Milestones: A Model Checker Combining Symbolic Model Checking and Partial Order Reduction*

José Vander Meulen[1] and Charles Pecheur[2]

[1] Université catholique de Louvain, jose.vandermeulen@uclouvain.be
[2] Université catholique de Louvain, charles.pecheur@uclouvain.be

Abstract. Symbolic techniques and partial order reduction (POR) are two fruitful approaches to deal with the combinatorial explosion of model checking. Unfortunately, past experience has shown that symbolic techniques do not work well for loosely-synchronized models, whereas, by applying POR methods, explicit-state model checkers are able to deal with large concurrent models. This paper presents the Milestones model checker which combines symbolic techniques and POR. Its goal is to verify temporal properties on concurrent systems. On such a system, Milestones allows to check the absence of deadlock, LTL properties, and CTL properties. In order to compare our approach to others, Milestones is able to translate a model into an equivalent Spin model [7] or NuSMV model [4]. We briefly present the theoretical foundation on which Milestones is based on. Then, we present the Milestones model checker, and an evaluation based on an example.

1 Introduction

Two common approaches are commonly exploited to fight the combinatorial state-space explosion problem in model checking. On one hand, the *partial-order reduction* methods (POR) explore a reduced state space in a property-preserving way [10, 6]. On the other hand, symbolic techniques use functional representations of the state space to tackle the state-space explosion problem. Two different approaches to symbolic model-checking have been broadly considered: the BDD-based approach uses *binary decision diagrams* (BDDs) to concisely encode and compute state spaces [3], while the bounded model-checking (BMC) approach translates the original problem into a SAT problem. In their basic form, symbolic approaches tend to perform poorly on asynchronous models where concurrent interleavings are the main source of explosion, and explicit-state model-checkers with POR have been the preferred approach for such models.

This paper presents the Milestones model checker, which combines POR techniques and symbolic methods. Milestones defines a language for describing transition systems. CTL properties (as well as absence of deadlock) can be checked by combining BDD-based approach and POR [11]. LTL properties can be verified by combining POR either with the BDD-based approach or with the BMC approach [12]. In order to evaluate our approaches, Milestones can translate its

* This work is supported by project MoVES under the Interuniversity Attraction Poles Programme — Belgian State — Belgian Science Policy.

model into a Promela model [7] or into a NuSMV model [4]. In order to make the comparison as fair as possible, the resulting state machines are (almost) exactly the same as those generated by Milestones. In the case of NuSMV, the generated BDDs are the same as well. Milestones is available under the GNU General Public License at http://lvl.info.ucl.ac.be/Tools/Milestones.

2 Model Checking

When applying model-checking to verify a concurrent system, the size of the combined state space can grow exponentially in the number of processes, due to all the different interleavings among the executions of all the processes. Different approaches were developed to tackle this problem, among which the partial order methods (POR) and symbolic model checking.

The goal of partial-order reduction is to reduce the number of states explored by model-checking, by not exploring different equivalent interleavings of concurrent events. Naturally, these methods are best suited for strongly asynchronous programs. Interleavings which are required to be preserved may depend on the property to be checked. Intuitively, if two concurrent transitions α and β do not interfere with each other and do not affect the property f that one wants to verify, then it does not matter whether α is executed before or after β, and the exploration can be restricted to either of these two alternatives.

Symbolic model-checking, based on Binary Decision Diagrams (BDD), allows to reason on set of states rather than individual states. This technique made it possible to verify systems with a very large number of states [3]. However for large models, the size of the BDD structures themselves can become intractable. In contrast, bounded model-checking characterizes an error execution path of length k as a propositional formula, and searches for solutions to that formula with a SAT solver. BMC is limited by the need to fix the bound k but takes only polynomial space with respect to the model.

3 The Milestones Symbolic Model-Checker

Our tool, Milestones, is a symbolic model-checker which takes as input a model of a concurrent system annotated with temporal logic properties and produces as output the truth value of those properties. It also generates statistical data such as verification time, memory usage, BDD construction time, etc. Because collecting this information can significantly influence the verification time, the data generation can be switched off.

3.1 Modeling Language

Milestones defines a language for describing transition systems. The design of the language has been influenced by the NuSMV language [4] but supports synchronization by rendez-vous. Figure 1 shows parts of the Milestones model of the turntable system discussed in Section 4.

A model of a concurrent system declares a set of integer variables. Each variable is declared as follows `INTEGER` *name*`:` n `[:= expr]` where *name* is its name, n is the number of bits which are used to encode it, and `expr` is an

```
1  SYNC
2     ADD;
3     ERR;
4
5     exit;
6     REQremoveTrue;
7     REQremoveFalse;
8     ...
9  END //SYNC
10
11 VARIABLE
12       INTEGER r:1:=0;
13       INTEGER tr:1:=0;
14       ...
15 END //VARIABLES
16
17 LOCAL MC4
18    SYNC
19       exit;
20       REQremoveTrue;
21       REQremoveFalse;
22       ...
23    END //SYNC
24
25    ACTION
26       tau;
27    END //ACTION
28
29    VARIABLE
30       INTEGER pc:4:=0;
31       INTEGER mr:4;
32       ...
33    END //VARIABLES
34    CASE [0]
35    [exit] p3 == 0:
36       pc := 3;
37    [REQremoveTrue] p3 ==  1 & tr == 1:
38       pc := 1;
39    [REQremoveFalse] p3 ==  1 & tr == 0:
40       pc := 1;
41    END // CASE [0]
42
43    CASE [1]
44       ...
45    END //CASE [1]
46 END //LOCAL MC4
47
48 GLOBAL
49    VAR mc1: MC1;
50    VAR mc2: MC2[3];
51    VAR mc3: MC3;
52    VAR mc4: MC4;
53    VAR mc5: MC5;
54
55    CASE
56       [exit] true : p3 := 0;
57         p0 := 1;
58       ...
59    END //CASE
60 END//GLOBAL
61
62 LTL
63     F (p0 == 1);
64 END //LTL
```

Fig. 1. A Milestone model of a turntable system

,

expression which represents its initial value. The expression `expr` is optional. If it is not mentioned the initial value can be any values which are representable with `n` bits. A model declares a set of global variables (line 11), a set of shared actions (line 1) and a set of processes (line 17). A process p declares a set of local variables (line 29), a set of local actions (line 25) and the set of shared actions which p is synchronized on (line 18). Each process has a distinguished local program counter variable *pc* (line 30). For each value of *pc*, the behavior of a process is defined by means of a list of *action-labelled guarded commands* of the form `[`α`] c : u`, where α is an action, `c` is a condition on variables and `u` is an assignment updating some variables (line 36). *Shared* actions are used to define synchronization between the processes. A *shared* action occurs simultaneously in all the processes that share it, and only when all enable it. Properties can be expressed in CTL as well as LTL (line 62).

Milestones consists of a set command-line tools; it does not provide any graphical interface.

3.2 BDD-based CTL Verification

Milestones allows to check whether a model verifies a CTL property. The check can be performed with or without POR reduction (commands `checkCTLWithPOR` and `checkCTL`). Without POR, the classical *backward* CTL model checking algorithm of [3] is applied, as in NuSMV. With POR, Milestones uses the FwdUntilPOR approach which was first presented in [11]. In order to perform

the verification, a *forward* CTL model checking approach of Iwashita et al. [8] is combined with a symbolic POR forward exploration derived from Lerda et al.'s Improviso [9]. Contrary to the backward algorithm which does not apply POR methods, the forward approach is only applicable for a subset of CTL. Both methods can be combined together to check all the possible CTL formulæ.

3.3 BDD-based LTL Verification

Milestones is able to verify LTL properties, with or without POR reduction (commands `checkLTLWithPOR` and `checkLTL`), using the symbolic *tableau-based* LTL model checking algorithm of [5]. This method results in looking for fair executions in the product P of the model and a tableau-based encoding of the (negated) property. With POR, we construct P_r, a property-preserving partial-order reduction of P, using an adaptation of Lerda et al.'s ImProviso algorithm [9]. Finally, we check within P_r whether P contains a fair cycle using the forward traversal approach of Iwashita et al. [8].

3.4 SAT-based LTL Verification

LTL properties can also be checked by means of the *Bounded Model Checking* (BMC) approach, either with or without POR (command `checkLTLWithSBTP` and `checkLTLWithBMC`). Without POR, the algorithm of Biere et al. is executed [1]. With POR, we use the Stuttering Bounded Two-Phase algorithm (SBTP) first described in [12]. This algorithm merges a variant of Improviso [9] with the BMC procedure. In short, from a model and the negation of a property f, the BMC method constructs a propositional formula which represents a finite unfolding of the transition relation and $\neg f$. Our method proceeds in the same way, but instead of using the entire transition relation during the unfolding of the model, we only use a safe subset based on POR considerations.

Intuitively, SBTP alternately executes two phases. For each process of the model under verification, the first phase of SBTP unfolds some fixed number m of safe deterministic transitions. If less than m deterministic transitions are allowed, an idle transition which does not modify anything is performed instead. In the second phase, a full expansion occurs, even if there are safe deterministic transitions remaining. This avoids cycles of partial expansions, thus ensuring a property-preserving reduction. Because the generated propositional formula contains only few disjunctions, its satisfiability verification generates little backtracking, making it well-suited for modern DPLL-based SAT-solvers.

3.5 Exporting to Promela and NuSMV

Milestones can translate a model which is defined in its own language to NuSMV [4] (command `translateIntoNuSMV`) or to Promela, the language used by Spin [7] (command `translateIntoPromela`). We think it is important to compare Milestones to NuSMV and Spin, respectively the most prominent symbolic and explicit-state model-checkers.

The generated NuSMV model defines exactly the same state machine as the Milestones model. Because BDD variable ordering can considerably influence

the size of BDDs, and so the performance of the algorithm, a file which represents this order is generated[3]. This file can be used by NuSMV to construct its BDDs. Together these allow a close and fair comparison between Milestones and NuSMV.

The generated Promela model also defines almost the same state machine as the Milestones model, except for one more state which is necessary to correctly initialize the variables. We thus have good support for fair comparison between Spin and Milestones as well.

The accuracy of the translation was confirmed by comparing number of states and BDD nodes (in the NuSMV case), as well as by detailed comparison on small examples.

4 Evaluation

To evaluate the performance of Milestones, we used it to verify a turntable model which was first described in [2]. We also verified this model with NuSMV and Spin.

The turntable system consists of a round turntable, n drills and a testing device. The turntable transports products between the drills, the testing device and input and output positions. The drills bore holes in the products. After being drilled, the products are delivered to the tester, where the depth of the holes is measured, since it is possible that drilling went wrong. The turntable has $n + 3$ slots that each can hold a single product.

In [11], thirteen CTL properties have been checked on this model, For instance, the $p11$ property states that *each piece will be removed from the turntable after it is tested*. It is shown that a turntable with 40 drills can be checked in approximatively 40 seconds with the classical backward approach and in 4 seconds when the POR reduction is applied.

Six LTL properties, three of which are invalid, have been verified on the turntable model. For instance, the property T_3 states that *if in the future there is a piece which is not well drilled then the alarm will necessarily resonate*. Within a max time of 16 minutes, we are able to check T_3 on a turntable model composed of 61 drills with the Milestones model checker (with POR), 20 drills with the Spin model checker, 6 drills with the NuSMV model checker.

In essence, the length of the failure traces influences greatly the performance of the BMC algorithms. It turns out that the turntable model features failure traces that are too long for BMC approaches. On a variant of the producer-customer model which is composed of n producers and n consumers, SBTP achieves an improvement in comparison to the classical bounded model checking algorithm [12]. For $n = 3$ (resp. $n = 7$), the reachable state space of this system is approximatively equal to 3×10^6 states (resp. 10^{14}). When $n = 3$, the classical BMC approach verifies such a system in 11,679 seconds, and it takes more than 8 hours to verify a bigger model. By contrast, when $n = 7$, SBTP checks such a system in 77 seconds.

5 Conclusion

In this paper, we introduced the Milestones model checker. It merges POR methods and symbolic approaches to provide automatic verification of CTL and LTL

[3] For more details about the Milestones variable ordering, we refer the reader to [11]

properties on asynchronous models. The CTL properties are checked by means of a BDD-based approach, and the LTL properties can be verified either by the BDD-based technique or by the bounded model checking approach.

We show on a realistic-sized case study that our methods achieve an improvement in comparison to the classical algorithms. Although it is usually considered that symbolic model checking is inadequate for asynchronous systems, our results show that with appropriate optimization this approach might in fact be quite effective to tackle the state space explosion problem.

Although Milestones is able to check temporal properties, it needs to be extended by adding generation of counter-examples for failed properties. The heuristic used to determine safe transitions, i.e. transitions which can be exploited to perform POR, is quite simple. Instead of defining a set of processes, we could define a hierarchy of processes, and exploit it to discover more safe transitions.

References

1. Biere, A., Cimatti, A., Clarke, E.M., Strichman, O., Zhu, Y.: Bounded model checking. Advances in Computers 58, 118–149 (2003)
2. Bortnik, E.M., Trčka, N., Wijs, A., Luttik, B., van de Mortel-Fronczak, J.M., Baeten, J.C.M., Fokkink, W., Rooda, J.E.: Analyzing a χ model of a turntable system using spin, cadp and uppaal. J. Log. Algebr. Program. 65(2), 51–104 (2005)
3. Burch, J.R., Clarke, E.M., McMillan, K.L., Dill, D.L., Hwang, J.: Symbolic model checking: 10^{20} states and beyond. Information and Computation 98(2), 142–170 (1992)
4. Cimatti, A., Clarke, E., Giunchiglia, F., Roveri, M.: NuSMV: a new symbolic model verifier. In: Proc. of International Conference on Computer-Aided Verification (1999)
5. Clarke, E.M., Grumberg, O., Hamaguchi, K.: Another look at LTL model checking. Form. Methods Syst. Des. 10(1), 47–71 (1997)
6. Godefroid, P.: Partial-Order Methods for the Verification of Concurrent Systems – An Approach to the State-Explosion Problem, Lecture Notes in Computer Science, vol. 1032. Springer-Verlag (1996)
7. Holzmann, G.J.: The model checker SPIN. IEEE Transactions on Software Engineering 23(5) (1997)
8. Iwashita, H., Nakata, T., Hirose, F.: CTL model checking based on forward state traversal. In: ICCAD '96: Proceedings of the 1996 IEEE/ACM international conference on Computer-aided design. pp. 82–87. IEEE Computer Society, Washington, DC, USA (1996)
9. Lerda, F., Sinha, N., Theobald, M.: Symbolic model checking of software. In: Cook, B., Stoller, S., Visser, W. (eds.) Electronic Notes in Theoretical Computer Science. vol. 89. Elsevier (2003)
10. Peled, D.: Combining partial order reductions with on-the-fly model-checking. Formal Methods in System Design 8(1), 39–64 (1996)
11. Vander Meulen, J., Pecheur, C.: Efficient symbolic model checking for process algebras. In: 13th International Workshop on Formal Methods for Industrial Critical Systems (FMICS 2008). vol. 5596, pp. 69–84. LNCS (2008)
12. Vander Meulen, J., Pecheur, C.: Combining partial order reduction with bounded model checking. In: Communicating Process Architectures 2009 - WoTUG-32. Concurrent Systems Engineering Series, vol. 67, pp. 29 – 48. IOS Press (2009)