On-The-Fly Partial Order Reduction

Lecture #11 of Advanced Model Checking

Joost-Pieter Katoen

Lehrstuhl 2: Software Modeling & Verification

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

December 4, 2006

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

Ample-set conditions for LTL

(A1) **Nonemptiness condition**

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

(A2) **Dependency condition**

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

(A3) **Stutter condition**

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

(A4) **Cycle condition**

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

Correctness theorem

For action-deterministic, finite *TS* without terminal states:

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

Strong cycle condition

(A4') Strong cycle condition

On any cycle $s_0\, s_1\, \ldots\, s_n$ in $\widehat{TS}$,

there exists $j \in \{\, 1, \ldots, n \,\}$ such that $\mathit{ample}(s_j) = \mathit{Act}(s_j)$.

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 taking

$\Rightarrow$ Nested depth-first search can be extended similarly

  - this yields a model-checking procedure with on-the-fly POR for LTL$_{\setminus\bigcirc}$

Example

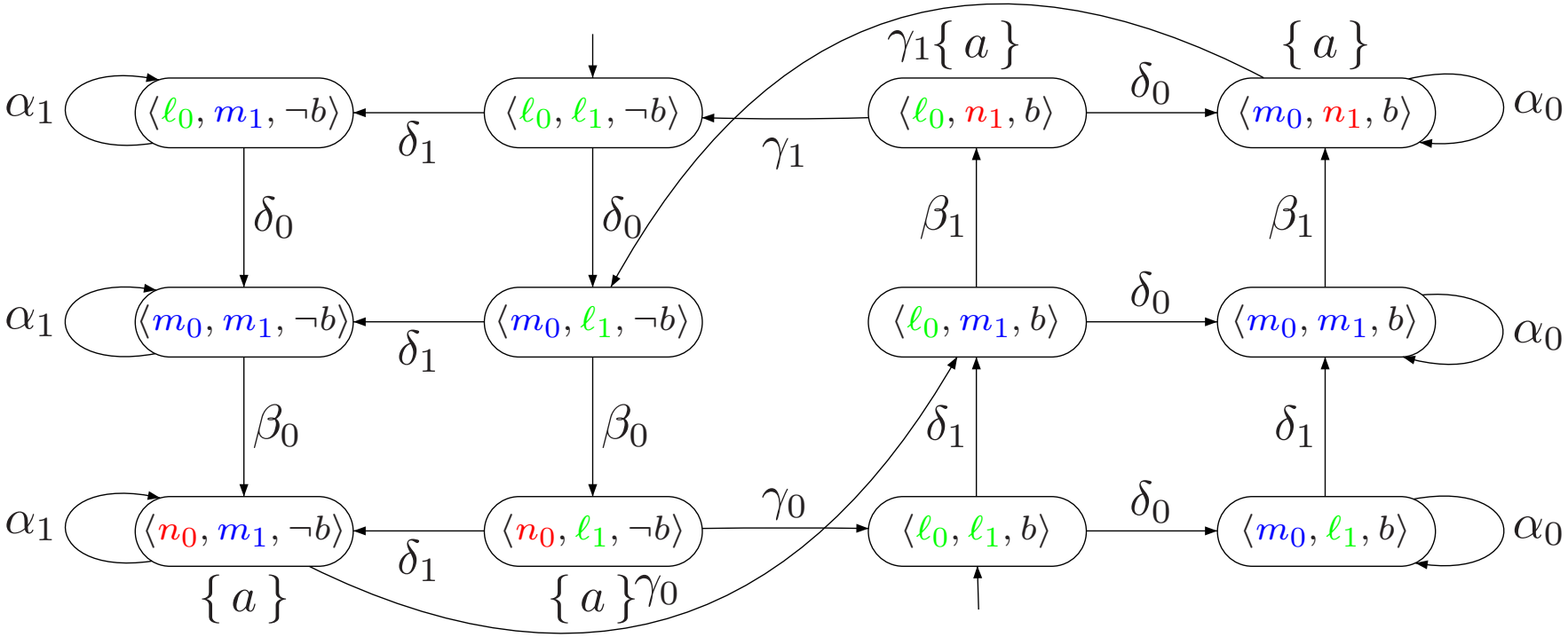
Process 0:

```
          while true {
ℓ0 :          skip;
m0 :          wait until (¬b) {
n0 :              . . . critical section . . .}
              b := true;
          }
```

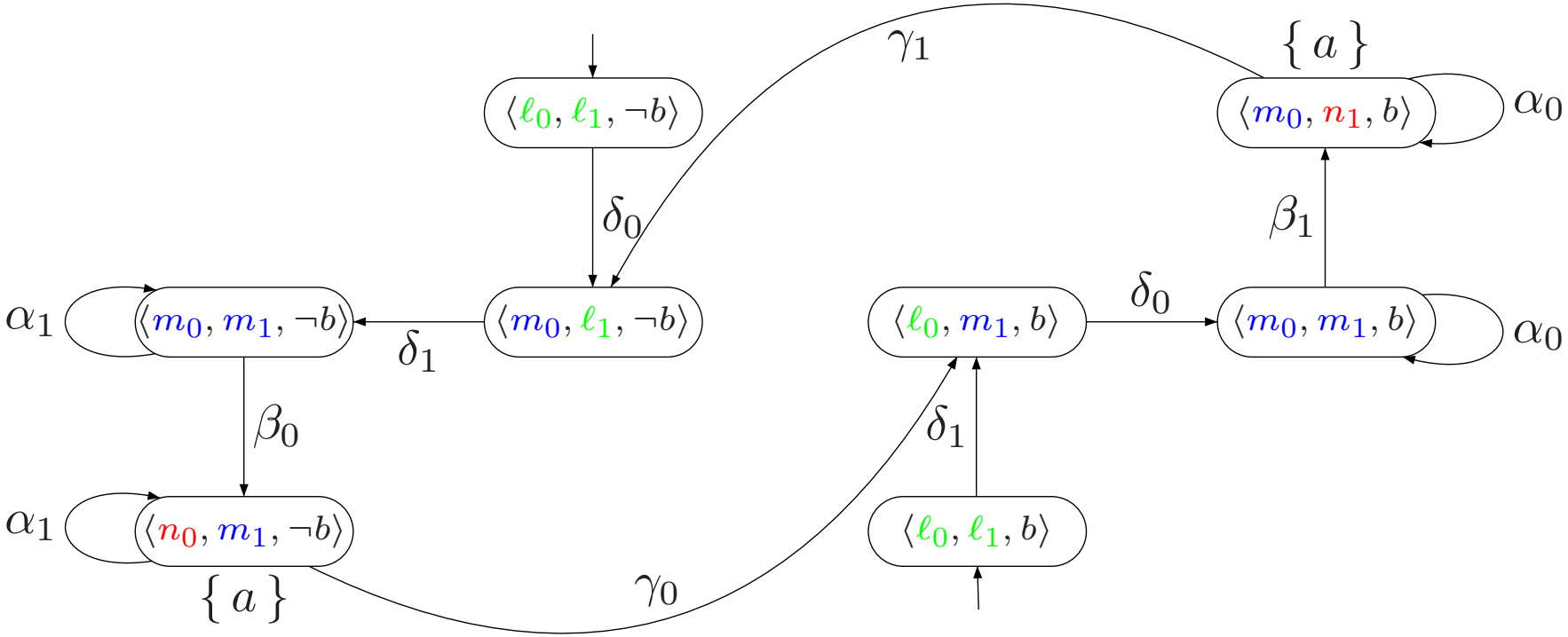
Process 1:

```
          while true {
ℓ1 :          skip;
m1 :          wait until (b) {
n1 :              . . . critical section . . .}
              b := false;
          }
```

Transition system



Reduced transition system



Computing ample sets

- Aim: determine ample sets by a static analysis of channel system *CS*

$$TS = TS(CS) \quad \text{where} \quad CS = [PG_1 \mid \ldots \mid PG_n]$$

  - state s in *TS* has the form $\langle \ell_1, \ldots, \ell_n, \eta, \xi \rangle$ where
  - ℓ_i denotes the current location (control point) of PG_i
  - η is the variable valuation, and ξ the channel valuation

- Basic idea:

  - partition the set of processes $\mathcal{P}_1$ through $\mathcal{P}_n$ into two blocks
  - one block $\mathcal{P}_{i_1}, \ldots \mathcal{P}_{i_k}$ such that $\mathcal{P}_{i_j}$ does not communicate with $\mathcal{P}_i$ outside block
  - intuition: $ample(s) = Act_{i_1}(s) \cup \ldots \cup Act_{i_k}(s)$, for state s in $TS(CS)$
  - for simplicity: mostly $k{=}1$ is considered: $ample(s) = Act_i(s)$, for some i

Checking ample set conditions

Let $\mathit{Act}_i(s) \subset \mathit{Act}(s)$:

- Nonemptiness condition (A1):
  - check whether process $\mathcal{P}_i$ can perform an action in state s, i.e., $\mathit{Act}_i(s) \neq \varnothing$
- Stutter condition (A3):
  - α is a stutter action if the atomic propositions do neither refer to:
    - a variable that is modified by α, nor
    - the source or target location of edges of the form $\ell \overset{g:\alpha}{\rightsquigarrow} \ell'$, nor
    - the content of channel c in case α is a receive or send action on c
- Cycle condition (A4):
  - fully expand s if during its (nested) 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 transition system TS equals that of checking $TS' \models \exists\Diamond a$, for some $a \in AP$, where TS' is a finite, action-deterministic transition system of the same size as TS.

Proof

Conservative heuristic for 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
- Send (receive) actions on the same channel are dependent
  - but $c!v$ and $c?x$ for channel c with capacity one can never be enabled both
- Handshake actions depend on all actions in both processes

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

Local criteria for (A2)

To ensure condition (A2) check the conditions:

(A2.1) Any $\beta \in \mathit{Act}_j$ is independent of $\mathit{Act}_i(s)$ for $i \neq j$

- inspect program graphs PG_j and check whether $(\alpha, \beta) \notin D$ for $\alpha \in \mathit{Act}_i$ and $\beta \in \mathit{Act}_j$
- note: all actions local to PG_i are considered to be dependent

(A2.2) Any $\beta \in \mathit{Act}_i \setminus \mathit{Act}(s)$ may not become enabled through the activities of some process $\mathcal{P}_j$ with $i \neq j$

- consider $s = \langle \ell_1, \ldots, \ell_i, \ldots, \ell_n, \eta, \xi \rangle$ and $\beta \in \mathit{Act}_i \setminus \mathit{Act}(s)$
- e.g., in $\ell_i \overset{g:\beta}{\rightsquigarrow} \ell_i'$ in PG_i, g does not hold or β is blocked
- . . . e.g., a send action to a full channel, or a receive on an empty channel

if (A2.1) and (A2.2) hold, then $\mathit{ample}(s) = \mathit{Act}_i(s)$ *satisfies (A2)*

Input: state $s = \langle \ell_1, \ldots, \ell_n, \eta, \xi \rangle$ in $\widehat{TS}$; *Output*: *ample*(s) satisfying (A1)-(A3)

---

if $(\exists i.\, Act_i(s) = Act(s))$ **then return** $Act(s)$ **fi**;
for $i = 1$ **to** n **do** (* check whether $ample(s) = Act_i(s)$ is possible *)
if ($Act_i \neq \varnothing$ and $Act_i(s)$ only contains stutter actions) **then**
  if $(\exists j \neq i.\, Act_i(s) \times Act_j(s) \cap D = \varnothing)$ **then**
    $b :=$ true; (* (A2.1) holds *)
    if $\exists \ell_i \overset{g:\beta}{\rightsquigarrow} \ell_i'$ in PG_i where β is a handshaking action **then**
      $b :=$ false; (* (A2.2) violated *)
    else
      for all $\ell_i \overset{g:\beta}{\rightsquigarrow} \ell_i'$ in PG_i and $\ell_j' \overset{h:\gamma}{\rightsquigarrow} \ell_j''$ in PG_j with $j \neq i$ and $\ell_j \rightsquigarrow^* \ell_j'$ **do**
        if ($\eta \not\models g$ and γ modifies some variable that occurs in g) or
            (β and γ are complementary communication actions) **then**
          $b :=$ false; (* (A2.2) violated *)
        fi
      od
    fi
    if (b) **then return** $Act_i(s)$ **fi** (* (A1)-(A3) hold *)
  fi
fi
od
return $Act(s)$ (* $ample(s) := Act(s)$ *)

The branching-time ample approach

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

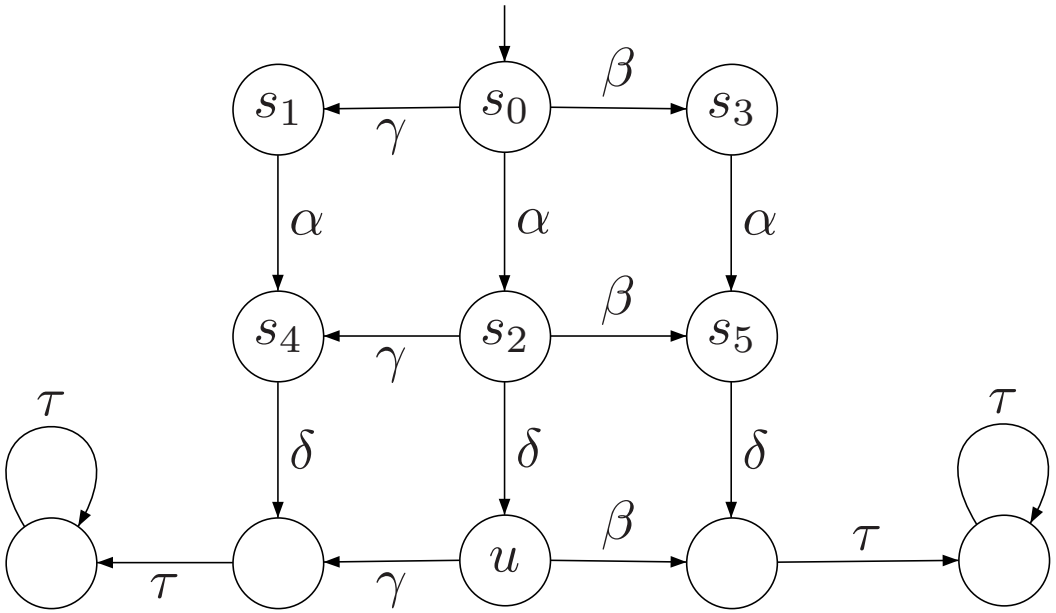
  $\Rightarrow$ this preserves all stutter sensitive LT properties, such as LTL$_{\setminus\bigcirc}$
  - static partial order reduction: generate $\widehat{TS}$ prior to verification
  - on-the-fly partial order reduction: generate $\widehat{TS}$ during the verification
  - generation of $\widehat{TS}$ 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 CTL$_{\setminus\bigcirc}$ and CTL$^{*}_{\setminus\bigcirc}$ formulas
  - static partial order reduction only

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

Example



transition system *TS*

Conditions (A1)-(A4) are insufficient

Branching condition

(A5)

If $\mathit{ample}(s) \neq \mathit{Act}(s)$ then $|\mathit{ample}(s)| = 1$

A sound reduction

Correctness theorem

For action-deterministic, finite *TS* without terminal states:

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

proof: show that (A1)-(A5) imply a normed bisimulation between *TS* and $\widehat{TS}$

normed bisimulation is strictly finer than $\approx^{div}$

Ample-set conditions for CTL

(A1) **Nonemptiness condition**

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

(A2) **Dependency condition**

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

(A3) **Stutter condition**

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

(A4) **Cycle condition**

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

(A5) **Branching condition**

If $\mathit{ample}(s) \neq \mathit{Act}(s)$ then $|\mathit{ample}(s)| \;=\; 1$