

Abstract Model Checking and Refinement of Temporal Logic in α SPIN

María del Mar Gallardo, Jesús Martínez, Pedro Merino, Ernesto Pimentel
Computer Science Department, University of Malaga
29071 Malaga , SPAIN
{gallardo,jmcruz, pedro, ernesto}@lcc.uma.es

Abstract

This paper gives an overview of the features offered by the tool α SPIN in order to perform abstract model checking of LTL formulas. Shortly, these features are: construction of over-approximated PROMELA models, checking satisfaction of universal formulas, checking refutation of existential formulas, and on-the-fly refinement of the model by means of a refinement of the temporal formula to be verified.¹

1 Introduction and Motivation

Model checking of complex systems is only applicable if we deal with the state explosion problem. Given M the (concrete) model of a system, and f the LTL temporal formula to be verified, the problem is that we usually have not enough memory to check whether the formula is satisfied for every trace in the model, i.e. $M \models \forall f$ (note that \forall is not being used as a logical connective, but it represents that f must be satisfied by *all traces* in the model M). The alternative provided by *abstract model checking* [1, 3, 4] consists of constructing an abstract version of the model, M^α , and an abstract version of the standard satisfaction relation for temporal logic formulas, \models_c^α , and then checking $M^\alpha \models_c^\alpha \forall f$. The model M^α is constructed as an over-approximation of M . Briefly, this means that given a concrete state s and a trace t produced by M , it is possible to find a state s^α and a trace t^α produced by M^α representing s and t , respectively. In contrast, the relation \models_c^α is defined to under-approximate the evaluation of f . A proposition p in a formula is under-approximated when given an abstract state s^α , p is defined to be true in s^α only when p is true for all concrete states in M that are abstracted by s^α . The same idea of under-approximation is applied to evaluate formulas over abstract traces. This way of constructing M^α and \models_c^α preserves satisfaction from the abstract to the concrete model: $M^\alpha \models_c^\alpha \forall f \Rightarrow M \models \forall f$.

¹Work supported by projects TIC2002-04309-C02-02 and TIC2001-2705-C03-02

We have developed a dual approach to abstract model checking that considers the over-approximated relation \models^α when checking the formula f against M^α [6]. In the same context as before, a proposition p is over-approximated when p is defined to be true in s^α if p is true at least for some concrete state s . This method naturally preserves the refutation of formulas from the abstract to the concrete model: $M^\alpha \not\models^\alpha \exists f \Rightarrow M \not\models \exists f$ (note that \exists is not part of the formula. It is employed to denote *some trace* satisfying f .)

Our tool α SPIN [5, 10] extends the model checker SPIN [9] to implement both abstraction methods. So, when standard SPIN is not able of verifying a formula, we can also offer users the possibility of choosing the abstraction method that best meets the property to be checked.

A major problem in both approaches for abstraction is the presence of spurious traces in the over-approximated version of M . The usual way to solve the problem is the refinement of M^α to obtain a more precise abstraction. Current methods usually work by re-constructing the model [2, 8]. Recently, in [7], Gallardo et. al propose the refinement of the formula (by combining under and over approximations) as an alternative to refine the abstract traces inspected during verification. α SPIN also supports this new feature.

2 Integrating the dual approaches in α SPIN

SPIN offers the users the possibility to express a property as a desirable behaviour ($M \models \forall f$) or as an error behaviour ($M \not\models \exists f$). SPIN works following the automata-theoretic approach to model checking: by default it translates the formula f into an automaton T_f and then check that the automata cannot recognize any trace in M . Checking a desirable behaviour is done by producing $T_{\neg f}$. In SPIN, both the automaton and the system M are represented in PROMELA. We take advantage of these features to implement abstraction by syntactic transformation of PROMELA, as shown in Fig. 1. When SPIN runs out-of-memory verifying a temporal property, α SPIN assists the user in choosing a function α to apply *data abstraction* to some variables in the joint

code including the model and the automaton, represented as $M \parallel T_f$, and then automatically performs the abstraction and calls SPIN to run a new verification. Abstraction by transformation is done replacing the instructions that manage the abstracted variables in order to implement the over-approximation of $M \parallel T_f$. The output $M^\alpha \parallel T_f^\alpha$ is given to SPIN, and the new verification checks $M^\alpha \not\models^\alpha \exists f$. The whole process is controlled with a graphical user interface (see [5, 10].)

Although abstracting the automata seems to work only for refutation, it actually implements both \models^α and \models_c^α . In [6] it is widely explained how \models^α and \models_c^α can be related in different ways, especially taking into account that they deal with negation in dual forms. One important result in [6] is that we can implement \models_c^α based on \models^α , using the following relation: $M^\alpha \models_c^\alpha \forall f \Leftrightarrow M^\alpha \not\models^\alpha \exists \neg f$.

3 Refinement of Temporal Logic

Our approach to remove spurious traces when using one of the previous abstraction methods is to refine the formula, in such a way that the new constraints in the formula conduct (on-the-fly) the refinement of the abstract model. The theory for this method is based on defining a new satisfaction relation, denoted as \Vdash , which generalizes both \models_c^α and \models^α (see [7]). In summary, \Vdash is defined to evaluate formulas that contains propositions to be considered as standard, under-approximated and over-approximated. Given f and g two temporal formulas, and f^α and g^α being their over-approximations, the refinement power is given by the following assertions:

$$\begin{aligned} (a) \quad M \models \forall g \text{ and } M^\alpha \Vdash \forall (g^\alpha \rightarrow f) &\Rightarrow M \models \forall f \\ (b) \quad M \not\models \exists g \text{ and } M^\alpha \Vdash \forall (f^\alpha \rightarrow g) &\Rightarrow M \not\models \exists f \end{aligned}$$

The practical application of these results consists of using a previously checked formula g to refine the abstract state space to be explored in order to know if $M \models \forall f$ or $M \not\models \exists f$. Fortunately, the implementation of this refinement method can again be done using only the over-approximation method, using that $M^\alpha \Vdash \forall (f^\alpha \rightarrow g)$ is equivalent to $M^\alpha \not\models^\alpha \exists (f \wedge \neg g)$.

Abstracting a model M typically introduces spurious non progress cycles that could be removed if we know that $M \models \forall \square \text{ progress}$. Here $g \equiv \square \text{ progress}$ works as the satisfied formula in the case (a), and progress is defined as the PROMELA predicate (!np-), np- being a predefined variable to identify whether the current state belongs to a non-progress cycle.

4 Conclusions

Current tools for abstract model checking are mainly focused to several of these features: symbolic model check-

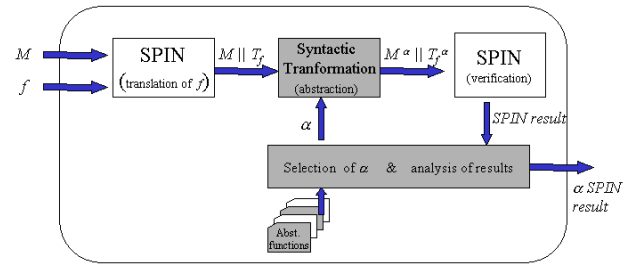


Figure 1. α SPIN architecture

ing, under-approximation of temporal formulas, refinement of the model to remove spurious traces. α SPIN is a complementary tool that covers: a) explicit on-the-fly model checking, b) under and over approximation of formulas and c) refinement with temporal formulas. Specific future work with α SPIN consists of integrating some of the methods for refinement of models, and combining their possibilities with the ones based on refining the formulas.

References

- [1] E. Clarke, O. Grumberg, and D. Long. Model checking and abstraction. *ACM Trans. on Programming Languages and Systems*, 16(5):1512–1245, 1994.
- [2] E.M. Clarke, O. Grumberg, S. Jha, Y. Lu, H. Veith. Counterexample-guided abstraction refinement. *CAV'00: Computer-Aided Verification*, LNCS-1855, pp. 154–169, 2000.
- [3] R. Cleveland, P. Iyer, D. Yankelevich. Optimality in Abstractions of Model Checking. *SAS'95: Static Analysis Symposium* LNCS-983, pp. 51–63, 1995.
- [4] D. Dams, R. Gerth, and O. Grumberg. Abstract interpretation of reactive systems. *ACM Trans. on Programming Languages and Systems*, 19(2):253–291, 1997.
- [5] M.M. Gallardo, J. Martínez, P. Merino, E. Pimentel: A Tool for Abstraction in Model Checking. *FMICS'02: 7th Int. Workshop on Formal Methods for Industrial Critical Systems*. ENTCS-66(2), 2002.
- [6] M.M. Gallardo, P. Merino, E. Pimentel: Comparing Under and Over-Approximations of LTL Properties for Model Checking. *WFLP'02: 11th Int. Workshop on Functional and (Constraint) Logic Programming*. ENTCS-76, 2002.
- [7] M.M. Gallardo, P. Merino, E. Pimentel: Refinement of LTL Formulas for Abstract Model Checking. *SAS'02: Static Analysis Symposium SAS '02* LNCS-2477, pp. 395–410, 2002.
- [8] R. Giacobazzi, E. Quintarelli. Incompleteness, Counterexamples and Refinement in Abstract Model-Checking. *SAS'01: Static Analysis Symposium SAS'01*, LNCS-2126, pp. 356–373, 2001.
- [9] G. J. Holzmann. The model checker SPIN. *IEEE Trans. on Software Engineering*, 23(5):279–295, 1997.
- [10] α SPIN project. <http://www.lcc.uma.es/gisum/fmse/tools>