Partial Order Reduction

Lecture #9 of Advanced Model Checking

Joost-Pieter Katoen

Lehrstuhl 2: Software Modeling & Verification

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

November 23, 2006

Symbolic versus explicit model checking

- Symbolic model checking using BDDs
  - (sets of) states and transitions are represented as Boolean functions
  - model-checking operations work with BDD representations
  - is appropriate for synchronous systems (e.g., hardware)
  - in practice mostly applied to CTL (but LTL possible)
- Explicit-state model checking
  - the state space is explicitly represented by states
  - optimization techniques are necessary to keep
    1. the number of states as low as possible, and/or
    2. the memory usage per state as low as possible
  - is appropriate for asynchronous systems (e.g., concurrent software)
  - in practice mostly applied to LTL (but CTL possible)

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

Stutter equivalence

- $s \rightarrow s'$ in transition system *TS* is a *stutter step* if $L(s) = L(s')$
  - stutter steps do not affect the state labels of successor states
- Paths π_1 and π_2 are *stutter equivalent*, denoted $\pi_1 \cong \pi_2$, if:

$$\textit{trace}(\pi_1) \text{ and } \textit{trace}(\pi_2) \text{ belong to } A_0^+ A_1^+ A_2^+ \ldots \text{ for } A_i \subseteq AP$$

if *trace*(π_1) and *trace*(π_2) only differ in the number of stutter steps per “segment”

Stutter trace equivalence

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

$$\mathit{TS}_1 \cong \mathit{TS}_2 \quad \text{if and only if} \quad \mathit{TS}_1 \sqsubseteq \mathit{TS}_2 \text{ and } \mathit{TS}_2 \sqsubseteq \mathit{TS}_1$$

where $\sqsubseteq$ is defined by:

$$\mathit{TS}_1 \sqsubseteq \mathit{TS}_2 \quad \text{iff} \quad \forall \pi_1 \in \mathit{Paths}(\mathit{TS}_1) \; (\exists \pi_2 \in \mathit{Paths}(\mathit{TS}_2). \; \pi_1 \cong \pi_2 \,)$$

clearly: $\mathit{Traces}(\mathit{TS}_1) = \mathit{Traces}(\mathit{TS}_2)$ implies $\mathit{TS}_1 \cong \mathit{TS}_2$, but not always the reverse

Stutter trace and LTL$_{\setminus\bigcirc}$ equivalence

For transition systems TS_1, TS_2 (over AP) without terminal states:

(a) $TS_1 \cong TS_2$ implies $TS_1 \equiv_{LTL\setminus\bigcirc} TS_2$

(b) if $TS_1 \sqsubseteq TS_2$ then for any LTL$_{\setminus\bigcirc}$ formula φ: $TS_2 \models \varphi$ implies $TS_1 \models \varphi$

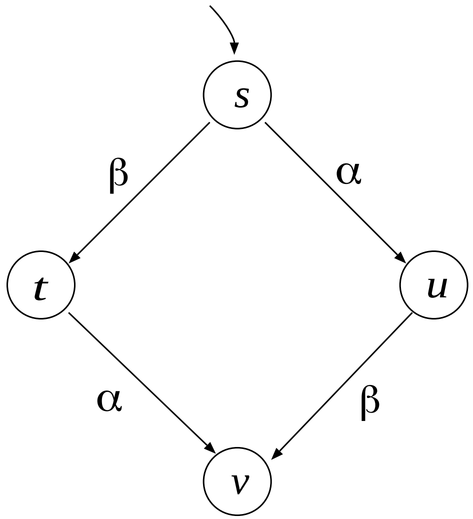
Outline of partial-order reduction

- During state space generation obtain $\widehat{TS}$
  - a *reduced version* of transition system TS such that $\widehat{TS} \cong TS$
  - $\Rightarrow$ this preserves all stutter sensitive LT properties, such as $\text{LTL}_{\setminus \bigcirc}$
  - 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 $\text{LTL}_{\setminus \bigcirc}$ 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*$(s) = \{ \alpha \in \mathit{Act} \mid \exists s' \in S.\, s \xrightarrow{\alpha} s' \}$
- $\alpha(s)$ denotes the unique *α-successor* of s, i.e., $s \xrightarrow{\alpha} \alpha(s)$

Independence of actions



- the execution of α cannot disable β, and vice versa, and
- if $\alpha, \beta \in Act(s)$ then $\alpha\,\beta$ and $\beta\,\alpha$ executed in s yield the same state

Independence of actions

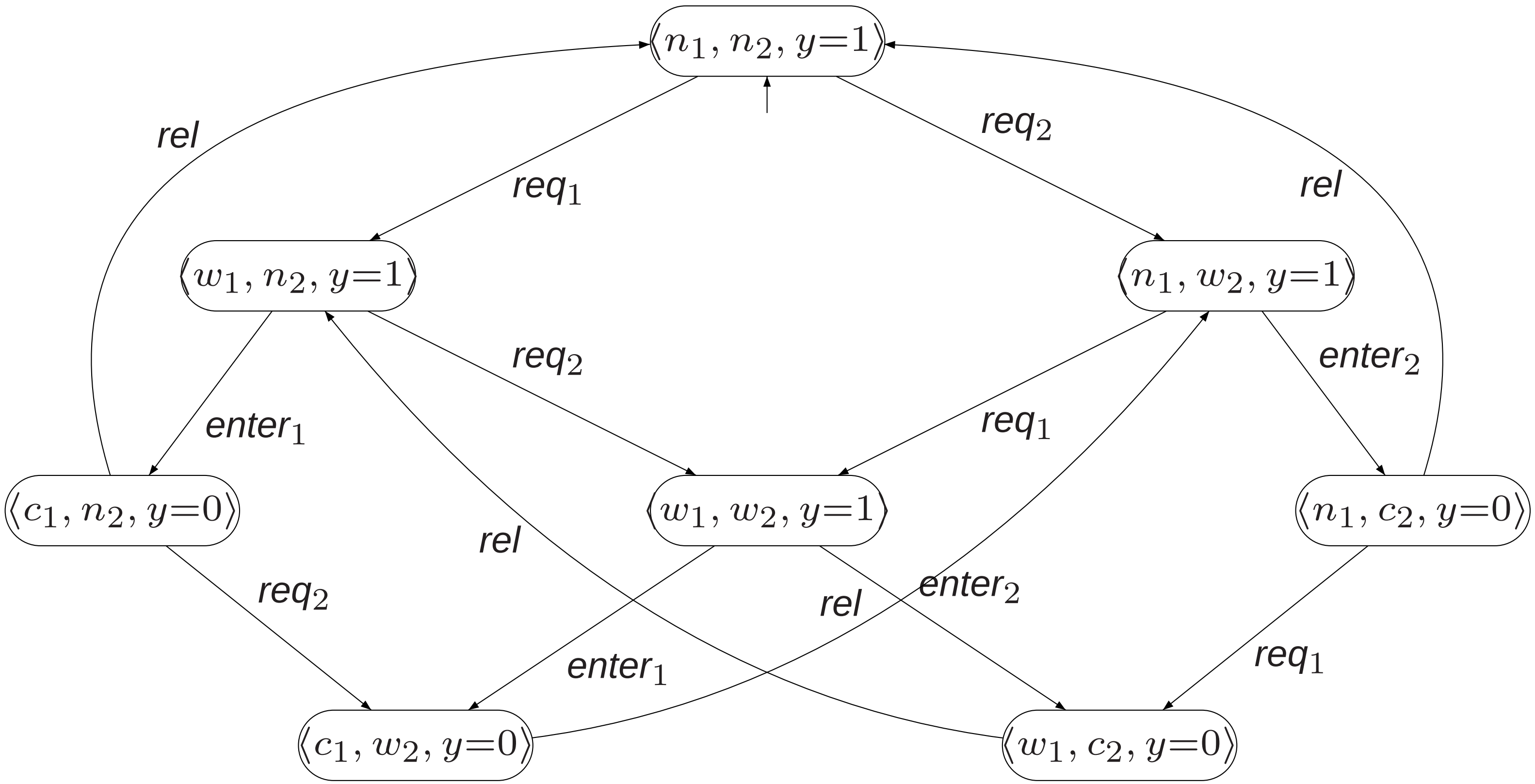
Let $TS = (S, Act, \rightarrow, I, AP, L)$ be action-deterministic and $\alpha \neq \beta \in Act$

- 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



$\langle n_1, n_2, y{=}1\rangle$
$\langle w_1, n_2, y{=}1\rangle$
$\langle n_1, w_2, y{=}1\rangle$
$\langle c_1, n_2, y{=}0\rangle$
$\langle w_1, w_2, y{=}1\rangle$
$\langle n_1, c_2, y{=}0\rangle$
$\langle c_1, w_2, y{=}0\rangle$
$\langle w_1, c_2, y{=}0\rangle$
rel
req_1
req_2
rel
req_2
$enter_2$
$enter_1$
req_1
rel
req_2
$enter_2$
rel
$enter_1$
req_1

Permuting independent actions

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

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

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

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

$$
\begin{array}{ccccccccccc}
s = s_0 & \xrightarrow{\beta_1} & s_1 & \xrightarrow{\beta_2} & s_2 & \xrightarrow{\beta_3} & \ldots & \xrightarrow{\beta_{n-1}} & s_{n-1} & \xrightarrow{\beta_n} & s_n \\
\downarrow \alpha & & & & & & & & & & \\
t_0 & & & & & & & & & &
\end{array}
$$

can be extended to

$$
\begin{array}{ccccccccccc}
s = s_0 & \xrightarrow{\beta_1} & s_1 & \xrightarrow{\beta_2} & s_2 & \xrightarrow{\beta_3} & \ldots & \xrightarrow{\beta_{n-1}} & s_{n-1} & \xrightarrow{\beta_n} & s_n \\
\downarrow \alpha & & \downarrow \alpha & & \downarrow \alpha & & & & \downarrow \alpha & & \downarrow \alpha \\
t_0 & \xrightarrow[\beta_1]{} & t_1 & \xrightarrow[\beta_2]{} & t_2 & \xrightarrow[\beta_3]{} & \ldots & \xrightarrow[\beta_{n-1}]{} & t_{n-1} & \xrightarrow[\beta_n]{} & t_n = t
\end{array}
$$

Proof

Adding an independent action

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

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

be an infinite execution fragment in *TS* from s with action seq. $\beta_1 \, \beta_2 \, \beta_3 \ldots$.

Then, for $\alpha \in \mathit{Act}(s)$ independent of $\{ \beta_1, \beta_2, \ldots \}$: $\alpha \in \mathit{Act}(s_i)$ 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 seq. $\alpha \, \beta_1 \, \beta_2 \ldots$

Stutter actions

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

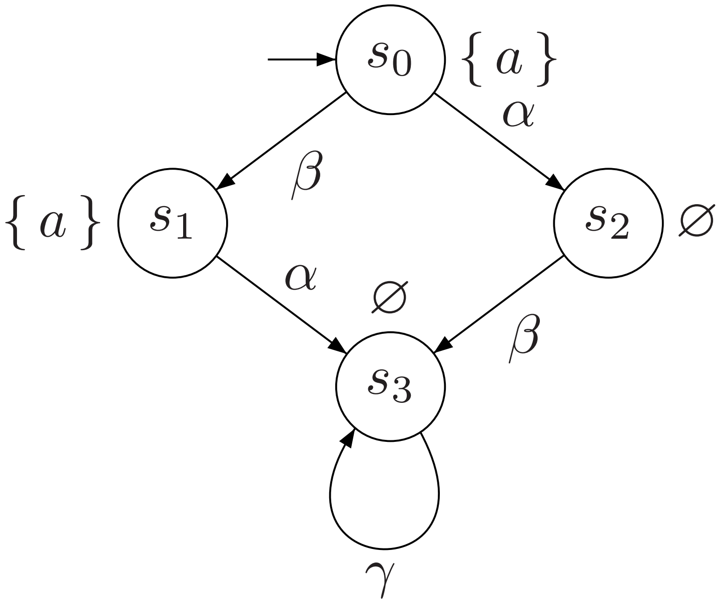
$$\rho = s_0 \xrightarrow{\beta_1} s_1 \xrightarrow{\beta_2} \ldots \xrightarrow{\beta_n} s_n \xrightarrow{\alpha} t \text{ and}$$

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

  will be distinct!

- If α does not affect the state-labelling (= "invisible"), then $\rho \cong \rho'$
- $\alpha \in$ *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



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 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 $\varrho \cong \varrho'$

Proof

Adding an independent stutter action

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

- ρ is an infinite execution in s with action sequence $\beta_1 \, \beta_2 \ldots$
- ρ' is an infinite execution 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 $\rho \cong \rho'$