CPOR-Spin: Spin with Cluster-based Partial Order Reduction

Twan Basten, Dragan Bošnački, and Marc Geilen

Eindhoven University of Technology, PO Box 513, NL-5600 MB Eindhoven, The Netherlands
{a.a.basten,d.bosnacki,m.c.w.geilen}@tue.nl

Abstract. Partial order reduction is one of the main weapons that the model checker Spin uses against the state space explosion problem. In this paper, we present CPOR-Spin, an extension of Spin which features an improved version of the partial order reduction algorithm. The partial order reduction scheme of Holzmann and Peled, which is implemented in the standard version of the tool, is enhanced using clustering of processes based on the hierarchical structure of the verified system.

Introduction. Spin is a software package for the verification of distributed systems [5, 7]. The full presentation of Spin and its input language Promela is beyond the scope of this paper and we refer the reader to [5, 7] for more detailed information. Here we give only a very brief overview of the main features. The syntax of Promela, Spin's input language, is derived from C, but also uses the denotations for communications from Hoare's CSP and control flow statements based on Dijkstra's guarded commands. In Promela, the system components are modeled as *processes* that can communicate via global *variables*. Given as input a Promela model, Spin can do random or interactive simulations, or perform a validation of the system by scanning the state space. The validation can be done for *safety* properties (invalid end-states and assertions) as well as *liveness* properties (formulated as non-progress cycles, linear-time temporal logic (LTL) formulae, or Büchi automata.) The simplest way to scan the state space is through an exhaustive search. In order to handle large state spaces, i.e., to cope with the infamous state space explosion problem, some more sophisticated methods are used.

Partial Order Reduction. Among the above mentioned reduction methods, partial order reduction (POR) [3, 6, 4] takes one of the most prominent places. POR exploits the independence of the checked property from the execution order of the process actions (Promela statements). More specifically, two actions a, b are independent precisely when, for all sequences of actions v, w, if $vabw$ is a valid execution sequence of the system, then $vbaw$ is an execution of the system as well. A typical example of independent actions are the actions that assign a value to a local variable. The order of the execution of two such statements does not affect the verification outcome. In a sense, instead of checking all the execution sequences, the desired property is checked only on representative sequences, which results in significant savings in space and time.

Optimally deciding independence of statements can be as difficult as the original verification problem (e.g., [3]). Consequently, in practice only sufficient conditions for independence are used that can be checked locally and preferably using only syntactic criteria, i.e., directly from the system specification. The actual reduction of the state space is realized during the state space exploration by limiting the search from a given state to only a subset of the actions that are executable in that state. Such a subset is called an *ample set*. A key property of an ample set is that all executable actions that it possibly ignores are independent from the selected actions. In the standard Spin POR algorithm [6], an ample set that is a strict subset of all executable actions (i.e., that realizes a true reduction) consists of actions of only one process. Spin requires that all actions of such an ample set be *safe*. Safeness includes the above mentioned independence property: an action is safe only if it is independent of all actions in other processes. (Besides the independence, safe actions should satisfy some additional conditions [6]. However, our improvement is based on relaxation of the independence requirement.)

Cluster-based Partial Order Reduction. In order to grasp the intuition behind our improvement of Spin's POR algorithm, consider the two specifications of the same concurrent system given in Fig. 1. Specification `example1` has two global variables and four processes, each of them executing a single action assigning a value to one of these variables. None of the actions is independent of all the other actions in the other processes. Thus, during state space exploration we cannot find a process which has only safe actions and use it to obtain a reduction.

```
System example1
int u, v;
P0: u:=0; /* u0 */
P1: u:=1; /* u1 */
P2: v:=0; /* v0 */
P3: v:=1; /* v1 */
```

```
System example2
C0: int u;
    P0: u:=0; /* u0 */
    P1: u:=1; /* u1 */
C1: int v;
    P2: v:=0; /* v0 */
    P3: v:=1; /* v1 */
```

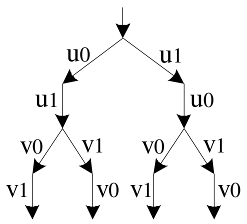

Fig. 1. Two specifications of a concurrent system and a reduced state space.

Specification `example2` is a variant of `example1` with the processes *clustered* in pairs. The two clusters encapsulate the dependencies between processes. As a result, all actions within one cluster are independent of all actions within the other one. The idea of cluster-based POR is now surprisingly simple: A cluster of processes whose executable actions are independent of all other actions takes the role of a single process in the standard POR algorithm. Thus, an ample set may contain actions of an entire cluster as long as they are independent of all actions enabled outside that cluster. Intuitively, with this modification all interleavings of actions within a cluster are still explored. However, the limited interleaving between actions from different clusters results in state space reduction. For `example2`, one possible reduction obtained in this way is shown in Fig. 1. In the initial state the ample set consists of the processes (actions) from cluster `C0`. Consequently, only four execution sequences are explored: $\langle u0, u1, v0, v1\rangle$, $\langle u0, u1, v1, v0\rangle$, $\langle u1, u0, v0, v1\rangle$, and $\langle u1, u0, v1, v0\rangle$.

More details about the theoretical background of cluster-based POR can be found in [1, 2]. Among others, it is shown that the clustering technique can be used to define reductions that preserve deadlocks, local assertions, as well as next-time-free LTL properties. In the remainder, we concentrate only on the extensions of Promela and the implementation.

Extended Promela. A process in Promela models is an instance of a `proctype` – a process template. Promela features only a simple C-like variable scoping. In other words, it distinguishes only between local variables and global variables, declared inside and outside proctypes, respectively. To represent system hierarchy, we need a more refined Pascal-like variable scoping hierarchy. To this end, we extend Promela with two new keywords `cluster` and `instance`. These extensions are purely syntactical, exploited only in our enhanced POR algorithm during verification, and do not change the semantics of the language. The `cluster` construct groups clusters, proctype definitions, and global variables in a straightforward way. The `instance` keyword is used to avoid multiplication of code when multiple processes are instances of the same proctype. For instance, the example of Fig. 1 can be coded in extended Promela as shown in Fig. 2. The variables `u,v` used in Fig. 1 correspond to `u[0],u[1]`, respectively. Process instances are given ids for the purpose of identification.

```
cluster C0 {
int u[2];
proctype P0(int i) {u[i] = 0;}
proctype P1(int i) {u[i] = 1;}
}

cluster C1 {
instance P0 {};
instance P1 {};
}
```

```
init
{
atomic{
   /* cluster C0 */
   run P0(0); run P1(0);
   /* cluster C1 */
   run P0(1); run P1(1);
}
}
```

Fig. 2. The example of Fig. 1 in extended Promela.

Implementation. Our CPOR extension is implemented on top of Spin, version 3.2.4. The clustering hierarchy is deduced from the Promela model using the language extensions described above. A clustering hierarchy consists of *levels* of clusters. For instance, the clustering hierarchy for our running example consists of three levels. At the lowest level, level 0, each process is a cluster of its own, yielding clusters $\{\mathtt{P0(0)}\}$, $\{\mathtt{P1(0)}\}$, $\{\mathtt{P0(1)}\}$, and $\{\mathtt{P1(1)}\}$. The next level, level 1, consists of two clusters, namely $\mathtt{C0} = \{\mathtt{P0(0)}, \mathtt{P1(0)}\}$ and $\mathtt{C1} = \{\mathtt{P0(1)}, \mathtt{P1(1)}\}$. Finally, the highest (trivial) level 2 has only one cluster that contains all the processes: $\{\mathtt{P0(0)}, \mathtt{P1(0)}, \mathtt{P0(1)}, \mathtt{P1(1)}\}$. Depending on the system, more complex clustering hierarchies can be formed by nesting clusters in an obvious way. Each level in such clustering hierarchies is a coarsening of all lower levels.

Using the notion of a clustering hierarchy we can easily generalize Spin's notion of safety. Instead of the binary notion safe-unsafe, we introduce the notion of level-i safety. The crucial point is that an enabled action can only be level-i safe if it is independent of any other enabled action *from another cluster* at level i. Thus, level-i safety is a true generalization of the standard notion of safety, which requires independence of any action *from another process*. In the above clustering hierarchy for our running example, all actions are level-1 and (trivially) level-2 safe, but not level-0 safe.

Our derivation of the safety levels of actions is based on syntactic criteria. We use the scopes of variables for that purpose. If a variable is completely within the scope of some cluster at level i in the hierarchy, it is said to be a level-i variable. Note that a level-i variable is also level-j for any level j greater than i. A Promela statement is level-i safe if and only if it touches only level-i variables. Thus, safety levels of variables can be efficiently calculated before the state space exploration is started. A run-time action inherits the safety level of the statement it is derived from.

What remains is how to compute ample sets during the state space traversal. The largest state space reductions are obtained by choosing the ample sets as small as possible. Thus, a simple way to obtain small ample sets is to search the levels in a clustering hierarchy for suitable clusters in increasing order starting from the trivial process level.

Experiments. We tested CPOR-Spin versus standard Spin with POR on several examples, including a directory-based cache-coherency protocol. In all test cases, CPOR-Spin was able to achieve much better state space reductions than standard Spin (measured in the number of explored states and transitions); in several cases, an improvement of several orders of magnitude was obtained. The least improvement obtained was 21%, whereas the largest improvement was over 99%. The reader is referred to $[1, 2]$ for detailed results. As a consequence of the reductions, also verification times improved.

Future Work. We are aware of two incompatibilities between CPOR-Spin and standard Spin: CPOR-Spin does not allow dynamic creation of processes, and also communication channel variables are not real variables, i.e., they can be used only as named constants. Although many interesting classes of systems can still be modeled without these two features, it remains a challenging problem to reconcile the latter with CPOR.

Availability. The tool, together with some additional information and several examples, is available from `http://www.win.tue.nl/~dragan/CPOR-Spin`.

Acknowledgments: We thank Gerard Holzmann for patiently answering numerous questions about the implementation of Spin's POR algorithm.

References

1. T. Basten and D. Bošnački. Enhancing Partial-Order Reduction via Process Clustering. In *Automated Software Engineering, ASE 2001, 16th. IEEE International Conference, Proceedings*, pages 245–253. IEEE Computer Society Press, 2001.
2. T. Basten, D. Bošnački, and M.C.W. Geilen. Cluster-based Partial-order Reduction. *Automated Software Engineering*. Submitted.
3. E.M. Clarke Jr., O. Grumberg, and D.A. Peled. *Model Checking*. MIT Press, 2000.
4. P. Godefroid. *Partial-Order Methods for the Verification of Concurrent Systems: An Approach to the State-Explosion Problem*. Lecture Notes in Computer Science 1032. Springer, 1996.
5. G.J. Holzmann. The model checker Spin. *IEEE Transactions on Software Engineering*, 23(5):279–295, 1997.
6. G.J. Holzmann and D. Peled. An Improvement in Formal Verification. In D. Hogrefe and S. Leue, editors, *Formal Descriptions Techniques VII, FORTE '94*, pages 197–211. Chapman & Hall, 1995.
7. SPIN. http://spinroot.com/. Last checked on 28/01/2002.