Inseguendo Fagiani Salvatici

Partial Order Reduction for Guarded Command Languages

Frank de Boer[1], Einar Broch Johnsen[2],
Rudolf Schlatte[2], S. Lizeth Tapia Tarifa[2], Lars Tveito[2]

[1] CWI, Amsterdam, the Netherlands
f.s.de.boer@cwi.nl

[2] University of Oslo, Norway
{einarj, rudi, sltarifa, larstvei}@ifi.uio.no

DIP 2020, 27 November 2020

UNIVERSITY OF OSLO

SIRIUS

sfi Centre for Research-based Innovation
The Research Council of Norway

The Allegory of the Wine-Maker

- A vineyard can be seen as a metaphor for a parallel execution space
- The wine-maker can take different paths on his way towards a final state
- On some paths, he may encounter the dreaded pheasants
- Manoeuvering through these paths reveals a complex search space

Partial Order Reduction (POR) [Clarke et al 2000, Godefroid 96]

POR is a technique to systematically search for reachable states while eliminating equivalent paths

Example 1	ev_1	ev_2	Scheduling traces:
`x↦1,y↦2, { x := x+1 || y := 3 }`			$\theta_1 = ev_1 \cdot ev_2$ $\theta_2 = ev_2 \cdot ev_1$
Example 2	ev_3	ev_4	
`x↦1, { x := x+1 || x := 3 }`			$\theta_3 = ev_3 \cdot ev_4$ $\theta_4 = ev_4 \cdot ev_3$

Are the executions equivalent?

Let $[\theta]$ be the set of traces equivalent to θ (ref. Mazurkiewicz trace theory).
We can reason about equivalence purely in terms of the scheduling traces
if the equivalence relation is strong enough to ensure the following theorem:

Theorem (Godefroid 96)

If $s_0 \xrightarrow{\theta_1} s_1$, $s_0 \xrightarrow{\theta_2} s_2$ *and* $\theta_2 \in [\theta_1]$, *then* $s_1 = s_2$.

From POR to Testing for Locally Deterministic Behavior

Testing theory for actors languages: Are actors locally deterministic?

Given a scheduling trace θ_1 reflecting the execution of an actor,
can we construct another trace θ_2 such that $\theta_2 \notin [\theta_1]$?

We are interested in this problem for processes which are **not always enabled**;
for example, processes may be **guarded by Boolean expressions.**
For a single actor, this problem corresponds to guarded commands:

$$\overset{ev_1}{\qquad\qquad}\qquad\qquad\overset{ev_2}{\qquad\qquad}$$

```
x↦1, { x < 2 ▷ x := x+1 || x := 3 }
```

Assume that we have executed the trace $\theta_1 = ev_1 \cdot ev_2$.
Clearly $\theta_2 = ev_2 \cdot ev_1 \notin [\theta_1]$.

However, θ_2 is not a scheduling trace for this program
— it does not correspond to any execution!

GCL and Symbolic Traces

$$
\begin{array}{rcl}
\mathit{Prog} & ::= & \sigma,\ g \\
\sigma \in \mathit{State} & ::= & \epsilon \mid \sigma[x \mapsto v] \\
g \in \mathit{GrdStm} & ::= & \texttt{skip} \mid g; g \\
& \mid & e \rhd s \mid s \lhd e \rhd s \\
s \in \mathit{Stm} & ::= & x := e \mid \texttt{spawn}(g) \\
e \in \mathit{Exp} & ::= & \texttt{True} \mid \texttt{False} \mid x \mid v \\
& \mid & \mathit{op}(e, \ldots, e)
\end{array}
$$

Need scheduling traces which are sufficiently expressive to determine both the equivalence classes and guard enabledness

- We define a concolic operational semantics for GCL
- Let the scheduling trace capture the symbolic linearization of the program
- We can then compute the path condition for a trace by backwards reasoning

$$
\frac{\sigma(e) = \texttt{True} \quad \sigma' = \sigma[x \mapsto \sigma(e')]}{\sigma, \{\iota(e \rhd x := e'; g)\} \xrightarrow{\iota(e \rhd x := e')} \sigma', \{\iota(g)\}} \ (\mathit{Assign})
$$

POR for GCL

Assume that we have observed a scheduling trace

$$\theta_1 = ev_1 \cdot ev_2 \cdot ev_3 \cdot ev_4 \cdot ev_5 \cdot ev_6 \cdot ev_7 \cdot ev_8 \cdot ev_9 \ldots$$

We can construct a new **candidate test case** by permuting events:

$$\theta_2 = ev_1 \cdot ev_2 \cdot ev_3 \cdot ev_5 \cdot ev_4 \cdot ev_6 \cdot ev_7 \cdot ev_8 \cdot ev_9 \ldots$$

Is θ_2 an actual test case?

- Is $\theta_2 \notin [\theta_1]$? Define an interference relation on scheduling events
- Is θ_2 executable? Compute the weakest precondition over the path condition of θ_2 to find an initial state for the trace

$$\begin{aligned} &path(\iota_1(g_1 \rhd \texttt{x := e}) \cdot \iota_1(g_2 \rhd \texttt{spawn}(\iota_2)) \cdot \iota_2(g_3 \rhd s_2) \cdot \iota_1(g_4 \rhd s_3) \cdot \ldots) \\ &\quad = g_1 \wedge path(\iota_1(g_2 \rhd \texttt{spawn}(\iota_2)) \cdot \iota_2(g_3 \rhd s_2) \cdot \iota_1(g_4 \rhd s_3) \cdot \ldots)[x/e] \\ &\quad = g_1 \wedge g_2[x/e] \wedge path(\iota_2(g_3 \rhd s_2) \cdot \iota_1(g_4 \rhd s_3) \cdot \ldots)[x/e] = \ldots \end{aligned}$$

Conclusions

Contributions of the paper

- **Formalize** POR for GCL in terms of traces, concolic execution and weakest precondition reasoning
- **Fagiani Algorithm:** algorithm which systematically generates test cases for a GCL program based on an observed scheduling trace
- **Prototype implementation:** Maude and Python
- **Language extensions:** nested guards, procedures, local scopes, loops

Context of this work

- This paper contributes to our work on **ExoDPOR**, a parallel stateless model checker for ABS, purely based on traces and record & replay supports
- POR algorithms are normally implemented in terms of runtime structure rather than scheduling traces (e.g., DPOR). This means that execution paths are explored sequentially

Questions?