Model Checking

Lecture #17: Partial-Order Reduction

[Baier & Katoen, Chapter 8]

Joost-Pieter Katoen

Software Modeling and Verification Group

Model Checking Course, RWTH Aachen, WiSe 2019/2020

Overview

1 Motivation

2 Action Independence

3 Ample Sets

4 Correctness and Complexity

5 Summary

Overview

1 Motivation

2 Action Independence

3 Ample Sets

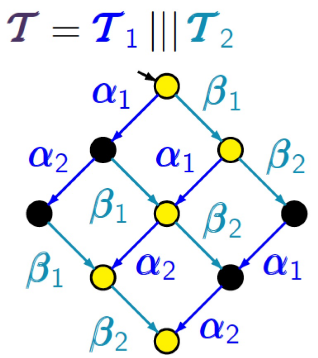
4 Correctness and Complexity

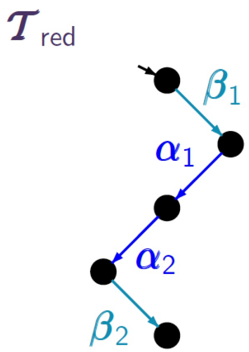
5 Summary

Motivation

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

Idea of Partial-Order Reduction





Inventors of Partial-Order Reduction

Patrice Godefroid (USA)

Pierre Wolper (Belgium)

Antti Valmari (Finland)

Doron Peled (Israel)

Outline of Ample-Set POR

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

Stutter Equivalence

Definition: stutter step

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

Definition: stutter equivalence

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

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

for $A_i \subseteq AP$.

For positive integers n_i and m_i:

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

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

Stutter Trace Equivalence

Definition: stutter trace equivalence

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

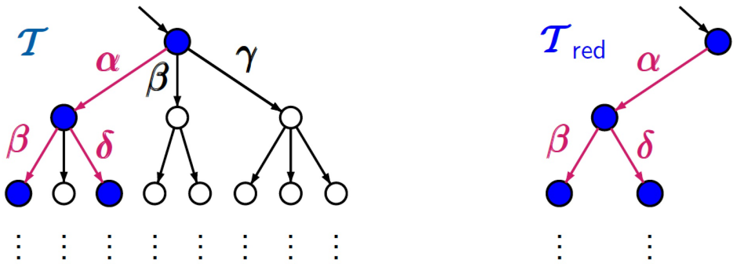
$$TS_1 \equiv_{sttrace} TS_2 \quad \text{if and only if} \quad TS_1 \trianglelefteq TS_2 \text{ and } TS_2 \trianglelefteq TS_1$$

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

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

Trace-equivalent transition systems are stutter trace-equivalent,
but not the converse.

Idea of Ample Sets

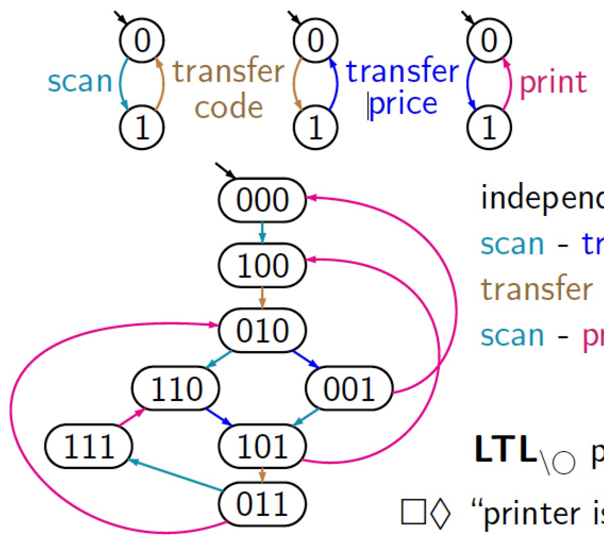


Transition relation $\Longrightarrow$ of reduced transition system TS_{red} defined by:

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

The actions outside of $ample(s)$ are pruned.

Example: Booking System

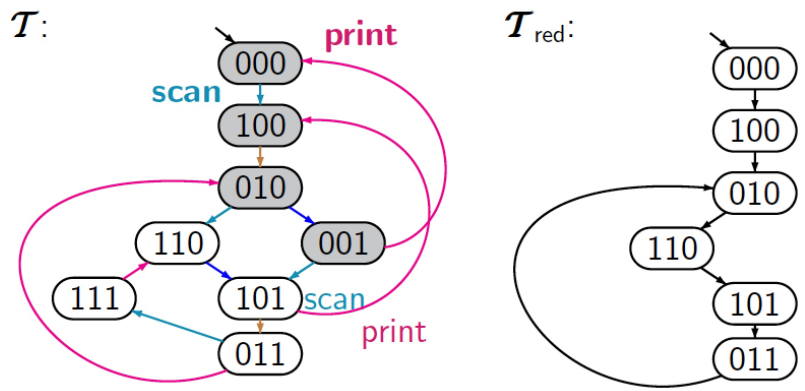


independent actions:
scan - transf. price
transfer code - print
scan - print

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

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

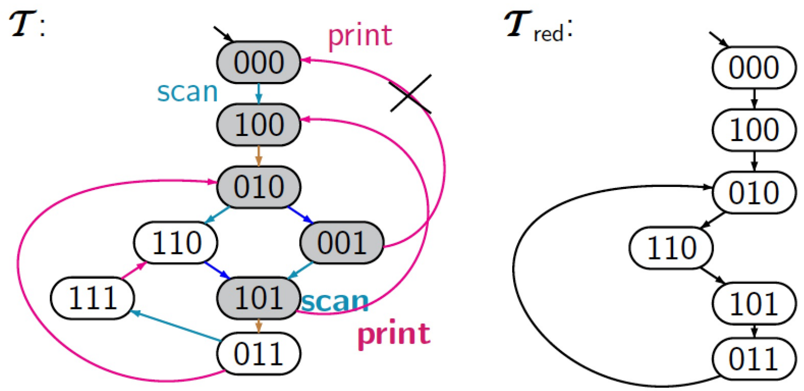
Example: Booking System



scan
code
price
print
scan
code
. . .

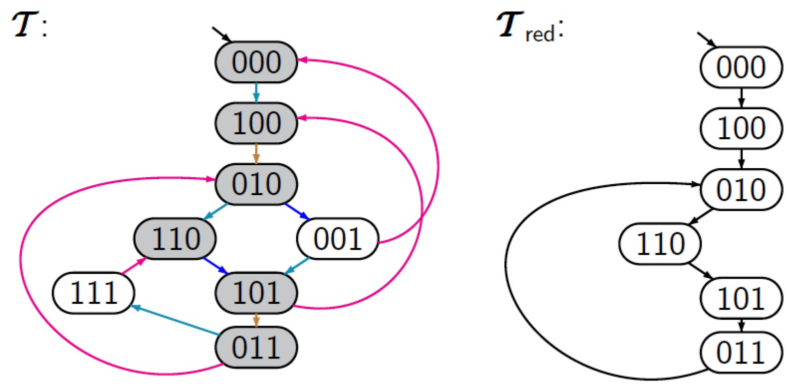
scan
code
scan
price
code
print
. . .

Example: Booking System



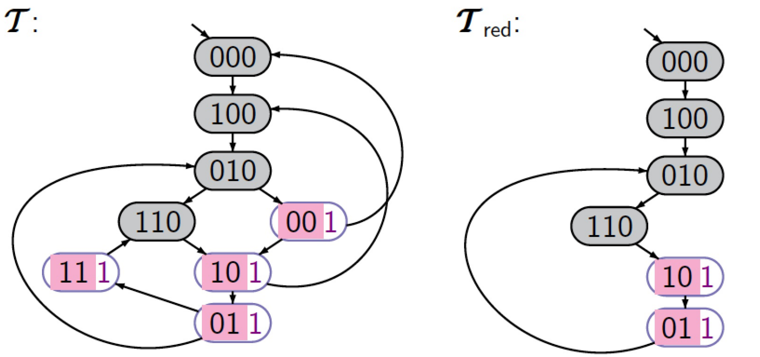
scan, code, price, print, scan, code, ... ⇝ scan, code, price, **scan**, **print**, code, ... ⇝ scan, code, scan, price, code, print, ...

Example: Booking System



scan, code, price, print, scan, code, ... ⇝ scan, code, scan, price, code, print, ... = scan, code, scan, price, code, print, ...

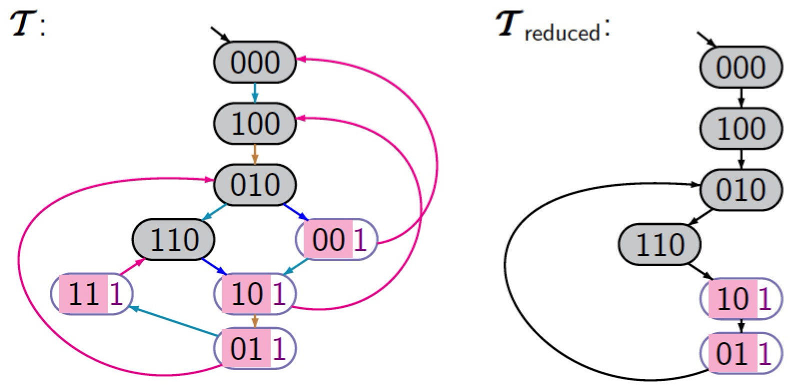
Example: Booking System



scan	∅		scan	∅		scan	∅
code	∅		code	∅		code	∅
price	∅		price	∅		scan	∅
print	{1}	⇝	scan	{1}	⇝	price	∅
scan	∅		print	{1}		print	{1}
code	∅		code	∅		code	∅

Proposition = "printer is in control state 1".

Example: Booking System



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

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

Overview

1. Motivation
2. Action Independence
3. Ample Sets
4. Correctness and Complexity
5. Summary

Action Determinism

Definition: action deterministic

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

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

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

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

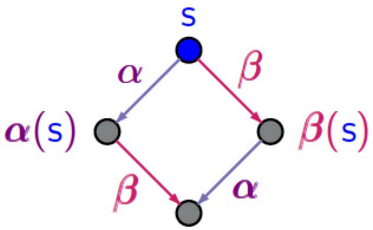
Action Independence

Definition: action independence

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

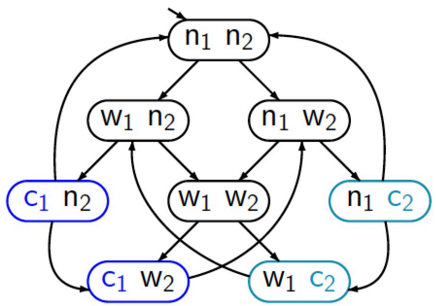
Actions $\alpha \in Act$ and $\beta \in Act$ are independent in TS if for all states s with $\alpha, \beta \in Act(s)$ the following holds:

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



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

Example: Semaphore-Based Mutual Exclusion

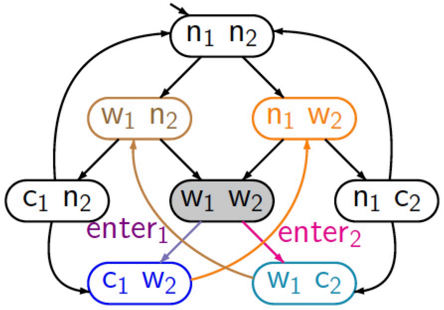


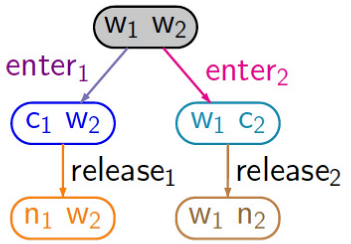
independent actions:

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

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

Example: Semaphore-Based Mutual Exclusion



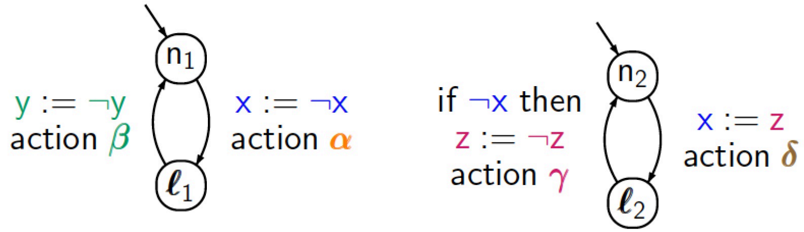


dependent actions:

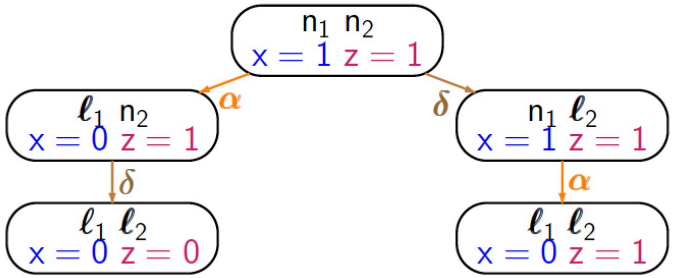
$enter_1$, $enter_2$

access both to the semaphore

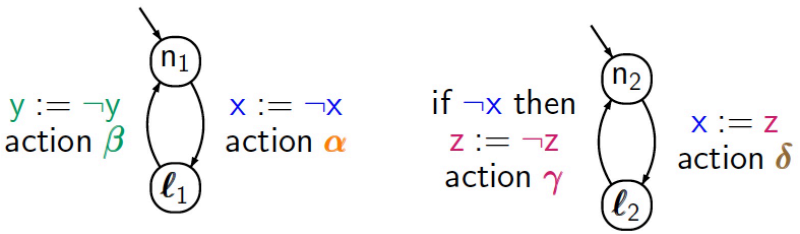
Example: Shared Variables



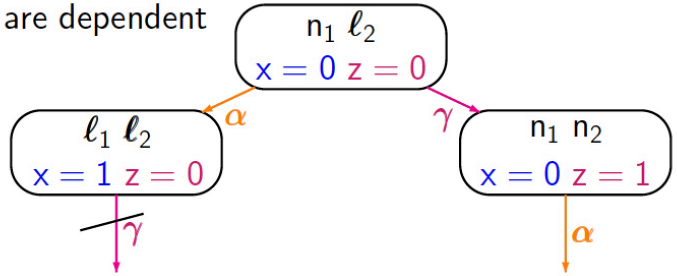
α, δ are dependent for $\mathcal{T}_{P_1 \,|||\, P_2}$



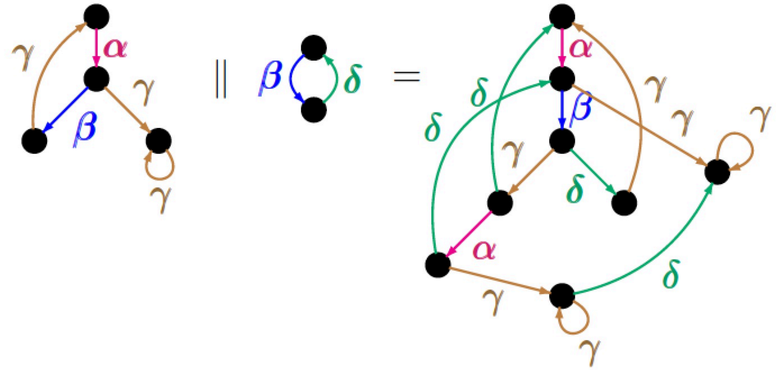
Example: Shared Variables



α, γ are dependent



Example: Synchronised Threads



α, δ independent $\surd$

γ, δ independent $\surd$

β, γ dependent

Permuting Independent Actions

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

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

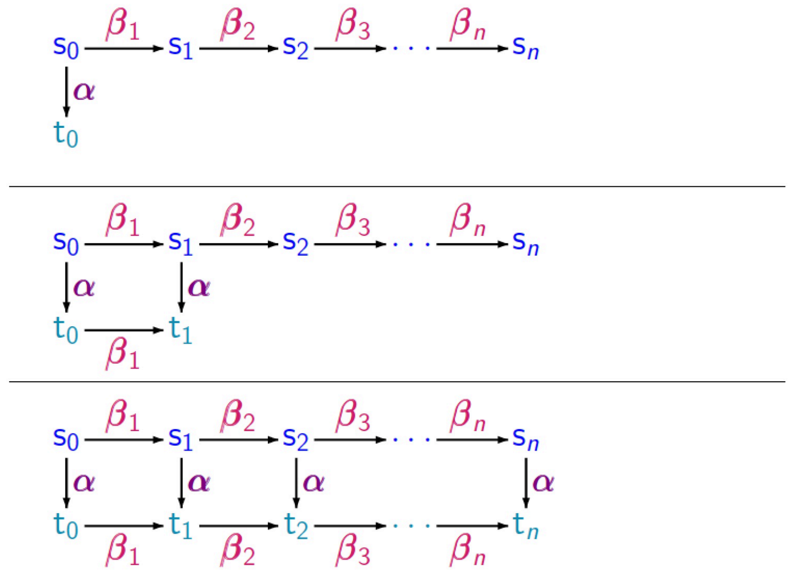
be a finite run in TS from s with action sequence $\beta_1 \ldots \beta_n$.

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

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

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

Pictorial Proof Sketch



Stutter Actions

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

$$\begin{array}{llllllllllll} \rho & = & s_0 & \xrightarrow{\beta_1} & s_1 & \xrightarrow{\beta_2} & \ldots & \xrightarrow{\beta_n} & s_n & \xrightarrow{\alpha} & t & \text{and} \\ \rho' & = & s_0 & \xrightarrow{\alpha} & t_0 & \xrightarrow{\beta_1} & \ldots & \xrightarrow{\beta_{n-1}} & t_{n-1} & \xrightarrow{\beta_n} & t & \end{array}$$

  will be distinct

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

Definition: stutter action

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

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

Permuting Independent Stutter Actions

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

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

Then:

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

Adding Independent Stutter Actions

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

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

Then:

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

Overview

1. Motivation
2. Action Independence
3. Ample Sets
4. Correctness and Complexity
5. Summary

The Ample-Set Approach

- Partial-order reduction for LTL formulas using ample sets
  - on state-space generation select $ample(s) \subseteq Act(s)$
  - such that $|ample(s)| \ll |Act(s)|$
- Reduced system $TS_{red} = (S', Act, \Longrightarrow, I, AP, L')$ where:
  - S' = the set of states reachable from some $s_0 \in I$ under $\Longrightarrow$
  - $\Longrightarrow$ is the smallest relation defined by:

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

  - $L'(s) = L(s)$ for any $s \in S'$
- Constraints: correctness ($\equiv_{sttrace}$), effectiveness and efficiency

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

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

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

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

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

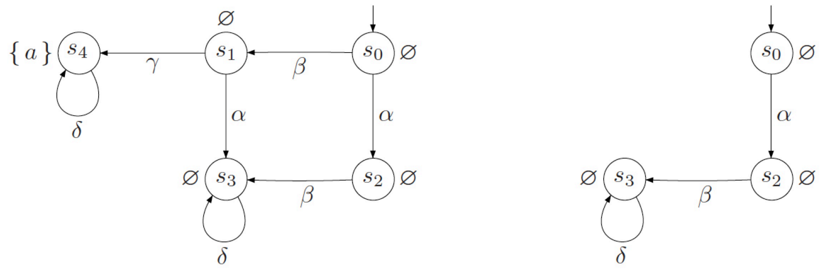
(A1) through (A3) apply to states in S'; (A4) to cycles in TS_{red}.

Example

Naive Dependency Condition (A2')

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

This is incorrect. (A2') allows the following reduction:



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

Dependency Condition (A2)

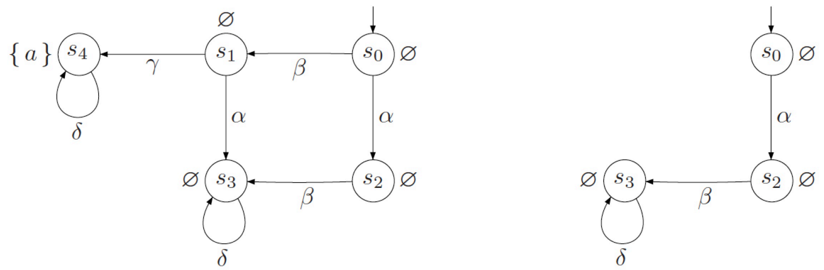
Dependency Condition (A2)

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

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

- In every (!) finite run of *TS*, an action dependent on $ample(s)$ cannot occur before some action from $ample(s)$ occurs first
- (A2) ensures that for any state s with $ample(s) \subset Act(s)$, 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 $\{ \alpha \} = ample(s_0)$

Properties

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

if $ample(s)$ satisfies (A2), then α is independent of $Act(s) \setminus ample(s)$.

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

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

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

Ample Set Conditions So Far

(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 $0 < i \leq n$.

(A3) **Stutter condition**

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

First Consequence of (A1)–(A3)

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

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

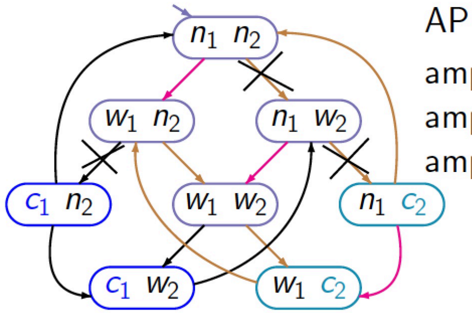
where $\beta_i \notin ample(s)$, for $0 < i \leq n$, and $\alpha \in ample(s)$.

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

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

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

Example: Ample Sets for Semaphore



$AP = \{c_1, c_2\}$

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

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

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

...

$n_1 n_2 \xrightarrow{request_2} n_1 w_2 \xrightarrow{enter_2} n_1 c_2 \xrightarrow{release_2} n_1 n_2 \xrightarrow{request_1} w_1 n_2$

$n_1 n_2 \xrightarrow{request_2} n_1 w_2 \xrightarrow{enter_2} n_1 c_2 \xrightarrow{request_1} w_1 c_2 \xrightarrow{release_2} w_1 n_2$

$n_1 n_2 \xrightarrow{request_2} n_1 w_2 \xrightarrow{request_1} w_1 w_2 \xrightarrow{enter_2} w_1 c_2 \xrightarrow{release_2} w_1 n_2$

$n_1 n_2 \xrightarrow{request_1} w_1 n_2 \xrightarrow{request_2} w_1 w_2 \xrightarrow{enter_2} w_1 c_2 \xrightarrow{release_2} w_1 n_2$

Second Consequence of (A1)–(A3)

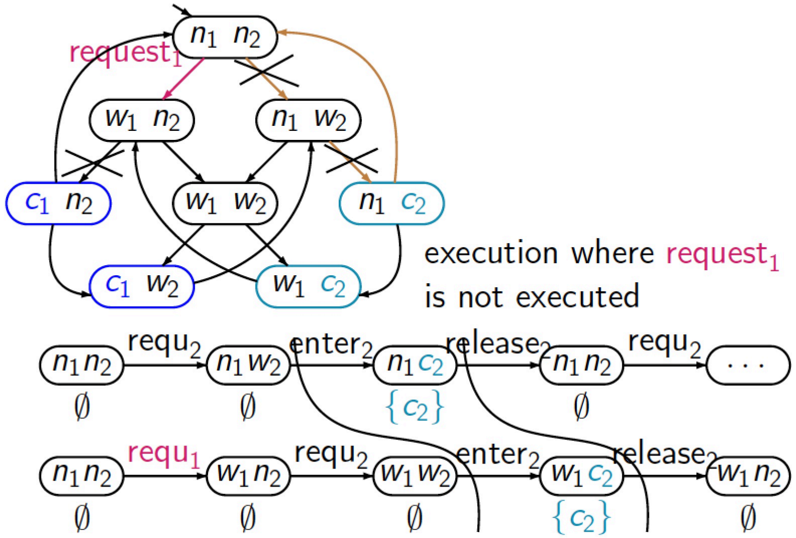
Let $\rho = s \xrightarrow{\beta_1} s_1 \xrightarrow{\beta_2} s_2 \xrightarrow{\beta_3} \ldots$ be an infinite run in $Reach(TS)$ where $\beta_i \notin ample(s)$, for $i > 0$.

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

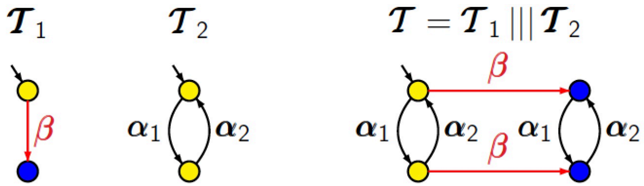
$$\rho' = s \xrightarrow{\alpha} t_0 \xrightarrow{\beta_1} t_1 \xrightarrow{\beta_2} t_2 \xrightarrow{\beta_3} \ldots$$

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

Example: Ample Sets for Semaphore



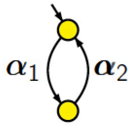
The Necessity of Cycle Condition (A4)



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

β, α_i independent
α_1, α_2 stutter actions

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



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

Cycle Condition

(A4) Cycle condition

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

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

Ample Set Conditions

(A1) **Nonemptiness condition**

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

(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 $0 < i \leq n$.

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

Overview

1 Motivation

2 Action Independence

3 Ample Sets

4 Correctness and Complexity

5 Summary

Correctness

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

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

Complexity Considerations

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

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

Proof.

Sketch on the black board. □

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

Some Experimental Results

[Clarke, Grumberg, Minea, Peled, 1999]

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

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

Overview

1 Motivation
2 Action Independence
3 Ample Sets
4 Correctness and Complexity
5 Summary

Summary

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

Next Lecture

Friday January 10, 14:30