Invariant-Free Clausal Temporal Resolution

J. Gaintzarain[1], **M. Hermo**[2], **P. Lucio**[2], **M. Navarro**[2] **and F. Orejas**[3]

[1] The University of the Basque Country, 48012-Bilbao, Spain.
[2] The University of the Basque Country, 20080-San Sebastián, Spain.
[3] Technical University of Catalonia, 08034-Barcelona, Spain.

Abstract: We provide an extended abstract of the paper with the same title and authors that is going to appear in Journal of Automated Reasoning (Online from December 2th, 2011).

Keywords: Temporal Logic, Resolution, Invariant-free, Clausal Normal Form.

Temporal logic plays a significant role in computer science, since it is an ideal tool for specifying object behaviour, cooperative protocols, reactive systems, digital circuits, concurrent programs and, in general, for reasoning about dynamic systems whose states change over time. Propositional Linear-time Temporal Logic (PLTL) is one of the most widely used temporal logics. This logic has, as the intended model for time, the standard model of natural numbers. Different contributions in the literature on temporal logic show its usefulness in computer science and other related areas. For a recent and extensive monograph on PLTL techniques and tools, we refer to [6], where the reader can find sample applications along with references to specific work that uses this temporal formalism to represent dynamic entities in a wide variety of fields. The minimal language for PLTL adds to classical propositional connectives two basic temporal connectives $\circ$ ("next") and $\mathcal{U}$ ("until") such that $\circ p$ is interpreted as "the next state makes p true" and $p\,\mathcal{U}\,q$ is interpreted as "p is true from now until q eventually becomes true". Many other useful temporal connectives can be defined as derived connectives, e.g. $\diamond$ ("eventually"), $\square$ ("always") and $\mathcal{R}$ ("release").

Automated reasoning for temporal logic is a quite recent trend. In temporal logics, as well as in the more general framework of modal logic, different proof methods are starting to be designed, implemented, compared, and improved. Automated reasoning for PLTL, and related logics, is mainly based on tableaux and resolution. In this paper, we deal with clausal resolution for PLTL. The earliest temporal resolution method [1] uses a non-clausal approach, hence a large number of rules are required for handling general formulas instead of clauses. There is also early work (e.g. [2, 4]) related to clausal resolution for (less expressive) sublogics of PLTL. The language in [2] includes no eventualities, whereas in [4] the authors consider the strictly less expressive sublanguage of PLTL defined by using only $\circ$ and $\diamond$ as temporal connectives. The early clausal method presented in [8] considers full PLTL and uses a clausal form similar to ours, but completeness is only achieved in absence of eventualities (i.e. formulas of the form $\diamond\varphi$ or $\varphi\,\mathcal{U}\,\psi$). More recently, a fruitful trend of clausal temporal resolution methods, starting with the seminal paper of M. Fisher [5], achieves completeness for full PLTL by means of a specialized *temporal resolution* rule that needs to generate an invariant formula from a set of clauses that behaves as a loop. The methods and techniques developed in such an approach have been successfully adapted to Computation Tree Logic (CTL) (see [3]), but invariant handling seems to be a handicap for further extension to more general branching temporal logics such as Full Computation Tree Logic (CTL*). In this paper, we introduce a new clausal resolution method that is sound and complete for full PLTL. Our method is based on the dual methods of tableaux and sequents for PLTL presented in [7]. On this basis we are able to perform clausal resolution in the presence of eventualities avoiding the requirement of invariant generation. We define a notion of *clausal normal form* and prove that every PLTL-formula can be translated into an equisatisfiable set of clauses. Our resolution mechanism explicitly simulates the transition from one world to the next one. Inside each world, we apply two kinds of rules: (1) the resolution and subsumption rules and (2) the fixpoint rules that split a clause with an eventuality atom into a finite number of new clauses. We prove that the method is sound and complete. In fact, it finishes for any set of clauses deciding its (un)satisfiability, hence it gives rise to a new decision procedure for PLTL. We also compare our approach with the methods in [4, 1, 8, 5].

Acknowledgements: This work has been partially supported by the Spanish Project TIN2007-66523 and the Basque Project LoRea GIU07/35.

Bibliography

[1] Martín Abadi and Zohar Manna. Nonclausal temporal deduction. In Rohit Parikh, editor, *Logic of Programs*, volume 193 of *Lect. Notes in Computer Sci.*, pages 1–15. Springer, 1985.

[2] Marianne Baudinet. Temporal logic programming is complete and expressive. In *Proceedings, Sixteenth Annual ACM Symposium on Principles of Programming Languages (POPL), Austin, Texas*, pages 267–280, 1989.

[3] Alexander Bolotov and Michael Fisher. A clausal resolution method for CTL branching-time temporal logic. *Journal of Experimental & Theo. Artif. Intell.*, 11(1):77–93, 1999.

[4] Ana R. Cavalli and Luis Fariñas del Cerro. A decision method for linear temporal logic. In Robert E. Shostak, editor, *7th International Conference on Automated Deduction, Napa, California, USA, May 14-16, 1984, Proceedings*, volume 170 of *Lecture Notes in Computer Science*, pages 113–127. Springer, 1984.

[5] M. Fisher. A resolution method for temporal logic. In John Mylopoulos and Raymond Reiter, editors, *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI), Sydney, Australia.*, pages 99–104, 1991.

[6] Michael Fisher. *An Introduction to Practical Formal Methods Using Temporal Logic*. John Wiley & Sons, Ltd, June, 2011.

[7] Jose Gaintzarain, Montserrat Hermo, Paqui Lucio, Marisa Navarro, and Fernando Orejas. Dual systems of tableaux and sequents for PLTL. *Journal of Logic and Algebraic Programming*, 78(8):701–722, 2009.

[8] G. Venkatesh. A decision method for temporal logic based on resolution. In S. N. Maheshwari, editor, *Foundations of Software Technology and Theoretical Computer Science, Fifth Conference, New Delhi, India, December 16-18, 1985, Proceedings*, volume 206 of *Lecture Notes in Computer Science*, pages 272–289. Springer, 1985.

Testing Temporal Logic on Infinite Java Traces

Damián Adalid, Alberto Salmerón, María del Mar Gallardo and Pedro Merino[1]*

[1](damian,salmeron,gallardo,pedro)@lcc.uma.es
Dpto. de Lenguajes y Ciencias de la Computación
University of Málaga
Spain

Abstract: This paper summarizes an approach for testing reactive and concurrent Java programs which combines model checking and runtime monitoring. We use a model checker for two purposes. On the one hand, it analyzes multiple program executions by generating test input parameters. On the other hand, it checks each program execution against a linear temporal logic (LTL) property. The paper presents two methods to abstract the Java states that allow efficient testing of LTL. One of this methods supports the detection of cycles to test LTL on potentially infinite Java execution traces. Runtime monitoring is used to generate the Java execution traces to be considered as input of the model checker. Our current implementation in the tool TJT uses Spin as the model checker and the Java Debug Interface (JDI) for runtime monitoring. TJT is presented as a plug-in for Eclipse and it has been successfully applied to complex public Java programs.

Keywords: testing, model checking, runtime monitoring

1 Introduction

This paper is a summary of [ASGM12] where we present a new method to convert a Java execution trace into a sequence of states that can be analyzed by the model checker Spin [Hol03]. As far as Spin implements the analysis of LTL formulas by translation to Büchi automata, we can check the formulas on Java programs with potential cycles. The Spin stuttering mechanism to deal with finite execution traces allows us to analyze any kind of program without redefining the semantics of LTL.

Our conversion of Java traces into Spin oriented traces is based on two efficient abstraction methods of the full state of the program. The so-called *counter projection* abstracts the Java state by preserving the variables in the LTL formula and adding a counter to make each state unique. As long as we do not keep all the information, the counter projection is very efficient at the price of being useful only for a limited subset of LTL. The *hash projection* abstracts each Java state with the variables in the formula plus a hash of the whole state. The way of constructing the hash makes negligible the probability of conflict for two different states, so we can trust in the Spin algorithm to check LTL based on cycle detection.

* Work partially supported by projects P07-TIC-03131 and WiTLE2.