Trio2Promela: a Model Checker for Temporal Metric Specifications

Domenico Bianculli[1] Angelo Morzenti[2] Matteo Pradella[3] Pierluigi San Pietro[2] Paola Spoletini[2]

[1]Faculty of Informatics
University of Lugano
domenico.bianculli@lu.unisi.ch

[2]Dipartimento di Elettronica e Informazione
Politecnico di Milano
[3]CNR IEIIT-MI,
{morzenti, pradella, sanpietr, spoleti}@elet.polimi.it

Abstract

We present Trio2Promela, a tool for model checking metric temporal logic specifications written in the TRIO language. Our approach is based on the translation of formulae into Promela programs for the model checker Spin, guided by equivalence between temporal logic and alternating Büchi automata. Trio2Promela may be used also to check satisfiability of temporal logic specifications (a distinguishing difference with other model checking tools).

1. Introduction

TRIO is a first order, linear-time temporal logic with both future and past operators and a quantitative metric on time, which has been extensively applied to the specification, validation and verification of critical, real-time systems [7]. Over the years a variety of methods and tools have been defined to support typical validation and verification activities in TRIO, as described in [6] and [2].

In the present paper we report on a new approach to verification, by means of a fully automatic tool: it consists of defining a decidable fragment of the logic that includes a suitable subset of its original operators, upon which applying methods and algorithms for deciding satisfiability. The tool, called Trio2Promela, is built on top of a well-known model checker such as Spin [4] to perform proof of properties and simulation. The approach and background theory upon which Trio2Promela is constructed was originally presented in [8, 9]; the tool, together with examples and results of several experiments, is available at `http://www.elet.polimi.it/upload/sanpietr/Trio2Promela.zip`.

Trio2Promela supports property proof by model checking and satisfiability checking of generic TRIO formulae. Hence, Trio2Promela can be used to accomplish a traditional approach to model-checking, by translating the property to be checked from TRIO into Promela, and by combining the resulting code with a pure-Promela model to perform verification. Satisfiability checking may also be used to perform property verification in TRIO, by checking the validity of a TRIO formula of the kind *specification→property*, where the premise *specification* describes the features that are *assumed to hold* for (any possible implementation of) the analyzed system, and *property* is another formula describing the conjecture that we want to prove to be implied by the premise. Therefore, in this approach what we call *specification* has a role similar to the one played by the so-called *model* in the usual model checking scenery (e.g., a Promela program in Spin) while what we call *property* in the above implication is usually called *specification* (or, sometimes, *user requirement*) and takes the form of a formula in temporal logic (e.g., an LTL formula in Spin).

2. Trio2Promela

Trio2Promela translates a TRIO specification, i.e., a complex TRIO formula, into Promela code. The translation is based on a correspondence between TRIO and *Alternating Modulo Counting Automata* (AMCA) described in [8]. The main idea is based on the well-known correspondence between Linear Temporal Logic and Alternating Automata (see for instance [5]), together with *counters*, associated with states of the AMCA, that are used to express TRIO's metric temporal operators in a natural and concise manner.

An AMCA is *directly* translated into a Promela program: every state of the automaton will correspond to a single type of process (i.e., a Promela `proctype`), to be instantiated when needed. An or-combination of states $s1 \vee s2$ in the transition function corresponds to a nondeterministic choice (`if ::s1; ::s2; fi`), while an and-combination $s1 \wedge s2$ corresponds to the starting of two new

Promela process instances, having type `s1` and `s2`, respectively. Hence, the produced code consists of a network of processes, each corresponding to a temporal subformula of the original specification. Each Promela process receives as input a chronological sequence of values for the alphabet of the associated TRIO formula, and then it returns its computed truth value to the network. When the process representing the whole TRIO formula being analyzed returns *False*, every process in the network is stopped, and the analysis terminates.

As TRIO past operators are concerned, we take advantage of the fact that time is unlimited only towards the future (there is a *start* time instant) to treat past operators differently, using a technique illustrated in our paper [9] that stores a bounded amount of information derived from the past portion of the sequence of input values.

3. Assessment

Detailed experimental results on the application of Trio2Promela to various case studies are reported in [1]; some will be illustrated also in the tool demonstration. Our approach can be naturally compared with recent works (such as those on LTL2BA [3] and Wring [10]) that aim at the translation of LTL properties into Büchi automata and then Promela programs). In all examples that those tools can manage (successfully completing the translation into a Büchi automaton) our tool also translates the corresponding TRIO formula, and carries out the verification with a performance that is comparable or superior. Hence, current experimental evidence shows that Trio2Promela can be used also for LTL model checking.

However, our approach differs substantially from others (such as LTL2BA and Wring): these translate LTL formulae (which in the first place are less compact than TRIO formulae, as they cannot include integer values representing time constants, and therefore must use long chains of nested Next operators), into Spin *never claims* or Büchi automata whose size can grow to become unmanageable even for relatively simple specifications. When the desired property is fairly complex or contains several bounded temporal statements, which is typical of real-time systems, the traditional approach of generating a so-called *never claim* for Spin becomes unfeasible. For instance, the LTL version of the statement "every occurrence of event A must be followed within 20 time units by an occurrence of event B", which in TRIO can be modeled as simply as $AlwF(A \rightarrow WithinF(B, 20))$ cannot be translated by LTL2BA or Wring (the system was stopped after waiting for 24 hours), even if it actually corresponds (if one time unit is taken to correspond to one transition) to a Büchi automaton with only 21 states. Trio2Promela is able to translate the above statement into a short Promela program almost instantaneously.

The size of the resulting Promela code is always *linear in the size of the AMCA*, and therefore also *in the size of the original TRIO specification* (which may be substantially smaller than an equivalent LTL formula). This is obtained by avoiding, at least at translation time, the state explosion problem, thanks to the fact that we generate a Promela program that will simulate (at verification time) the alternating automaton. The alternation removal is then left to the model checker, allowing the verification of many temporal logic formulae which could not be translated into Promela by means of other techniques. Instead, if the alternation is removed during the translation phase, as all other techniques we know of do, then there are many cases where a translator cannot even build the resulting automaton (where all states are explicitly enumerated). This typically happens when specifications are very large, or use metric temporal operators, or mix past and future temporal operators.

References

[1] D. Bianculli, P. Spoletini, A. Morzenti, M. Pradella, and P. San Pietro. Model checking temporal metric specifications with Trio2Promela. In *FSEN'07, Proc. of the International Symposium on Fundamentals of Software Engineering*, 2007.

[2] A. Gargantini and A. Morzenti. Automated deductive requirements analysis of critical systems. *ACM Trans. Softw. Eng. Methodol.*, 10(3):255–307, 2001.

[3] P. Gastin and D. Oddoux. Fast LTL to Büchi automata translation. In *CAV '01: Proc. of the 13th International Conference on Computer Aided Verification*, volume 2102 of *LNCS*, pages 53–65, 2001.

[4] G. J. Holzmann. The model checker SPIN. *IEEE Trans. Softw. Eng.*, 23(5):279–295, 1997.

[5] O. Kupferman and M. Vardi. Weak alternating automata are not that weak. In *ISTCS'97: Proc. of the 5th Israel Symposium on Theory of Computing and Systems*, pages 147–158. IEEE Computer Society Press, 1997.

[6] D. Mandrioli, S. Morasca, and A. Morzenti. Generating test cases for real-time systems from logic specifications. *ACM Trans. Comput. Syst.*, 13(4):365–398, 1995.

[7] A. Morzenti, D. Mandrioli, and C. Ghezzi. A model parametric real-time logic. *ACM Trans. Program. Lang. Syst.*, 14(4):521–573, 1992.

[8] A. Morzenti, M. Pradella, P. San Pietro, and P. Spoletini. Model checking TRIO specifications in Spin. In *FME 2003: Proc. of the International Symposium of Formal Methods Europe*, volume 2805 of *LNCS*, pages 542–561, 2003.

[9] M. Pradella, P. San Pietro, P. Spoletini, and A. Morzenti. Practical model checking of LTL with past. In *ATVA03: 1st Workshop on Automated Technology for Verification and Analysis*, 2003.

[10] F. Somenzi and R. Bloem. Efficient Büchi automata from LTL formulae. In *CAV '00: Proc. of the 12th International Conference on Computer Aided Verification*, volume 1855 of *LNCS*, pages 248–263, 2000.