Model Checking
Lecture #17: Partial-Order Reduction
[Baier & Katoen, Chapter 8]

Joost-Pieter Katoen
Software Modeling and Verification Group
Model Checking Course, RWTH Aachen, WiSe 2019/2020

Overview

1. Motivation
2. Action Independence
3. Ample Sets
4. Correctness and Complexity
5. Summary

Motivation

- Interleaving semantics
  - independent concurrent actions are interleaved
  - a run is defined by a totally ordered sequence of states

- Modelling concurrency by interleaving
  - may enforce an order of actions that has no real “meaning”
  - state space size = product of number of states of threads
  - this is a major cause of the state-space explosion problem

- Partial-order reduction
  - groups runs for which the order of “independent” actions is irrelevant
  - considers a single representative run for equivalent runs
**Motivation**

**Idea of Partial-Order Reduction**

![Transition System](image1.png)

\[ \mathcal{T} = \mathcal{T}_1 \parallel \mathcal{T}_2 \]

\[ \mathcal{T}_{\text{red}} \]

- **Given:** a syntactic description of transition system \( \mathcal{T} \)
- **Aim:** On-the-fly construction of “reduced” transition system \( \mathcal{T}_{\text{red}} \)
  - for state \( s \) only consider outgoing actions \( \text{ample}(s) \subseteq \text{Act}(s) \)
  - expand only \( \alpha \)-successors with \( \alpha \in \text{ample}(s) \)
- **Key issue:** which actions to choose from \( \text{Act}(s) \)?
- **Requirements:**
  - such that \( \mathcal{T}_{\text{red}} \equiv \text{sttrace} \mathcal{T} \), hence \( \mathcal{T}_{\text{red}} \) and \( \mathcal{T} \) are LTL\(_\text{\top}\)-equivalent
  - \( \mathcal{T}_{\text{red}} \) is (much) smaller than \( \mathcal{T} \)
  - \( \mathcal{T}_{\text{red}} \) can be obtained efficiently

### Outline of Ample-Set POR

- \( \text{ample}(s) \subseteq \text{Act}(s) \), the enabled actions in \( s \)
- expand only actions with \( \alpha \in \text{ample}(s) \)

### Stutter Equivalence

**Definition:** stutter step

Transition \( s \rightarrow s' \) in transition system \( \mathcal{T} \) is a **stutter step** if \( L(s) = L(s') \).

**Definition:** stutter equivalence

Paths \( \pi_1 \) and \( \pi_2 \) are **stutter equivalent**, denoted \( \pi_1 \equiv_{\text{sttrace}} \pi_2 \) whenever

\[ \text{trace}(\pi_1) \text{ and } \text{trace}(\pi_2) \text{ are both of the form } A_0^+ A_1^+ A_2^+ \ldots \]

for \( A_i \subseteq \mathcal{AP} \).

For positive integers \( n_1 \) and \( m_1 \):

\[ \text{trace}(\pi_1) = A_0 \ldots A_0 A_1 \ldots A_1 A_2 \ldots A_2 \ldots \]

\[ \text{ trace}(\pi_2) = A_0 \ldots A_0 A_1 \ldots A_1 A_2 \ldots A_2 \ldots \]

\[ n_1 \text{ times} \quad n_1 \text{ times} \quad n_1 \text{ times} \]

\[ m_1 \text{ times} \quad m_1 \text{ times} \quad m_1 \text{ times} \]

**Inventors of Partial-Order Reduction**

- Patrice Godefroid (USA)
- Pierre Wolper (Belgium)
- Antti Valmari (Finland)
- Doron Peled (Israel)
Stutter Trace Equivalence

**Definition: stutter trace equivalence**

Transition systems \( TS_i \) over \( AP \), \( i=1,2 \), are stutter-trace equivalent:

\[ TS_1 \equiv_{\text{sttrace}} TS_2 \] if and only if \( TS_1 \preceq TS_2 \) and \( TS_2 \preceq TS_1 \)

where the stutter trace inclusion relation \( \preceq \) is defined by:

\[ TS_1 \preceq TS_2 \text{ iff } \forall \sigma_1 \in \text{Traces}(TS_1) \left( \exists \sigma_2 \in \text{Traces}(TS_2) . \sigma_1 \equiv_{\text{sttrace}} \sigma_2 \right) \]

Trace-equivalent transition systems are stutter trace-equivalent, but not the converse.

Example: Booking System

\[ \begin{align*}
\mathcal{T} : & \quad \begin{array}{c}
000 \\
100 \\
010 \\
110 \\
001 \\
011 \\
\end{array} \\
\mathcal{T}_{\text{red}} : & \quad \begin{array}{c}
000 \\
100 \\
010 \\
110 \\
001 \\
011 \\
\end{array}
\end{align*} \]

LTL\( \bigcirc \) property:

\[
\square \Diamond \text{ "printer is in state 1"}
\]
Motivation

Example: Booking System

Proposition = “printer is in control state 1”.

\[
\begin{align*}
T & : \quad 000 
\quad 100 
\quad 010 
\quad 110 
\quad 101 
\quad 011 \\
\text{scan} \quad \text{code} \quad \text{price} \quad \text{print} \quad \text{scan} \quad \text{code} \\
\cdots & \quad \cdots & \quad \cdots & \quad \cdots & \quad \cdots & \quad \cdots \\
\end{align*}
\]

\[
\begin{align*}
T_{\text{red}} : \quad 000 
\quad 100 
\quad 010 
\quad 110 
\quad 101 
\quad 011 \\
\text{scan} \quad \text{code} \quad \text{price} \quad \text{print} \quad \text{scan} \quad \text{code} \\
\cdots & \quad \cdots & \quad \cdots & \quad \cdots & \quad \cdots & \quad \cdots \\
\end{align*}
\]

\[
\begin{align*}
TS_{\text{red}} \equiv & \text{strace} \ TS, \text{ hence } TS_{\text{red}} \models \varphi \text{ implies } TS \models \varphi \\
\text{for } \varphi \in \text{LTL}_{\text{i/o}}, \text{ e.g., } \varphi = \square \diamond \text{ “printer is in control state 1”}
\end{align*}
\]
Overview

1. Motivation
2. Action Independence
3. Ample Sets
4. Correctness and Complexity
5. Summary

Action Independence

Definition: action independence

Let $TS$ be an action-independent transition system with action-set $Act$. Actions $\alpha \in Act$ and $\beta \in Act$ are independent in $TS$ if for all states $s$ with $\alpha, \beta \in Act(s)$ the following holds:

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

Example: Semaphore-Based Mutual Exclusion

Let $\alpha(s)$ denote the unique $\alpha$-successor of $s$, i.e., $s \xrightarrow{\alpha} \alpha(s)$.

Action Determinism

Definition: action deterministic

Transition system $TS$ is action deterministic whenever for any state $s$ in $TS$ and action $\alpha$, it holds $s \xrightarrow{\alpha} u$ and $s \xrightarrow{\alpha} t$ implies $u = t$.

Every transition system can be made action deterministic by renaming actions.

Assumption: from now on, transition systems are action deterministic.

Example: Semaphore-Based Mutual Exclusion

Request 1 is independent from the action-set $\{\text{request}_2, \text{enter}_2, \text{release}_2\}$.
Action Independence

Example: Semaphore-Based Mutual Exclusion

dependent actions:
enter₁, enter₂
access both to the semaphore

Example: Shared Variables

\[ y := \neg y \]
\[ x := \neg x \]
\[ \text{if } \neg x \text{ then } z := \neg z \]
\[ \text{action } \alpha \]
\[ \text{action } \beta \]
\[ \text{action } \gamma \]
\[ \text{action } \delta \]
\[ \alpha, \delta \text{ are dependent for } T_{P_1 || P_2} \]

Example: Synchronised Threads

\[ \alpha, \delta \text{ independent } \checkmark \]
\[ \gamma, \delta \text{ independent } \checkmark \]
\[ \beta, \gamma \text{ dependent} \]
Permuting Independent Actions

Let $TS$ be action-deterministic, $s$ 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 a finite run in $TS$ from $s$ with action sequence $\beta_1 \ldots \beta_n$.

Then, for $\alpha \in \text{Act}(s)$ independent of $\{\beta_1, \ldots, \beta_n\}$: $\alpha \in \text{Act}(s_i)$ 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 a run in $TS$ from $s$ with action sequence $\alpha \beta_1 \ldots \beta_n$

Stutter Actions

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

  $$\rho = s_0 \xrightarrow{\beta_1} s_1 \xrightarrow{\beta_2} \ldots \xrightarrow{\beta_n} s_n \xrightarrow{\alpha} t$$
  $$\rho' = s_0 \xrightarrow{\alpha} t_0 \xrightarrow{\beta_1} \ldots \xrightarrow{\beta_n} t_{n-1} \xrightarrow{\beta_n} t$$

  will be distinct

- If $\alpha$ does not affect the state-labelling (= “invisible”): $\rho \equiv_{\text{strace}} \rho'$.

**Definition: stutter action**

Action $\alpha \in \text{Act}$ is a **stutter action** if for each $s \xrightarrow{\alpha} s'$ in $TS$: $L(s) = L(s')$.

Equivalently: $\alpha$ is a stutter action if all transitions $s \xrightarrow{\alpha} s'$ are stutter steps.

Permuting Independent Stutter Actions

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

- $\rho$ a finite run from $s$ with action sequence $\beta_1 \ldots \beta_n \alpha$
- $\rho'$ a finite run from $s$ with action sequence $\alpha \beta_1 \ldots \beta_n$

Then:

if $\alpha$ is a stutter action independent of $\{\beta_1, \ldots, \beta_n\}$, then $\rho \equiv_{\text{strace}} \rho'$.
Adding Independent **Stutter** Actions

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

- $\rho$ an infinite run from $s$ with action sequence $\beta_1 \beta_2 \ldots$
- $\rho'$ an infinite run from $s$ with action sequence $\alpha \beta_1 \beta_2 \ldots$

Then:

if $\alpha$ is a stutter action independent of $\{\beta_1, \beta_2, \ldots\}$, then $\rho \equiv_{\text{strace}} \rho'$.

---

**The Ample-Set Approach**

- Partial-order reduction for LTL formulas using **ample sets**
  - on state-space generation select $\text{ample}(s) \subseteq \text{Act}(s)$
  - such that $|\text{ample}(s)| \ll |\text{Act}(s)|$

- Reduced system $TS_{\text{red}} = (S', \text{Act}, \Rightarrow, I, AP, L')$ where:
  - $S'$ = the set of states reachable from some $s_0 \in I$ under $\Rightarrow$
  - $\Rightarrow$ is the smallest relation defined by:

\[
\begin{align*}
  & s \xrightarrow{\alpha} s' \wedge \alpha \in \text{ample}(s) \\
  \Rightarrow & s \xrightarrow{\alpha} s' \\
  & L'(s) = L(s) \text{ for any } s \in S'
\end{align*}
\]

- **Constraints**: correctness ($\equiv_{\text{strace}}$), effectiveness and efficiency

---

**Which Actions to Put in $\text{ample}(s)$?**

(A1) **Non-emptiness condition**
Select in any state in $TS_{\text{red}}$ at least one action.

(A2) **Dependency condition**
For any finite run in $TS$: an action depending on $\text{ample}(s)$ can only occur after some action in $\text{ample}(s)$ has occurred.

(A3) **Stutter condition**
If an enabled action in $s$ is not selected, then all selected actions are stutter actions.

(A4) **Cycle condition**
Any action in $\text{Act}(s_i)$ with $s_i$ on a cycle in $TS_{\text{red}}$ must be selected in some $s_j$ on that cycle.

(A1) through (A3) apply to states in $S'$; (A4) to cycles in $TS_{\text{red}}$. 

---

**Overview**

1. Motivation
2. Action Independence
3. Ample Sets
4. Correctness and Complexity
5. Summary
Example

Naive Dependency Condition (A2‘)

For any $s \in S'$ with $\text{ample}(s) \neq \text{Act}(s)$:

$\alpha \in \text{ample}(s)$ is independent of $\text{Act}(s) \setminus \text{ample}(s)$.

This is incorrect. (A2’) allows the following reduction:

This is incorrect. (A2’) allows the following reduction:

\[
\begin{align*}
\text{TS} & \not\models \Box \neg a \\
\text{TS}_{\text{red}} & \models \Box \neg a
\end{align*}
\]

so $\text{TS} \not\equiv_{\text{strace}} \text{TS}_{\text{red}}$

Example

Dependency Condition (A2)

Let $s \xrightarrow{\beta_1} s_1 \xrightarrow{\beta_2} \ldots \xrightarrow{\beta_n} s_n \xrightarrow{\alpha} t$ be a finite run in $T S$ such that $\alpha$ depends on $\text{ample}(s)$.

Then: $\beta_i \in \text{ample}(s)$ for some $0 < i \leq n$.

- In every (!) finite run of $T S$, an action dependent on $\text{ample}(s)$ cannot occur before some action from $\text{ample}(s)$ occurs first

- (A2) ensures that for any state $s$ with $\text{ample}(s) \subset \text{Act}(s)$, any $\alpha \in \text{ample}(s)$ is independent of $\text{Act}(s) \setminus \text{ample}(s)$

run $s_0 \xrightarrow{\beta} s_1 \xrightarrow{\alpha} s_4$ violates (A2) as $\gamma$ depends on $\{ \alpha \} = \text{ample}(s_0)$
Properties

For any $\alpha \in \text{ample}(s)$ and $s \in \text{Reach}(TS)$:

if $\text{ample}(s)$ satisfies (A2), then $\alpha$ is independent of $\text{Act}(s) \setminus \text{ample}(s)$.

For finite run $s = s_0 \xrightarrow{\beta_1} \ldots \xrightarrow{\beta_n} s_n$ in $TS$:

if $\text{ample}(s)$ satisfies (A2) and $\{\beta_1, \ldots, \beta_n\} \cap \text{ample}(s) = \emptyset$, then:

$\alpha$ is independent of $\{\beta_1, \ldots, \beta_n\}$ and $\alpha \in \text{Act}(s_i)$ for $0 \leq i \leq n$.

First Consequence of (A1)–(A3)

Let $\varrho$ be a finite run in $\text{Reach}(TS)$ of the form

$\varrho = s \xrightarrow{\beta_1} s_1 \xrightarrow{\beta_2} \ldots \xrightarrow{\beta_n} s_n \xrightarrow{\alpha} t$

where $\beta_i \notin \text{ample}(s)$, for $0 < i \leq n$, and $\alpha \in \text{ample}(s)$.

If $\text{ample}(s)$ satisfies (A1)–(A3), then there exists a run:

$\varrho' = s \xrightarrow{\alpha} t_0 \xrightarrow{\beta_1} t_1 \xrightarrow{\beta_2} \ldots \xrightarrow{\beta_{n-1}} t_{n-1} \xrightarrow{\beta_n} t$

such that $\varrho \equiv_{\text{sttrace}} \varrho'$.

Ample Set Conditions So Far

(A1) Nonemptiness condition

$\emptyset \neq \text{ample}(s) \subseteq \text{Act}(s)$

(A2) Dependency condition

Let $s \xrightarrow{\beta_1} \ldots \xrightarrow{\beta_n} t$ be a finite run in $TS$ such that $\alpha$ depends on $\text{ample}(s)$. Then: $\beta_i \in \text{ample}(s)$ for some $0 < i \leq n$.

(A3) Stutter condition

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

Example: Ample Sets for Semaphore

$\text{AP} = \{c_1, c_2\}$

$\text{ample}(n_1, n_2) = \{\text{request}_1\}$

$\text{ample}(w_1, n_2) = \{\text{request}_2\}$

$\text{ample}(w_1, w_2) = \{\text{enter}_1, \text{enter}_2\}$

$\text{ample}(n_1, n_2, n_2) = \{\text{enter}_1\}$
Second Consequence of (A1)–(A3)

Let \( \rho = s \overset{\beta_1}{\rightarrow} s_1 \overset{\beta_2}{\rightarrow} s_2 \overset{\beta_3}{\rightarrow} \ldots \) be an infinite run in \( \text{Reach}(TS) \) where \( \beta_i \notin \text{ample}(s) \), for \( i > 0 \).

If \( \text{ample}(s) \) satisfies (A1)–(A3), then there exists a run:

\[ \rho' = s \overset{\alpha}{\rightarrow} t_0 \overset{\beta_1}{\rightarrow} t_1 \overset{\beta_2}{\rightarrow} t_2 \overset{\beta_3}{\rightarrow} \ldots \]

where \( \alpha \in \text{ample}(s) \) and \( \rho \equiv \text{sttrace} \rho' \).

Example: Ample Sets for Semaphore

The Necessity of Cycle Condition (A4)

\[ \mathcal{T}_1 \quad \mathcal{T}_2 \quad \mathcal{T} = \mathcal{T}_1 \parallel \mathcal{T}_2 \]

\[ \beta, \alpha_1, \alpha_2 \text{ independent} \]

\( \mathcal{T} \nvDash \square \neg \text{blue} \)

\( \beta, \alpha_i \text{ independent} \)

\[ \mathcal{T}_\text{red} \text{ satisfies (A1), (A2), (A3)} \]

\[ \mathcal{T}_\text{red} \nvDash \square \neg \text{blue} \]

(A4) Cycle condition

For any cycle \( s_0 \ldots s_n \) in \( TS_{\text{red}} \) and \( \alpha \in \text{Act}(s_i) \), for some \( 0 < i \leq n \), \( \alpha \in \text{ample}(s_j) \) for some \( j \in \{1, \ldots, n\} \).

Every enabled action in some state on a cycle in \( TS_{\text{red}} \) must be selected in some state on that cycle.
Ample Set Conditions

(A1) **Nonemptiness condition**
\[ \emptyset \neq \text{ample}(s) \subseteq \text{Act}(s) \]

(A2) **Dependency condition**
Let \( s \xrightarrow{\beta_1} \ldots \xrightarrow{\beta_n} t \) be a finite run in \( TS \) such that \( \alpha \) depends on \( \text{ample}(s) \). Then: \( \beta_i \in \text{ample}(s) \) for some \( 0 < i \leq n \).

(A3) **Stutter condition**
If \( \text{ample}(s) \neq \text{Act}(s) \) then any \( \alpha \in \text{ample}(s) \) is a stutter action.

(A4) **Cycle condition**
For any cycle \( s_0 \ldots s_n \) in \( TS_{\text{red}} \) and \( \alpha \in \text{Act}(s_i) \), for some \( 0 < i \leq n \), \( \alpha \in \text{ample}(s_j) \) for some \( j \in \{1, \ldots, n\} \).

Correctness

Let \( TS \) be a finite, action-deterministic transition system w/o terminal states.

If all ample sets satisfy conditions (A1)–(A4), then \( TS_{\text{red}} \equiv_{\text{strace}} TS \).

Overview

1. Motivation
2. Action Independence
3. Ample Sets
4. Correctness and Complexity
5. Summary

Complexity Considerations

Let \( TS \) be a finite, action-deterministic transition system w/o terminal states.

The worst-case time complexity of checking (A2) in \( TS \) equals that of checking \( TS' \equiv \exists \circ a \) for some \( a \in AP \) where \( \text{size}(TS') \in O(\text{size}(TS)) \).

**Proof.**
Sketch on the black board.

(A1), (A3) and (A4) can relatively easy be incorporated in a DFS-based state-space generation.
Some Experimental Results

[Clarke, Grumberg, Minea, Peled, 1999]

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>$TS$ states</th>
<th>$TS$ transitions</th>
<th>$TS_{red}$ states</th>
<th>$TS_{red}$ transitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>sieve</td>
<td>10,878</td>
<td>35,594</td>
<td>157</td>
<td>157</td>
</tr>
<tr>
<td>data transfer protocol</td>
<td>251,049</td>
<td>648,467</td>
<td>16,459</td>
<td>17,603</td>
</tr>
<tr>
<td>snoopy (cache coherence)</td>
<td>164,258</td>
<td>546,805</td>
<td>29,796</td>
<td>44,145</td>
</tr>
<tr>
<td>file transfer protocol</td>
<td>514,188</td>
<td>1,138,750</td>
<td>125,595</td>
<td>191,466</td>
</tr>
</tbody>
</table>

partial-order reduction works good for loosely-synchronised multi-threaded systems

Summary

POR ignores several interleavings of independent actions in an on-the-fly-manner; i.e., during state-space generation

The ample set method relies on choosing ample(s) $\subseteq$ Act(s) in state s actions not in ample(s) are pruned

(A1) non-emptiness, (A2) dependency, (A3) stutter and (A4) cycle

Conditions (A1) and (A2) ensure that any run in $TS$ can be turned into an equivalent run in $TS_{red}$ by permuting independent actions (and adding independent actions)

(A3) and (A4) ensure that these two runs are stutter equivalent

POR is effective for loosely coupled multi-threaded systems

Next Lecture

Friday January 10, 14:30