On-The-Fly Partial Order Reduction

Lecture #9b of Advanced Model Checking

Joost-Pieter Katoen

Lehrstuhl 2: Software Modeling & Verification

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

May 27, 2009

Outline of partial-order reduction

- During state space generation obtain $\widehat{TS}$
  - a *reduced version* of transition system TS such that $\widehat{TS} \triangleq TS$
  - $\Rightarrow$ 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 $\widehat{TS}$ 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**

$\varnothing \neq$ *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 execution fragment in *TS* such that α depends on *ample*(s). Then: $\beta_i \in$ *ample*(s) for some $0 < i \leqslant n$.

(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 $s_0\, s_1\, \ldots\, s_n$ in $\widehat{TS}$ and $\alpha \in$ *Act*(s_i), for some $0 < i \leqslant n$, there exists $j \in \{ 1, \ldots, n \}$ such that $\alpha \in$ *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 $\widehat{TS}$,

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 under POR (2)

procedure visit (state s)
  push(s, U); $R := R \cup \{ s \}$; (* mark s as reachable *)
  compute *ample*(s) satisfying (A1)–(A3);
  mark$(s) := \varnothing$; (* taken actions in s *)
  repeat
    $s' :=$ *top*(U); $b := b \wedge (s' \models \Phi)$;
    if *ample*$(s') =$ *mark*(s') **then**
      pop(U); (* all ample actions have been taken *)
    else
      let $\alpha \in$ *ample*$(s') \setminus$ *mark*(s')
      mark$(s') :=$ *mark*$(s') \cup \{ \alpha \}$; (* mark α as taken *)
      if $\alpha(s') \notin R$ **then**
        push$(\alpha(s'), U)$; $R := R \cup \{ \alpha(s') \}$ (* $\alpha(s')$ is a new reachable state *)
        compute *ample*$(\alpha(s'))$ satisfying (A1)–(A3);
        mark$(\alpha(s')) := \varnothing$;
      else
        if $\alpha(s') \in U$ **then** *ample*$(s') :=$ *Act*(s'); **fi** (* enlarge *ample*(s) for (A4) *)
      fi
    fi
  until $((U = \varepsilon) \vee \neg b)$
endproc

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 $size(TS') \in \mathcal{O}(size(TS))$

Proof

Overapproximating 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 $\beta \in Act_j$ is independent of $Act_i(s)$ for $i \neq j$

- inspect program graphs PG_j and check whether $(\alpha, \beta) \notin D$ for $\alpha \in Act_i$ and $\beta \in Act_j$
- note: all actions local to PG_i are considered to be dependent

(A2.2) Any $\beta \in Act_i \setminus Act(s)$ 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 $\beta \in Act_i \setminus Act(s)$
- e.g., in $\ell_i \overset{g:\beta}{\hookrightarrow} \ell_i'$ 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 ample(s) = $Act_i(s)$ satisfies (A2)

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

```
if (∃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 ≠ ∅ and Act_i(s) only contains stutter actions) then
  if (∃j ≠ i. Act_i(s) × Act_j(s) ∩ D = ∅) then
    b := true;                                                          (* (A2.1) holds *)
    if ∃ℓ_i --g:β--> ℓ'_i in PG_i where β is a handshaking action then
      b := false;                                                       (* (A2.2) violated *)
    else
      for all ℓ_i --g:β--> ℓ'_i in PG_i and ℓ'_j --h:γ--> ℓ''_j in PG_j with j ≠ i and ℓ_j ↪* ℓ'_j do
        if (η ⊭ 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 $\widehat{TS} \triangleq TS$

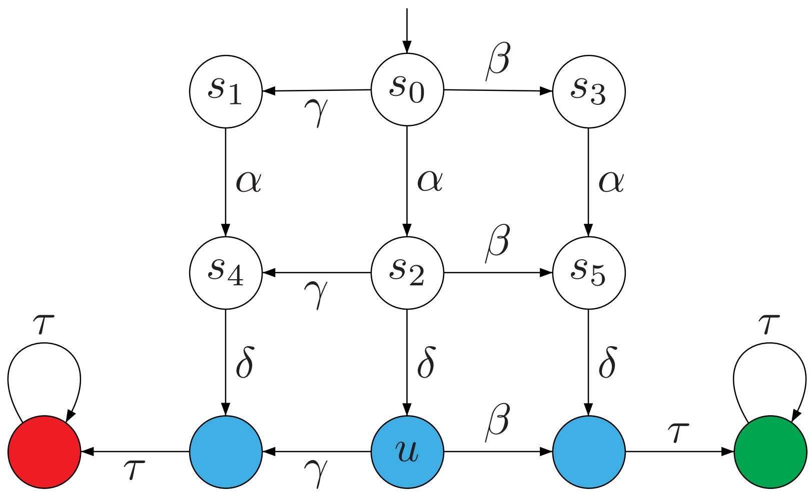
  $\Rightarrow$ this preserves all stutter sensitive LT properties, such as $\mathrm{LTL}_{\setminus \bigcirc}$
  - static partial order reduction: generate $\widehat{TS}$ prior to verification
  - on-the-fly partial order reduction: generate $\widehat{TS}$ during the verification
  - generation of $\widehat{TS}$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

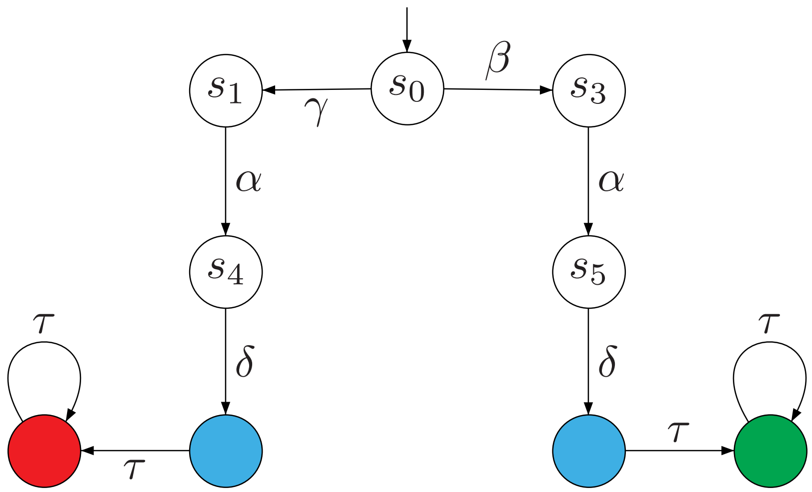
as $\approx^{div}$ is strictly finer than $\triangleq$, try (A1) through (A4)

Example



transition system *TS*

Conditions (A1)-(A4) are insufficient



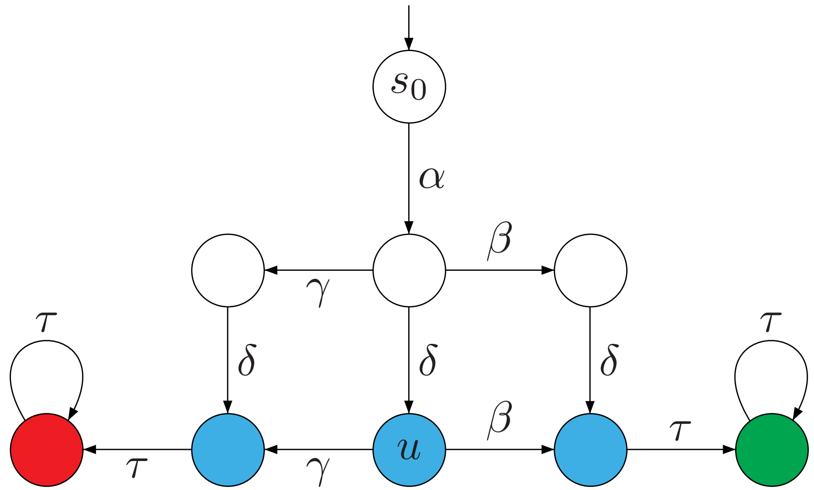
$\widehat{TS} \models \forall\Box\Big(a \;\rightarrow\; (\forall\Diamond b \vee \forall\Diamond c)\Big)$ but *TS* does not and thus $\widehat{TS} \not\approx^{div} TS$

Branching condition

(A5)

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

A sound reduction for $CTL^*_{\setminus\bigcirc}$



$\widehat{TS} \not\models \forall\Box\Big(a \;\rightarrow\; (\forall\Diamond b \vee \forall\Diamond c)\Big)$ and *TS* does not ;in fact $\widehat{TS} \approx^{div} TS$

Correctness theorem

For action-deterministic, finite TS without terminal states:

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

recall that this implies that $\widehat{TS}$ and TS are $\text{CTL}^*_{\setminus \bigcirc}$-equivalent

Ample-set conditions for CTL*

(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 execution fragment in *TS* such that α depends on *ample*(s). Then: $\beta_i \in$ *ample*(s) for some $0 < i \leqslant n$.

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

(A5) **Branching condition**

If *ample*$(s) \neq$ *Act*(s) then $|$*ample*$(s)| = 1$