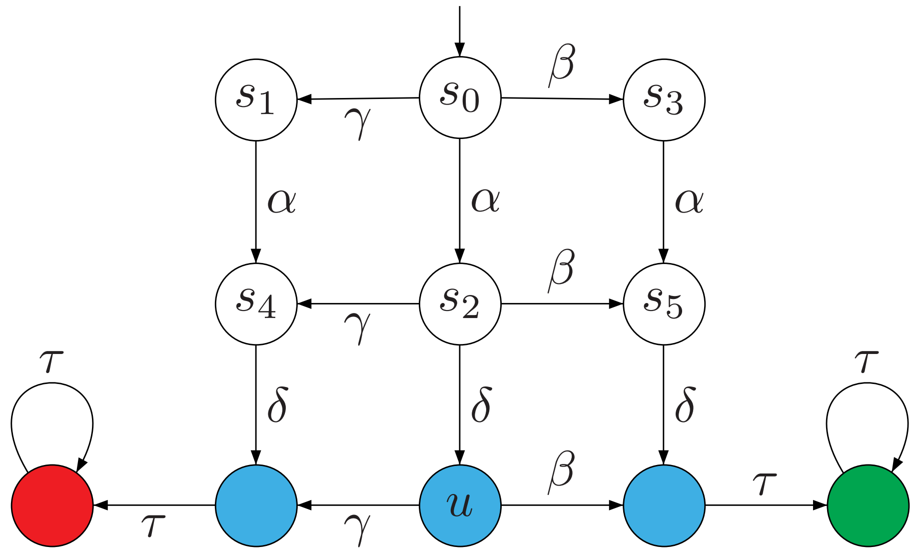
  $\Rightarrow$ this preserves all stutter sensitive LT properties, such as $\text{LTL}_{\setminus \bigcirc}$
  - static partial order reduction: generate $\widehat{TS}$ prior to verification
  - on-the-fly partial order reduction: generate $\widehat{TS}$ during the verification
  - generation of $\widehat{TS}$ by means of static analysis of program graphs

- Branching-time ample approach
  - during state space generation obtain $\widehat{TS}$ such that $\widehat{TS} \approx^{div} TS$

  $\Rightarrow$ this preserves all $\text{CTL}_{\setminus \bigcirc}$ and $\text{CTL}^*_{\setminus \bigcirc}$ formulas
  - static partial order reduction only

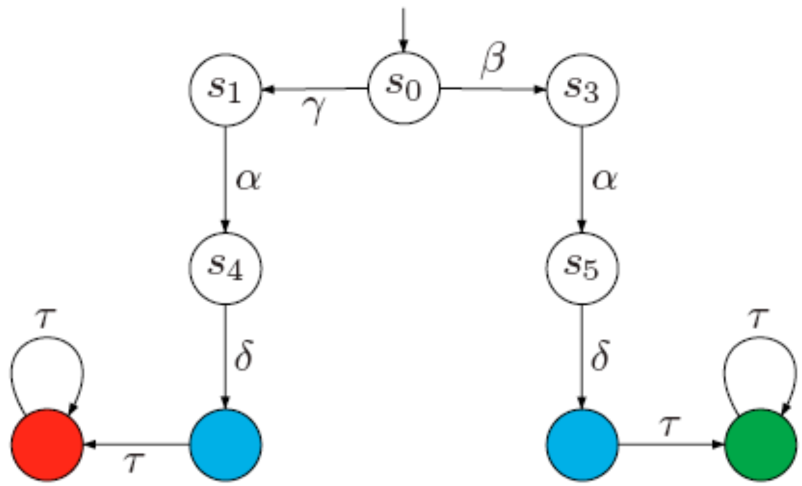
as $\approx^{div}$ is strictly finer than $\triangleq$, try (A1) through (A4)

Example



transition system *TS*, note $\alpha(s_0) \not\approx^{div} \beta(s_0) \not\approx^{div} \gamma(s_0)$

Conditions (A1)-(A4) are insufficient

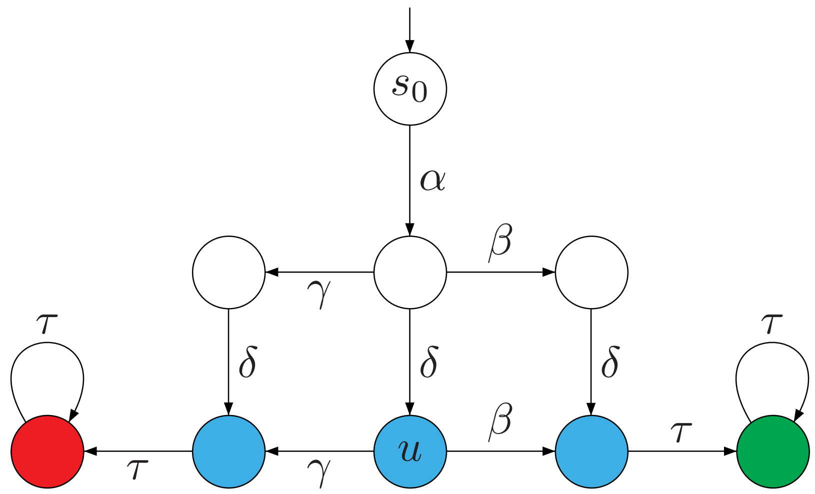


$\widehat{TS} \models \forall\Box\left(a \rightarrow (\forall\Diamond b \vee \forall\Diamond c)\right)$ but TS does not and thus $\widehat{TS} \not\approx^{div} TS$

Branching condition (A5)

If *ample*$(s) \neq$ *Act*(s) then $|$*ample*$(s)| = 1$

A sound reduction for $\mathrm{CTL}^*_{\setminus\bigcirc}$



$\widehat{TS} \not\models \forall\Box\Big(a \to (\forall\Diamond b \vee \forall\Diamond c)\Big)$ and TS does not ;in fact $\widehat{TS} \approx^{div} TS$

Correctness theorem

For action-deterministic, finite TS without terminal states:

if conditions (A1) through (A5) are satisfied, then $\widehat{TS} \approx^{div} TS$.

recall that this implies that $\widehat{TS}$ and TS are $\text{CTL}^*_{\setminus \bigcirc}$-equivalent

Ample-set conditions for CTL*

(A1) **Nonemptiness condition**

$\varnothing \neq$ *ample*$(s) \subseteq$ *Act*(s)

(A2) **Dependency condition**

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

(A3) **Stutter condition**

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

(A4) **Cycle condition**

For any cycle $s_0\, s_1\, \ldots\, s_n$ in $\widehat{TS}$ and $\alpha \in$ *Act*(s_i), for some $0 < i \leqslant n$,
there exists $j \in \{1, \ldots, n\}$ such that $\alpha \in$ *ample*(s_j).

(A5) **Branching condition**

If *ample*$(s) \neq$ *Act*(s) then $|$*ample*$(s)| = 1$