On-The-Fly Partial Order Reduction

Lecture #9 of Advanced Model Checking

Joost-Pieter Katoen

Lehrstuhl 2: Software Modeling & Verification

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

May 20, 2009

Outline of partial-order reduction

- During state space generation obtain $\widehat{TS}$
  - a *reduced version* of transition system *TS* such that $\widehat{TS} \triangleq TS$
  - ⇒ 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*)
  - ⇒ POR is preprocessing phase of model checking

- *Dynamic (or: on-the-fly)* partial-order reduction
  - construct $\widehat{TS}$ during $\text{LTL}_{\setminus \bigcirc}$ 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 $\widehat{TS} \triangleq TS$.

Strong cycle condition

(A4') Strong cycle condition

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

there exists $j \in \{1, \ldots, n\}$ such that *ample*$(s_j) =$ *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., $\mathit{ample}(s) = \mathit{Act}(P_i)$, enabled actions of a concurrent process
  - enlarge $\mathit{ample}(s)$ on demand using the strong cycle condition (A4')
  - mark actions to keep track of which actions have been taking

Example

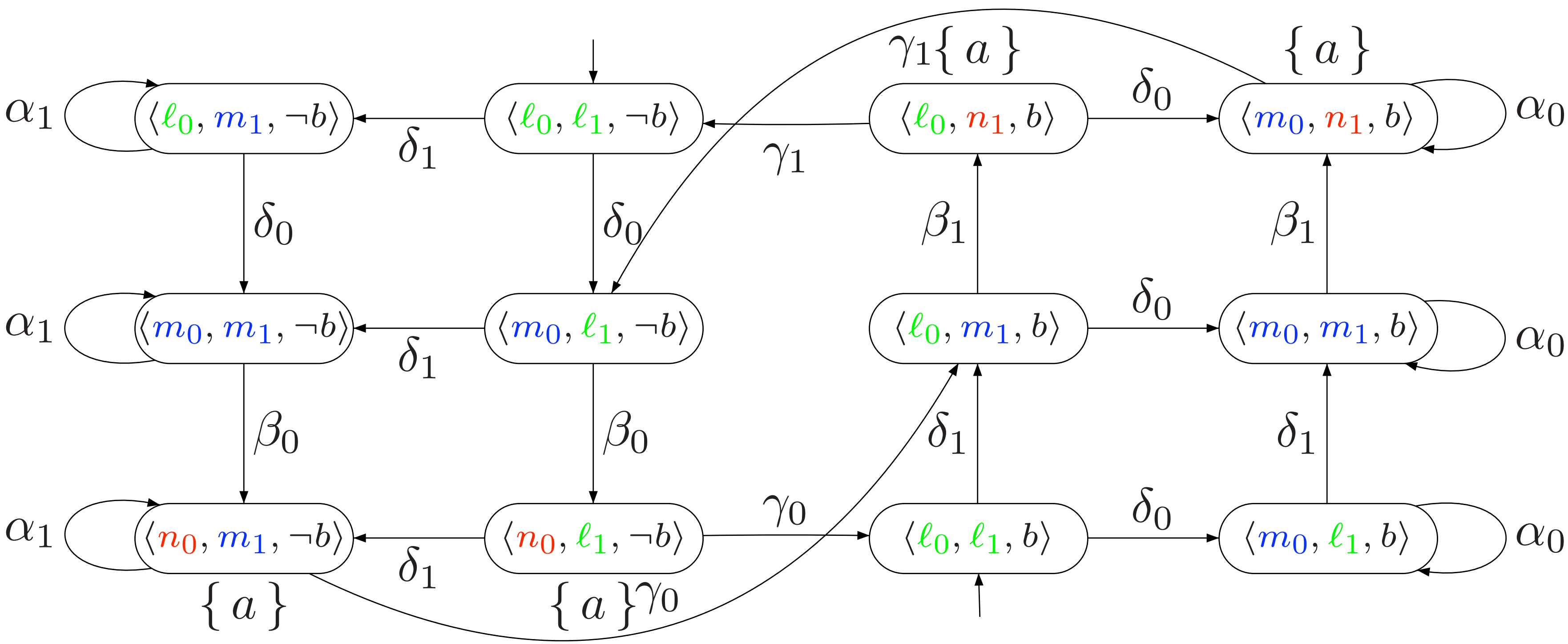
Process 0:

$$
\begin{array}{ll}
 & \textbf{while } \text{true } \{ \\
\ell_0 : & \quad \textit{skip}; \\
m_0 : & \quad \textbf{wait until } (\neg b) \ \{ \\
n_0 : & \quad\quad \ldots \text{critical section} \ldots \} \\
 & \quad b := \text{true}; \\
 & \}
\end{array}
$$

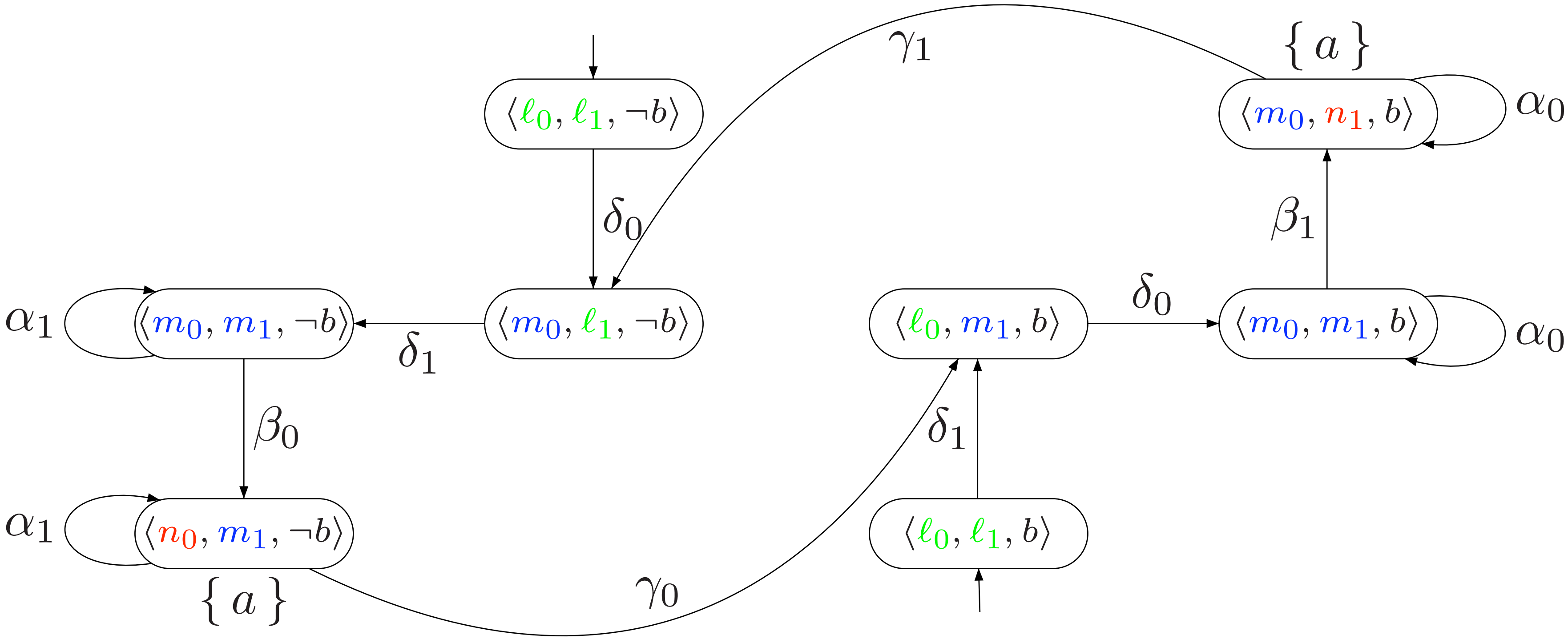
Process 1:

$$
\begin{array}{ll}
 & \textbf{while } \text{true } \{ \\
\ell_1 : & \quad \textit{skip}; \\
m_1 : & \quad \textbf{wait until } (b) \ \{ \\
n_1 : & \quad\quad \ldots \text{critical section} \ldots \} \\
 & \quad b := \text{false}; \\
 & \}
\end{array}
$$

Transition system



Reduced transition system



Invariant checking under POR (1)

Input: finite transition system *TS* and propositional formula Φ
Output: "yes" if *TS* $\models \Box\Phi$", otherwise "no" plus a counterexample

```
set of states R := ∅;                          (* the set of reachable states *)
stack of states U := ε;                        (* the empty stack *)
bool b := true;                                (* all states in R satisfy Φ *)
while (I \ R ≠ ∅ ∧ b) do
   let s ∈ I \ R;                              (* choose an arbitrary initial state not in R *)
   visit(s);                                   (* perform a DFS for each unvisited initial state *)
od
if b then
   return("yes")                               (* TS ⊨ "always Φ" *)
else
   return("no", reverse(U))                    (* counterexample arises from the stack content *)
fi
```

Invariant checking under POR (2)

```
procedure visit (state s)
   push(s, U); R := R ∪ { s };                              (* mark s as reachable *)
   compute ample(s) satisfying (A1)–(A3);
   mark(s) := ∅;                                            (* taken actions in s *)
   repeat
      s' := top(U); b := b ∧ (s' ⊨ Φ);
      if ample(s') = mark(s') then
         pop(U);                                            (* all ample actions have been taken *)
      else
         let α ∈ ample(s') \ mark(s')
         mark(s') := mark(s') ∪ { α };                      (* mark α as taken *)
         if α(s') ∉ R then
            push(α(s'), U); R := R ∪ { α(s') }              (* α(s') is a new reachable state *)
            compute ample(α(s')) satisfying (A1)–(A3);
            mark(α(s')) := ∅;
         else
            if α(s') ∈ U then ample(s') := Act(s'); fi      (* enlarge ample(s) for (A4) *)
         fi
      fi
   until ((U = ε) ∨ ¬ b)
endproc
```

Experimental results

[Clarke, Grumberg, Minea, Peled, 1999]

Algorithm	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

The core of LTL model checking

- For LTL formula φ, it holds $TS \models \varphi$ iff $TS \otimes \mathcal{A}_{\neg\varphi} \models \Diamond\Box\neg F$
  - where $\mathcal{A}_{\neg\varphi}$ is a nondeterministic Büchi automaton for $\neg\varphi$
  - and F holds in any of its accepting states
- Check $\Diamond\Box\Phi$ efficiently by "nesting" two depth-first searches:
  - the outer DFS looks for reachable $\neg\Phi$-states
  - the inner DFS seeks for backward edges to such states
  - important: start inner DFS on full expansion of $\neg\Phi$-state s in outer DFS

  $\Rightarrow$ in all invocations of inner DFS together each state is visited at most once
- On finding $\neg\Phi$-state: counterexample = concatenation DFS stacks
  - stack U for the outer DFS = path fragment from $s_0 \in I$ to s (in reversed order)
  - stack V for the inner DFS = a cycle from state s to s (in reversed order)

Nested depth-first search with POR

- Generate $\widehat{TS} \otimes \mathcal{A}_{\neg\varphi}$ and check for accepting cycles
- In inner and outer DFS, the same ample sets should be used
- Start inner DFS only if *ample*(s) does not change anymore cf. (A4')
- Abort once state is encountered in inner DFS which is on stack of outer DFS

more details can be found on pages 625 and 626 of book

next: how to compute *ample*(s) satisfying (A1) – (A3)?

Intermezzo: channel systems

- Processes communicate via *channels* ($c \in$ *Chan*)
- Channels are first-in, first-out buffers storing messages
- Channel capacity = maximum $\#$ messages that can be stored
  - if *cap*$(c) > 0$, there is some "delay" between sending and receipt
  - if *cap*$(c) = 0$, then communication via c amounts to handshaking

Actions acting on channels

- Process P_i = *program graph* PG_i + *communication actions*

  $c!e$ transmit the value of expression e along channel c

  $c?x$ receive a message via channel c and assign it to variable x

- $\mathit{Comm} = \{ \, c!e, \; c?x \mid c \in \mathit{Chan}, \; e \in \mathit{Expr}, \; x \in \mathit{Var}. \; \mathit{dom}(x) \supseteq \mathit{dom}(c) = \mathit{dom}(e) \, \}$

- Sending and receiving a message
  - $c!e$ puts the value of e at the rear of the buffer c (if c is not full)
  - $c?x$ retrieves the front element of the buffer and assigns it to x (if c is not empty)
  - if $\mathit{cap}(c) = 0$, channel c has *no* buffer
  - if $\mathit{cap}(c) = 0$, sending and receiving can takes place simultaneously
  - if $\mathit{cap}(c) > 0$, sending and receiving can never take place simultaneously

Channel systems

A program graph over (*Var*, *Chan*) is a tuple

$$PG = (Loc, Act, Effect, \rightarrow, Loc_0, g_0)$$

where

$$\rightarrow \;\subseteq\; Loc \times Cond(Var) \times (Act \cup Comm) \times Loc$$

A *channel system* *CS* over $(\bigcup_{0<i\leqslant n} Var_i, Chan)$:

$$CS \;=\; [PG_1 \mid \ldots \mid PG_n]$$

with program graphs PG_i over $(Var_i, Chan)$

Channel evaluations

- A *channel evaluation* ξ is
  - a mapping from channel $c \in \textit{Chan}$ onto a sequence $\xi(c) \in \textit{dom}(c)^*$ such that
  - current length cannot exceed the capacity of c: $\textit{len}(\xi(c)) \leqslant \textit{cap}(c)$
  - $\xi(c) = v_1 \, v_2 \, \ldots \, v_k$ ($\textit{cap}(c) \geqslant k$) denotes v_1 is at front of buffer etc.

- $\xi[c := v_1 \ldots v_k]$ denotes the channel evaluation

$$\xi[c := v_1 \ldots v_k](c') \;=\; \begin{cases} \xi(c') & \text{if } c \neq c' \\ v_1 \ldots v_k & \text{if } c = c'. \end{cases}$$

- Initial channel evaluation ξ_0 equals $\xi_0(c) = \varepsilon$ for any c

Transition system semantics of a channel system

Let $CS = [PG_1 \mid \ldots \mid PG_n]$ be a *channel system* over $(Chan, Var)$ with

$$PG_i = (Loc_i, Act_i, Effect_i, \rightsquigarrow_i, Loc_{0,i}, g_{0,i}), \qquad \text{for } 0 < i \leqslant n$$

$TS(CS)$ is the *transition system* $(S, Act, \rightarrow, I, AP, L)$ where:

- $S \;=\; (Loc_1 \times \ldots \times Loc_n) \times Eval(Var) \times Eval(Chan)$
- $Act \;=\; \left(\biguplus_{0<i\leqslant n} Act_i\right) \uplus \{\, \tau \,\}$
- $\rightarrow$ is defined by the inference rules on the next slides
- $I = \left\{ \langle \ell_1, \ldots, \ell_n, \eta, \xi_0 \rangle \;\middle|\; \forall i.\ (\ell_i \in Loc_{0,i} \wedge \eta \models g_{0,i}) \wedge \forall c.\ \xi_0(c) = \varepsilon \right\}$
- $AP \;=\; \biguplus_{0<i\leqslant n} Loc_i \;\uplus\; Cond(Var)$
- $L(\langle \ell_1, \ldots, \ell_n, \eta, \xi \rangle) \;=\; \{\, \ell_1, \ldots, \ell_n \,\} \cup \{\, g \in Cond(Var) \mid \eta \models g \,\}$

Inference rules (1)

- Interleaving for $\alpha \in \mathit{Act}_i$:

$$\frac{\ell_i \xrightarrow{g:\alpha} \ell_i' \;\wedge\; \eta \models g}{\langle \ell_1, \ldots, \ell_i, \ldots, \ell_n, \eta, \xi \rangle \xrightarrow{\alpha} \langle \ell_1, \ldots, \ell_i', \ldots, \ell_n, \eta', \xi \rangle}$$

where $\eta' = \mathit{Effect}(\alpha, \eta)$

- Synchronous message passing over $c \in \mathit{Chan}$, $\mathit{cap}(c) = 0$:

$$\frac{\ell_i \xrightarrow{g:c?x} \ell_i' \;\wedge\; \ell_j \xrightarrow{g':c!e} \ell_j' \;\wedge\; \eta \models g \wedge g' \;\wedge\; i \neq j}{\langle \ell_1, \ldots, \ell_i, \ldots, \ell_j, \ldots, \ell_n, \eta, \xi \rangle \xrightarrow{\tau} \langle \ell_1, \ldots, \ell_i', \ldots, \ell_j', \ldots, \ell_n, \eta', \xi \rangle}$$

where $\eta' = \eta[x := \eta(e)]$.

Inference rules (2)

- Asynchronous message passing for $c \in \mathit{Chan}$, $\mathit{cap}(c) > 0$:
  - receive a value along channel c and assign it to variable x:

$$\frac{\ell_i \xrightarrow{g:c?x} \ell_i' \;\wedge\; \eta \models g \;\wedge\; \mathit{len}(\xi(c)) = k > 0 \;\wedge\; \xi(c) = v_1 \ldots v_k}{\langle \ell_1, \ldots, \ell_i, \ldots, \ell_n, \eta, \xi \rangle \xrightarrow{\tau} \langle \ell_1, \ldots, \ell_i', \ldots, \ell_n, \eta', \xi' \rangle}$$

where $\eta' = \eta[x := v_1]$ and $\xi' = \xi[c := v_2 \ldots v_k]$.

  - transmit value $\eta(e) \in \mathit{dom}(c)$ over channel c:

$$\frac{\ell_i \xrightarrow{g:c!e} \ell_i' \;\wedge\; \eta \models g \;\wedge\; \mathit{len}(\xi(c)) = k < \mathit{cap}(c) \;\wedge\; \xi(c) = v_1 \ldots v_k}{\langle \ell_1, \ldots, \ell_i, \ldots, \ell_n, \eta, \xi \rangle \xrightarrow{\tau} \langle \ell_1, \ldots, \ell_i', \ldots, \ell_n, \eta, \xi' \rangle}$$

where $\xi' = \xi[c := v_1\, v_2 \ldots v_k\, \eta(e)]$.

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 $Act_i(s) \subset Act(s)$:

- Nonemptiness condition (A1):
  - check whether process $\mathcal{P}_i$ can perform an action in state s, i.e., $Act_i(s) \neq \varnothing$
- 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 $\ell \overset{g:\alpha}{\hookrightarrow} \ell'$, 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 *TS* equals that of checking $TS' \models \exists\Diamond\, a$ for some $a \in AP$ where $size(TS') \in \mathcal{O}(size(TS))$