Partial Order Reduction

Lecture #7 of Advanced Model Checking

Joost-Pieter Katoen

Lehrstuhl 2: Software Modeling & Verification

E-mail: `katoen@cs.rwth-aachen.de`

May 8, 2009

State space explosion

- Interleaving semantics

  - independent concurrent actions are interleaved
  - an execution 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 (or true concurrency) semantics

  - an execution is defined by a partially ordered sequence of states
  - models: posets, pomsets, event structures, Petri net unfoldings

- Partial-order reduction

  - group executions for which the order of "independent" actions is irrelevant
  - consider only one representative execution for equivalent executions

An example concurrent program

Dependencies

- ## Assume

  - x and y are local variables
  - g is a shared variable

- ## Dependent

  - $g := g * 2 + 10$ and $g := g + 2$ as they both operate on a shared variable
  - $x := 1$ and $g := g + 2$ as they are both executed by the same process
  - $y := 1$ and $g := g * 2 + 10$ as they are both executed by the same process

- ## Independent

  - $x := 1$ and $y := 1$
  - $x := 1$ and $g := g * 2 + 10$
  - $y := 1$ and $g := g + 2$

Dependencies

Idea of partial-order reduction

- Partition executions into equivalence classes

- Group executions for which the order of "independent" actions is irrelevant

- Consider only one representative execution for each equivalence class

in fact: model checking using representative executions
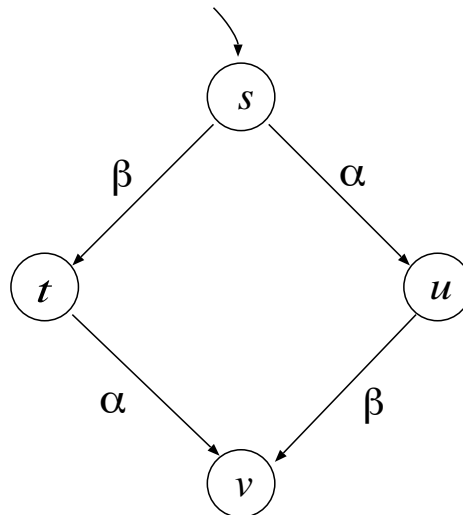
Pruning the state space

Preserving properties

Outline of partial-order reduction

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

  - a *reduced version* of transition system TS such that $\widehat{TS} \triangleq TS$
  $\Rightarrow$ this preserves all stutter sensitive LT properties, such as $\text{LTL}_{\backslash\bigcirc}$
  - at state s select a (small) subset of enabled actions in s
  - different approaches on how to select such set: consider Peled's *ample sets*

- *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 TS during $\text{LTL}_{\backslash\bigcirc}$ model checking
  - if accept cycle is found, there is no need to generate entire $\widehat{TS}$

Some 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 $u = t$
  - ... this should not be confused with *AP*-determinism
  - action-determinism is not a severe restriction: actions can always be renamed

- *Act*(s) is the set of *enabled* actions in state s

  - $Act(s) = \{\, \alpha \in Act \mid \exists s' \in S.\, s \xrightarrow{\alpha} s' \,\}$

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

Independence of actions

- the execution of α cannot disable β, and vice versa, and

- if $\alpha, \beta \in \mathit{Act}(s)$ then $\alpha\,\beta$ and $\beta\,\alpha$ executed in s yield the same state
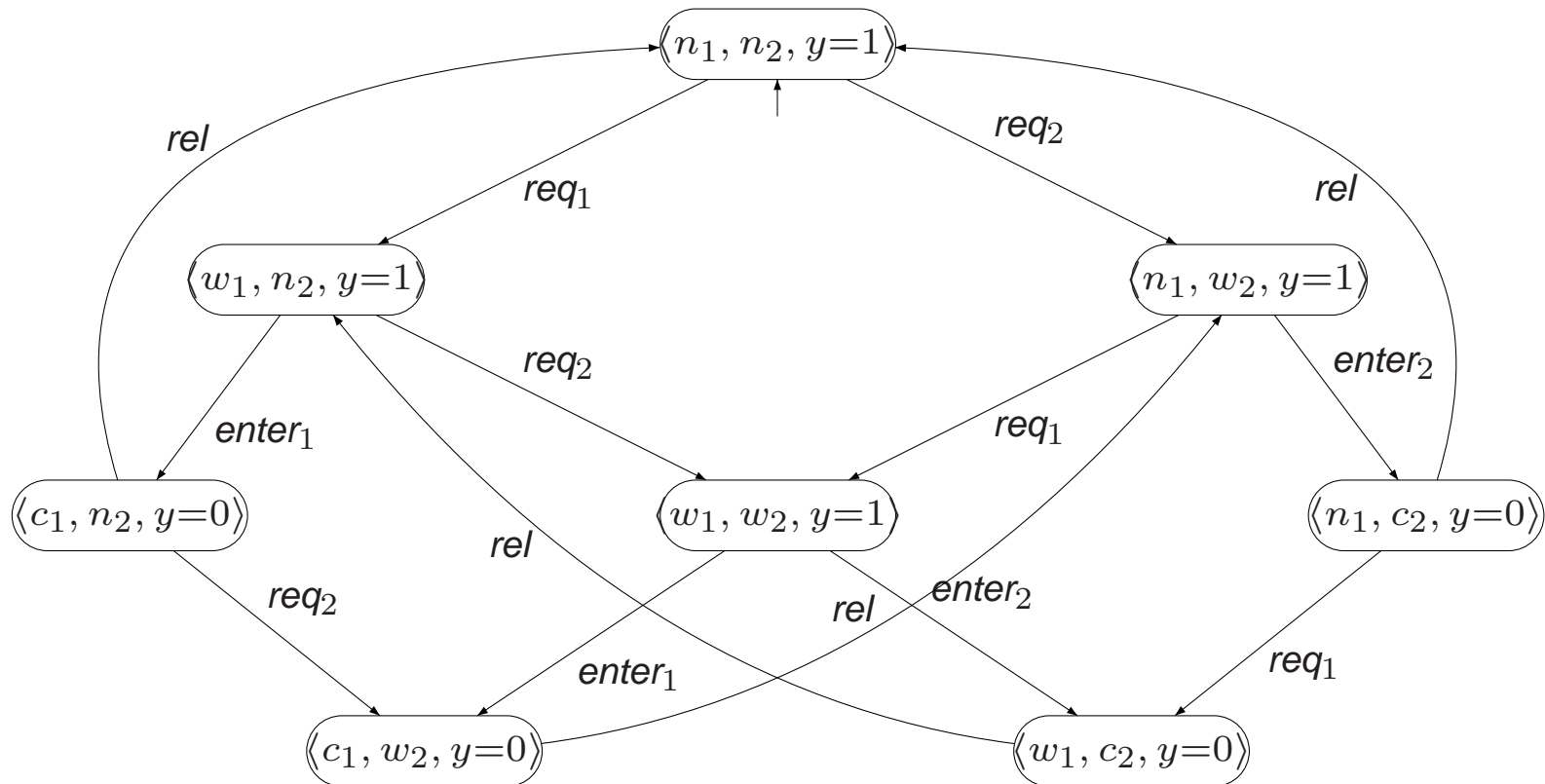
Independence of actions

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

- α and β are *independent* if for any $s \in S$ with $\alpha, \beta \in Act(s)$:

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

- α and β are *dependent* if α and β are not independent

- For $A \subseteq Act$ and $\beta \in Act \setminus A$:

  - β is independent of A if for any $\alpha \in A$, β is independent of α
  - β depends on A in TS if $\beta \in Act \setminus A$ and α are dependent for some $\alpha \in A$

Example

Permuting independent actions

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

$$s = s_0 \xrightarrow{\beta_1} s_1 \xrightarrow{\beta_2} \ldots \xrightarrow{\beta_{n-1}} s_{n-1} \xrightarrow{\beta_n} s_n$$
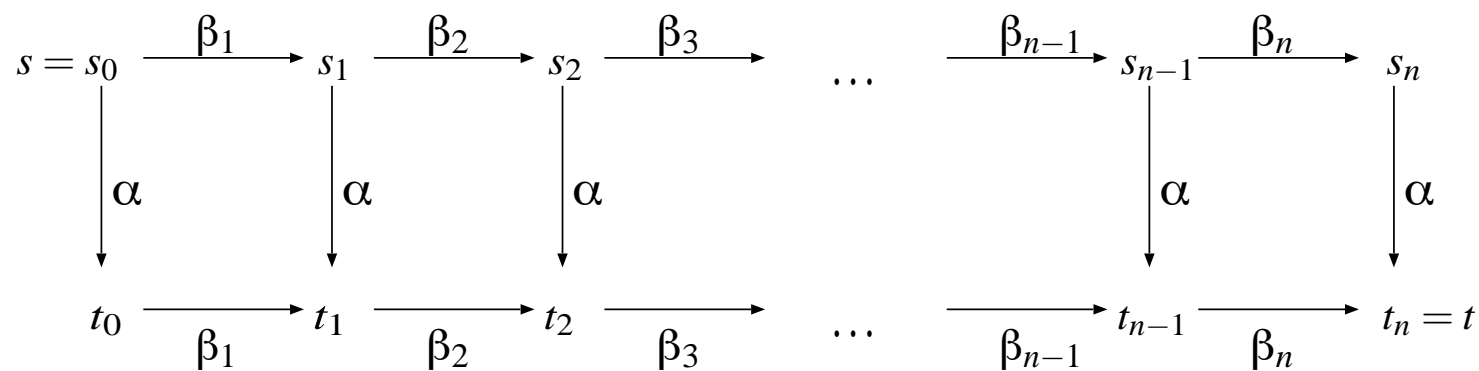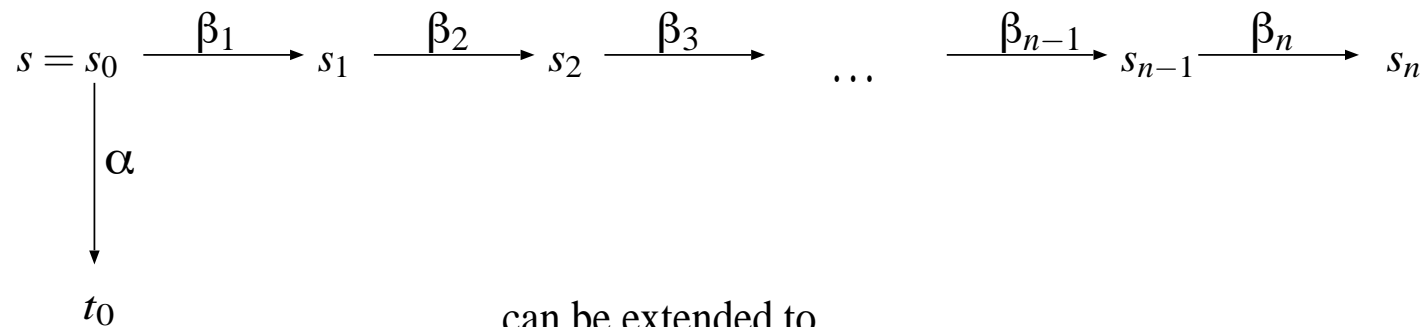
be an execution fragment in *TS* from s with action sequence $\beta_1 \ldots \beta_n$

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 an execution fragment in *TS* from s with action sequence $\alpha \, \beta_1 \ldots \beta_n$

Permuting independent actions

$$s = s_0 \xrightarrow{\beta_1} s_1 \xrightarrow{\beta_2} s_2 \xrightarrow{\beta_3} \quad \ldots \quad \xrightarrow{\beta_{n-1}} s_{n-1} \xrightarrow{\beta_n} s_n$$

$$\downarrow \alpha$$

$$t_0$$

can be extended to

$$s = s_0 \xrightarrow{\beta_1} s_1 \xrightarrow{\beta_2} s_2 \xrightarrow{\beta_3} \quad \ldots \quad \xrightarrow{\beta_{n-1}} s_{n-1} \xrightarrow{\beta_n} s_n$$

$$\downarrow \alpha \quad \downarrow \alpha \quad \downarrow \alpha \quad \downarrow \alpha \quad \downarrow \alpha$$

$$t_0 \xrightarrow{\beta_1} t_1 \xrightarrow{\beta_2} t_2 \xrightarrow{\beta_3} \quad \ldots \quad \xrightarrow{\beta_{n-1}} t_{n-1} \xrightarrow{\beta_n} t_n = t$$

Proof

Adding an independent action

Let *TS* be an action-deterministic transition system, 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 execution fragment in *TS* with action sequence $\beta_1 \, \beta_2 \, \beta_3 \ldots$

Then, for $\alpha \in Act(s)$ independent of $\{ \beta_1, \beta_2, \ldots \}$: $\alpha \in Act(s_i)$ for all 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 execution fragment in *TS* with action sequence $\alpha \, \beta_1 \, \beta_2 \, \beta_3 \, \ldots$
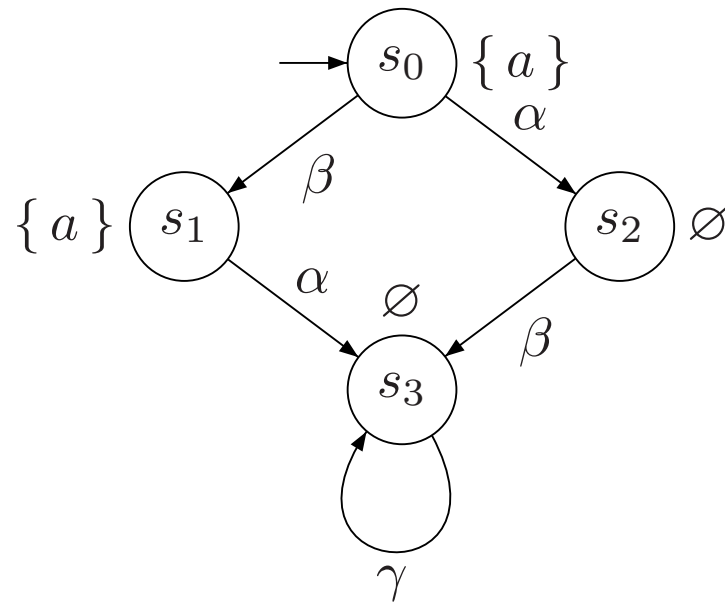
Stutter actions

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

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

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

 will be <u>distinct</u>!

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

Example

Permuting independent stutter actions

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

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

- ϱ' is a finite execution 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 $\varrho \triangleq \varrho'$

Proof

Adding an independent stutter action

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

- ρ is an **in**finite execution in s with action sequence $\beta_1 \beta_2 \dots$

- ρ' is an **in**finite execution in s with action sequence $\alpha \beta_1 \beta_2 \dots$

Then:

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

The ample-set approach

- Partial-order reduction for LT properties using *ample sets*

  - on state-space generation select *ample*$(s) \subseteq$ *Act*(s)
  - such that $|$*ample*$(s)| \ll |$*Act*$(s)|$

- *Reduced* system $\widehat{TS} = (\widehat{S}, Act, \Longrightarrow, I, AP, L')$ where:

  - $\widehat{S}$ contains the states that are reachable (under $\Longrightarrow$) from some $s_0 \in I$
  - $$\frac{s \xrightarrow{\alpha} s' \;\wedge\; \alpha \in \textit{ample}(s)}{s \xRightarrow{\alpha} s'}$$
  - $L'(s) = L(s)$ 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 execution 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