Improving Partial Order Reductions in Multithreaded Programs with Local First Search

Hernán Ponce de León[1], Cristian Rosa[2], and Keijo Heljanko[1]

[1] Helsinki Institute for Information Technology HIIT and Department of Computer Science, School of Science, Aalto University, Finland
{hernan.poncedeleon,keijo.heljanko}@aalto.fi

[2] CIFASIS, Rosario, Argentina
rosa@cifasis-conicet.gov.ar

Exploring the state space of a multithreaded program in an efficient manner is a fundamental problem in software verification. Concurrent transitions interleave in many ways quickly generating many equivalent but unequal states leading to the well known state space exposition problem. Two prominent approaches to deal with this problem are partial order reduction techniques (PORs) and unfoldings methods.

PORs techniques [1, 3, 4, 10] establish an equivalence relation between executions of the programs and explore a subset of all possible interleavings preserving at least one representative per equivalence class. At every state, they execute a subset of active or enabled transitions; those transitions can be computed statically [4] or dynamically [3]. Recently, an improvement to these methods have been proposed leading to an optimal PORs in the sense that exactly one execution is explored for each Mazurkiewicz trace [1] . All these approaches represent the possible executions of the program as a computation tree and prune some of its branching (once an equivalent branch has already been visited).

Unfoldings techniques [5, 6] models executions with partial orders together with a conflict relation to distinguish between different executions of the system. Both PORs and unfoldings techniques have shortcomings, but surprisingly, promising solutions for a given technique can be found in the opposite approach. For example PORs inexpensively add events to the current execution while computing possible extensions is the most demanding part of unfoldings techniques; on the other hand, explorations of repeated states and pruning of non-terminating executions is elegantly achieved in unfoldings with cut-off events.

The advantages of both approaches have been exploited together for the first time in [9] where they propose a technique that matches the test suite size of [1] but it explores an event structure rather than a computation tree. The former has a richer structure provided by a tree-like structure of partial orders. The use of partial order avoids the explicit enumeration of the order between concurrent or independent transitions of the program. The use of event structures suggests that improvements can be done to generate further reductions if we just intend to preserve local reachability [5].

Formally, an event structure is a tuple formed by a set of events E, a partially ordered relation $\leq$ called causality (representing dependencies) and a symmetric and antireflective relation # called conflict which is inherited w.r.t causality, i.e. $e_1 \leq e_2$ and $e_1 \# e_3$ implies $e_2 \# e_3$. Events not related by $\leq$ or # are called concurrent. The executions of an event structure are captured by its configurations, a causally-closed and conflict-free subset of events. Figure 1 shows the Hasse diagram of an event structure with nine events (transitive causalities or inherited conflict are removed for clarity); every event depends on $\bot$, e.g. $\bot \leq 1$ and $\bot \leq 4$ (since $\leq$ is transitive); events 1 and 3 cannot belong to a same configuration since they are in conflict, i.e 1#3; events 1 and 5 are concurrent. This event structure has four maximal configurations $\{\bot, 1, 2, 5, 6\}, \{\bot, 1, 2, 7, 8\}, \{\bot, 3, 4, 5, 6\}$ and $\{\bot, 3, 4, 7, 8\}$.

The unfoldings semantics of a program can be expressed as an event structure [9]; while independent[1] transitions give rise to concurrent events, dependent ones generate causally dependent or conflicting events depending if they belong or not to the same execution. Figure 1 shows a program with 4 threads accessing two global variables x and y; each pair of threads access a single variable by reading or writing it. Clearly the access to different variables is independent, thus $x = 5 \diamond y = 1, x = 5 \diamond c = y, b = x \diamond y = 1$ and $b = x \diamond c = y$; access to the same variable are dependent, i.e. $x = 5 \not\diamond b = x$ and $y = 1 \not\diamond c = y$. The unfolding semantics of this program is given by the event structure of Figure 1. Each of the four maximal configuration corresponds to a deadlocking execution of the program. For example the configuration $\{1, 2, 5, 6\}$ corresponds to the execution where variable x is written and then read followed by variable y being written and read.

```
Global variables:
int x,y = 0;

Thread 1:           Thread 2:       Thread 3:       Thread 4:
local b = x;        x = 5;          local c = y;    y = 1;
```

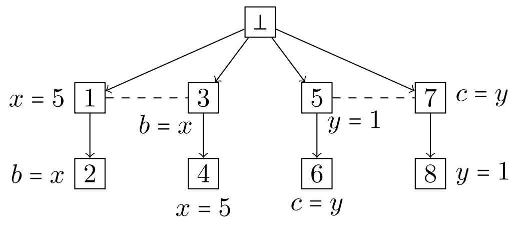

Figure 1: A multithreaded program and its unfolding semantics.

The optimality of [1, 9] states that no algorithm can explore less executions while preserving all Mazurkiewicz or deadlocking traces. If one instead targets at covering all local states of threads (which is sufficient to test for example local properties), it is not necessary any more to execute all Mazurkiewicz traces, but to cover every event of the event structure [5]. While computing the minimal set of executions to cover every local state is a very hard problem [8], we are interested at techniques that generate further reductions than those of [1, 9]. For the program generating the event structure of Figure 1, both optimal PORs explore four executions corresponding to the four maximal configurations. However if we are interested just in covering all the possible values local variables b and c might have, it is sufficient to explore only two executions, for example, all read transitions first in one execution and all write transitions first in another one. The first execution covers events 3,4,7,8 while the second one covers 1,2,5,6. To achieve this kind of reduction, we propose to use local first search [2, 7] on top of the unfolding-based PORs.

Local first search (LFS) was designed to optimize the search for local properties in transitions systems. The technique characterizes a restricted subset of traces that need to be explored to check local properties. For an event structure this means that only maximal events and their causal predecessors (those are called local or prime configurations) need to be explored. Since

[1]The independence relation arising from the program is denoted by $\diamond$; while its complement (the dependence relation) is denoted by $\not\diamond$.

the POR algorithms does not have complete information about the whole event structure (the event structure is constructed while the program is "being unfolded"), LFS performs an analysis to detect non prime configurations as soon as possible to avoid their exploration. This is based on a combinatorial aspect of the independence alphabet of the program. While the unfolding-based POR algorithm could explore the execution $1 \cdot 5 \cdot 2 \cdot 6$, we can detect (adding LFS) that the sub-configuration $\{1, 5\}$ does not lead to a prime configuration and stop the exploration. Unfolding-based POR with LFS only explores the executions $1{\cdot}2, 3{\cdot}4, 5{\cdot}6$ and $7{\cdot}8$. This approach explores shorter or smaller configuration, but still four executions are needed. However, it can be observed that those four configuration can be merged into, for example, $1{\cdot}2{\cdot}5{\cdot}6$ and $3{\cdot}4{\cdot}7{\cdot}8$, but doing this during the exploration needs further algorithmics. This is similar to the problem of obtaining the minimal test suite to test a multithreaded program [8].

While [5] generates further reductions in the number of executions than the PORs techniques, it still relies on the construction of a Petri net unfolding and as such it suffers the computational cost of computing possible extensions. Since LFS can be used on top of the POR technique from [9], we believe this approach generates a good trade-off between the reduction in the size of the obtained test suite and the computational cost of the exploration.

References

[1] P. A. Abdulla, S. Aronis, B. Jonsson, and K. F. Sagonas. Optimal dynamic partial order reduction. In *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, San Diego, CA, USA, January 20-21, 2014*, pages 373–384, 2014.

[2] S. Bornot, R. Morin, P. Niebert, and S. Zennou. Black box unfolding with local first search. In *Tools and Algorithms for the Construction and Analysis of Systems, 8th International Conference, TACAS'02, Proceedings*, volume 2280 of *Lecture Notes in Computer Science*, pages 386–400. Springer, 2002.

[3] C. Flanagan and P. Godefroid. Dynamic partial-order reduction for model checking software. In *Proceedings of the 32nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2005, Long Beach, California, USA, January 12-14, 2005*, pages 110–121, 2005.

[4] P. Godefroid. *Partial-Order Methods for the Verification of Concurrent Systems - An Approach to the State-Explosion Problem*, volume 1032 of *Lecture Notes in Computer Science*. Springer, 1996.

[5] K. Kähkönen. *Automated Systematic Testing Methods for Multithreaded Programs.* Doctoral dissertation, School of Science, Aalto University, 2015.

[6] K. L. McMillan. A technique of state space search based on unfolding. *Formal Methods in System Design*, 6(1):45–65, 1995.

[7] P. Niebert, M. Huhn, S. Zennou, and D. Lugiez. Local first search - A new paradigm for partial order reductions. In *12th International Conference on Concurrency Theory, CONCUR'01, Proceedings*, volume 2154 of *Lecture Notes in Computer Science*, pages 396–410. Springer, 2001.

[8] H. Ponce de León, O. Saarikivi, K. Kähkönen, K. Heljanko, and J. Esparza. Unfolding based minimal test suites for testing multithreaded programs. In *15th International Conference on Application of Concurrency to System Design, ACSD 2015, Brussels, Belgium, June 21-26, 2015*, To appear.

[9] C. Rodríguez, M. Sousa, S. Sharma, and D. Kroening. Unfolding-based partial order reduction. In *26th International Conference on Concurrency Theory, CONCUR'15, Proceedings*, LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015. To appear.

[10] A. Valmari. The state explosion problem. In *Lectures on Petri Nets I: Basic Models, Advances in Petri Nets, the volumes are based on the Advanced Course on Petri Nets, held in Dagstuhl, September 1996*, pages 429–528, 1996.