Partial Order Reduction

Ralf Huuck

The Problem

Many concurrent components:

Trying to build the product state space ...

State Explosion

Worst case: number of states increases exponentially with number of processes.

What to do?

Try minimizing the effect by reduction heuristics, e.g.:
Partial Order Reduction

Worst case: number of states increases exponentially with number of processes.
Overview

- Informal explanation
- Framework for partial order reduction (POR)
- POR in SPIN
- Summary

Introduction

Motivation

consider interleaving execution, what are the possible runs?

Expanded Asynchronous Product
Expanded Asynchronous Product

Possible Runs

Dependencies (1)

Dependencies (2)

How many runs are in this system?

These 3 plus 3 symmetric ones, i.e., 6

assume x, y are local variables, g is a global variable

Which operations are actually dependent and which are independent?
Equivalent Runs

These 3 runs are equivalent wrt independencies, same for other 3 runs.

Idea

- partitioning into equivalent classes
- we have to select one run in each class only

Necessary Runs

Eliminating all independencies, 2 runs left.

Proving Properties

- \( G(g=0 \lor g>x) \)
- \( F(g \geq 2) \)
- \( (g=0)U(x=1) \)

all hold in reduced graph, i.e., considering only 2 necessary runs.
Proving Properties

- \( G(g=0 \lor g>x) \)
- \( F(g \geq 2) \)
- \( (g=0) \cup (x=1) \)
- \( G(x \geq y) \)

All hold in full and reduced graph, with states of the two necessary runs.

Holds in reduced graph, but not full graph.

Visibility

- Introduces dependency that was not assumed to exist.
- Dependencies not only from data objects but also formula.
- Remove \( x=1, y=1 \) from independencies.

Equivalent Runs

Partition 1

\[ \begin{align*}
&0,0,0 \\
&1,0,0 \\
&0,1,0 \\
&1,1,0 \\
&1,0,2 \\
&0,1,0 \\
&1,1,0 \\
&1,1,2 \\
&1,1,4 \\
\end{align*} \]

\[ \begin{align*}
&x:=1 \ y:=1 \\
&y:=1 \ x:=1 \\
&x:=1 \ y:=1 \\
&g:=g+2 \\
&g:=g+2 \\
&g:=g+2 \\
&g:=g*2 \\
&g:=g*2 \\
&g:=g*2 \\
\end{align*} \]
Equivalent Runs

Questions

• Given a set of processes how can we automatically identify classes of equivalent runs?
• How to avoid full construction upfront, but deciding on-the-fly which states and transitions are necessary?

Such techniques are addressed as partial order reduction, which, e.g., SPIN makes use of.

Theory
Labeled Transition System

(S, s₀, A, τ, Π, L) is labeled transition system where
- S finite set of states
- s₀ initial state
- A finite set of actions
- τ: S × A → S (partial) transition function
- Π finite set of Boolean propositions
- L: S → 2^Π labeling function

(similar to a Kripke structure with symbols on transitions)

enabled/reachable

- action a ∈ A is enabled in state s ∈ S iff τ(a, s) is defined
- enabled(s) denotes set of all actions enabling in transition from state s
- state s is deadlock state iff enabled(s) = Ø
- execution sequence is sequence of subsequent transitions
- state s is reachable iff there exists an execution sequence from s₀ to s

Example

Partial Order Reduction

- avoid construction including “unnecessary” interleavings if possible
- decide per state which outgoing transitions to include
- reduction function r: S → 2^A, i.e., which actions have to be taken care of in a certain state
Reduced LTS

smallest $\left( S_r, s_{0r}, A_r, \tau_r, \Pi_r, L_r \right)$ such that

- $S_r \subseteq S$,
- $s_{0r} = s_{0r}$,
- $L_r = L \cap (S_r \times 2^{\Pi})$
- for any $s \in S_r$ and $a \in \tau(s)$ where $\tau(s,a)$ is defined, $\tau_r(s,a)$ is defined

Independence

two actions $a, b \in A$ ($a \neq b$) are independent
iff for all states $s \in S$ where $\{a, b\} \subseteq \text{enabled}(s)$

1. $b \in \text{enabled}(\tau(s,a))$ and $a \in \text{enabled}(\tau(s,b))$
2. $\tau(\tau(s,a), b) = \tau(\tau(s,b), a)$

This means actions do not disable each other (1) and their permutation leads to the same state (2).

Example

Proving Properties
**Properties**

POR is typically done with respect to certain classes of properties, e.g.:

- absence of deadlock,
- local property, depends on state of a single process or state of single shared object
- next-free LTL property, i.e., LTL with until operator only

**Preserving Deadlock**

To preserve deadlock states the reduction function must satisfy:

- **C₀** \( r(s) = \emptyset \) iff \( \text{enabled}(s) = \emptyset \)
- **C₁ (persistence)** for any execution sequence

\[
S = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \ldots \xrightarrow{a_{n-1}} s_n
\]

with all \( a_i \in r(s) \) \( (0 \leq i < n) \), \( a_{i+1} \) is independent of all \( a_i \in r(s) \)

**Example**

This path can be omitted

**Theorem**

Any reduced system satisfying **C₀** and **C₁** preserves deadlocks.
Local Properties

Property $\phi$ is local if for all $s \in S$ and independent actions $a, b \in A$ if $(a, b) \subseteq \text{enabled}(s)$:
- if $\phi$ holds in $s$ but not in $\tau(s, a)$ then $\phi$ holds in $\tau(s, b)$ but not in $\tau(\tau(s, b), a)$.

Intuition: $\phi$ cannot be changed by the combined effect of two independent actions, it only depends on local changes.

Preserving Local Properties

To preserve local properties the reduction function must satisfy:

**C2 (cycle)** for any cyclic execution sequence

$$s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \ldots \xrightarrow{a_{n-1}} s_n$$

where, $s_n = s_0$ there is an $s_i \ (0 \leq i < n)$ such that $r(s_i) = \text{enabled}(s_i)$.

Example

Two concurrent processes, $a$’s and $b$ are independent.
Full State Graph

a's and b are independent, whenever having the choice between them, why not choosing some a?

Reduced State Graph?

This means, we never see b and never \( \neg p \).

C2 requires in any cycle there is an \( s_i (0 \leq i < n) \) such that \( r(s_i) = enabled(s_i) \).

Therefore, cannot hide \( \neg p \) completely!

Theorem

Any reduced system satisfying C0, C1, and C2 preserves local properties.
Next-free LTL

- only allows Until as temporal operator,
- strict subset of LTL
- cannot, e.g., distinguish between the next and the second next state
- closed under stuttering

Invisibility

\[ \text{prop}(\phi) \text{ set of propositions in } \phi \]

- action \( a \) is \( \phi \)-invisible in \( s \) iff \( \tau(s,a) \) is undefined or \( \pi \in L(s) \iff \pi \in L(\tau(s,a)) \) for all \( \pi \in \text{prop}(\phi) \)
- \( a \) is globally \( \phi \)-invisible iff it is \( \phi \)-invisible for all \( s \in S \)

This means some action cannot change some truth value.

Preserving Next-free LTL

C3 (invisibility) for any state \( s \in S \), all actions are globally \( \phi \)-invisible or \( r(s) = \text{enabled}(s) \)

Example (1)

Which LTL and/or next-free LTL properties do (not) hold here?

More sophisticated examples?
Example (2)

This path can be omitted.

Theorem

Any reduced system satisfying $C_0, C_1, C_2,$ and $C_3$ preserves next-free LTL properties.

Well, yes but ...

• We defined constraints such that a reduced system still satisfies certain properties.
• But: How to find a suitable reduction?
• Also: building full state graph and then reducing is inefficient.

Challenging!

Let’s have a look at SPIN ...

POR in SPIN
1. depth first search
2. reduction function based on process structure

Preliminaries

(S, s₀, A, τ, Π, L) full LTS from set of processes P
each process P ∈ P is set of actions, i.e., P ⊆ A

we assume: P is a partitioning of A, i.e,
1. P, Q ∈ P, P ≠ Q ⇒ P ∩ Q = Ø, and
2. A = ∪ P ∈ P P

Pid: A → P returns process (ID) for a given action

Restriction of Process Structure

We do not allow concurrency within a process:

for all a, b ∈ P, a ≠ b, s ∈ S:
a, b ∈ enabled(s) ⇒ b ∈ enabled(τ(s, a))

This means we still have choice (if-then-else) in a process,
but no processes within processes.

Safety

Action a is safe
iff
it is independent from any b where Pid(a) ≠ Pid(b)
Which actions are safe in this example?

They are independent of any action in other process.

Action \( a \) is safe
\[ \text{iff} \]
\( \text{it is independent from any } b \text{ where } \text{Pid}(a) \neq \text{Pid}(b) \)

Action \( a \) is next-free safe for some \( \phi \in \text{LTL}_x \)
\[ \text{iff} \]
\( \text{it is independent from any } b \text{ where } \text{Pid}(a) \neq \text{Pid}(b), \) and
\( \text{globally } \phi \text{-invisible} \)

Which actions are next-free safe for:
- \( G \ (g=2) \)
- \( G \ (x<g) \)
Next-free Safe Example

\[ S_0 \]
\[ x:=1 \quad g:=g+2 \]
\[ S_1 \]
\[ y:=1 \quad g:=g^2 \]
\[ S_2 \]

next-free safe actions for \( g=2 \)

Other (counter)examples?

Reduction Function Ample (part 1)

Let \( s \in S \) be a state. Let \( P \in P \) be a process such that
1. \( \text{enabled}(s) \cap P \neq \emptyset \)
2. for all \( a \in \text{enabled}(s) \cap P \), \( a \) is (next-free) safe
3. for all \( a \in \text{enabled}(s) \cap P \), \( \tau(s,a) \) is not on DFS stack

Reduction Function Ample

Let \( s \in S \) be a state. Let \( P \in P \) be a process such that
1. \( \text{enabled}(s) \cap P \neq \emptyset \)
2. for all \( a \in \text{enabled}(s) \cap P \), \( a \) is (next-free) safe
3. for all \( a \in \text{enabled}(s) \cap P \), \( \tau(s,a) \) is not on DFS stack

Reminder: DFS Algorithm

Hash table:
\[ q_1 \quad q_2 \quad q_4 \]

Stack:
\[ q_1 \quad q_2 \quad q_4 \]

Remember DFS algorithm? Stack keeps record of states we have seen before, but not fully explored.
Let $s \in S$ be a state. Let $P \in P$ be a process such that
1. $\text{enabled}(s) \cap P \neq \emptyset$
2. for all $a \in \text{enabled}(s) \cap P$, $a$ is (next-free) safe
3. for all $a \in \text{enabled}(s) \cap P$, $\tau(s,a)$ is not on DFS stack

We define a reduction function ample as follows:
• if there is no such process then $\text{ample}(s) = \text{enabled}(s)$.
• otherwise choose arbitrary $P$ satisfying above requirements and define $\text{ample}(s) = \text{enabled}(s) \cap P$.

Example (POR deadlock)

What are the ample sets?
Consider simple safety only.
Example (2)

\[ \varphi = F \{ g = 2 \} \]

Reduction (2)

\[ \varphi = F \{ g = 2 \} \]

Example (3)

\[ \varphi = F \{ x < y \} \]

On-the-fly Construction

Constructing full state space first and then reducing it is not very smart, but:

- We can do POR while constructing the state space.

Basically, use DFS algorithm for state space construction and only follow the paths in the ample sets.

POR does not always help, but the more independent actions the better.
Summary

Partial Order Reduction

- tackles state explosion
- general framework for reduction
- SPIN example for implementation of reduction function
- other methods out there, e.g., symmetry reduction, automata minimizations, abstractions etc.

Good news 😊

We are done with "standard" model checking.