TIMED AUTOMATA MODEL AND COUPLING TO A PROTOTYPE BOUNDED MODEL CHECKER

A PREPRINT

Ehsan A. Elwan[*]
Department of Computer Science
Paul Sabatier University
118, route de Narbonne
ehsan.alidelbi-elwan@univ-tlse3.fr

Monpara Radhika
Department of Computer Science
Paul Sabatier University
118, route de Narbonne
monpararadhika96@gmail.com

Mukhtar I. Khalil
Department of Computer Science
Paul Sabatier University
118, route de Narbonne
khalildonteam@gmail.com

April 6, 2019

ABSTRACT

Timed automata (TA) were introduced as a formal notation to model and analyze the behaviour of real-time systems, network protocols, etc. Bounded Model Checking (BMC) has proven to be a very efficient technique for the verification of these systems and protocols.It reduces model checking to propositional satisfiability (SAT). Many computer scientists, software engineers etc. use these verification tools for many purposes, therefore it is important to explore new ideas and methods to make them more efficient. The method ensures safety and liveness by checking whether a given set of states is reachable and detecting loops in a system's state transition graph. In this paper, we explicate a timed automata network model and ensure the coupling on the server-side with an existing prototype bounded model checker, capable of computing feasible execution traces. The prototype implementation relies on an MILP solver (Gurobi) and SAT solver (MathSAT). Bounded model checking for networks of timed automata is also explained.

***K*eywords** Timed automata · Bounded model checking

1 Introduction

Timed automata [1] were introduced as a formal notation to model the behavior of real-time systems [2], called an execution trace. The behavior of the system is a set of such execution sequences. Since a set of sequences is a formal language, this leads naturally to the use of automata for the specification and verification of systems. When the systems are finite-state, as many are, we can use finite automata, leading to effective constructions and decision procedures for automatically manipulating and analyzing system behavior.

Bounded model checking [3] is a new type of model checking technique that reduces model checking to a SAT problem. The method ensures safety and liveness by checking whether a given set of states is reachable and detecting loops in a system's state transition graph. Favorable experimental results with bounded model checking have been obtained for safety properties. A simple, yet very important type of safety property is an invariant, a property that must hold in all reachable states. Obviously, if a sequence of states can be found that begins at an initial state and ends in a state where

[*]Use footnote for providing further information about author (webpage, alternative address)—*not* for acknowledging funding agencies.

the supposed invariant is false, that property is not an invariant. It turns out that such searches for counter examples can be done with remarkable efficiency with bounded model checking, even on designs that would be difficult for other model checking techniques such as BDD-based model checking.

There are several advantages of bounded model checking. SAT tools like Prover and MathSAT (for SMT Solver) do not require exponential space and large designs can be checked very fast, since the state space is searched in an arbitrary order. This technique uses depth first search procedures, unlike BDD based model checking, which usually operates in breadth first search consuming much more memory. Furthermore, this procedure is able to find paths of minimal length, which helps the user understand the examples that are generated. SAT tools [4] generally need far less by hand manipulation than BDDs. Generally, this technique is able to find bugs and sometimes to determine correctness, in situations where other techniques fail completely.

In this paper, we explicate a timed automata network model and ensure the coupling on the server-side with an existing prototype bounded model checker. The practical example of the front end is the implementation of a python-based diagram editor of a simple version of timed automata network model. After the coupling using JSON object, he execution traces are displayed in form of sequence diagrams in UML (Unified Modeling Language). We also study the reduction of bounded reachability analysis of timed automata to a mixed integer linear programming problem. In theoretical Computer science, a reduction is an algorithm for transforming one problem into another. "A is reducible to B" means that if an algorithm exists to solve the problem B, that algorithm can also be used as a subroutine to solve the problem of A. It also means that means if we find some problem A as difficult to solve, then we can reduce it to an easier problem B.

This paper is organized as follows. In the following section, we analyze timed automata in general and its semantics. In Section 3, we explain the basic idea of bounded model checking. Section 4 demonstrates the coupling of timed automata with a bounded model checker. Experimental results (execution traces) of one run are displayed and analyzed. Finally in Section 5, we conclude and discuss some directions for future research.

2 Timed automata

Timed automata are finite automata characterized by real valued clocks [5]. During a run of a timed automaton, clock values increase all with the same speed. Along the transitions of the automaton, we can compare the clock values with integers (for e.g. time t > 2000ms). Timed automata can be used to analyze the timing behavior of the computer system. MILP [6] is a linear programming problem (e.g the ones studied in Advanced Optimization, with line equations, constraints etc.) whose solution must be an integer. Revisiting (Consider again the) bounded reachability Analysis of timed automata based on MILP [7].

Q. What is reachability analysis?

It is a solution to a problem of reachability. We can use it to determine which global states can be reached by a system which consists of a certain number of local entities that communicate by exchange of messages. Reachability analysis considers the possible behavior of the distributed systems by analyzing all possible sequences of state transitions of the entities and the corresponding global states reached [8]. The result of the reachability analysis is a global state transition graph also called reachability graph which shows all global states of the system that are reachable from the initial global state, and all possible sequences of interactions performed by the entities. It is bounded, so that we can explore the graph completely.

Referring to the reduction theory, if the time required to transform the first problem to the second problem, and the number of times the subroutine is called is polynomial, then we can say that the reduction algorithm (to reduce A to B) is polynomial time reducible. So, even though MILP is NP-complete [9], we need to find an efficient algorithm for MILP in order to solve the reachability problem of timed automata. A MILP problem is an LP problem in which some of the variables are constrained to be integers. Although MILP is NP-complete, many of the solvers are capable of solving very large problems arising in practice and their performance has vastly improved during the past decades.

SAT solvers [10] are automatic and efficient. As a result, they are frequently used as the engine behind verification applications. However, systems are usually designed and modelled at a higher level than Boolean level and the translation to Boolean logic can be expensive.Therefore, SMT's goal is to create verification engines that can solve at a higher level of abstraction, while still retaining the speed and automata of Boolean engines. We have to reduce the search space and improve the solver's performance.

Regardless of the fact that timed automata give rise to infinite state spaces due to the dense domain of time, both reachability and model checking of various logics [11] are decidable based on finite symbolic representations of the state space. They rely on symbolic representation of state sets. BMC on the other hand is a successful method for analyzing models that yield very large state spaces. It relies on encoding the next state relationship as a logic formula and on instantiating this formula a bounded number of times to encode all possible runs of depth equal to the bound. Then, a valid run corresponds to an assignment of the variables that satisfies the formula. The verification of properties on runs is hence reduced to SAT for the logical formula, encoding possible runs.

During a run of a timed automaton, clock values increase all with the same speed. Along the transitions of the automaton, clock values can be compared to integers. These comparisons form guards that may enable or disable transitions and by doing so constrain the possible behaviours of the automaton. Further, clocks can be reset. Timed automata are a sub-class of a type hybrid automata [12]. Methods for checking both safety and liveness properties have been developed and intensively studied over the last 20 years. It has been shown that the state reachability problem for timed automata is decidable, which makes this an interesting sub-class of hybrid automata. Extensions have been extensively studied, among them stopwatches, real-time tasks, cost functions, and timed games. There exists a variety of tools to input and analyze timed automata and extensions, including the model checker UPPAAL [13] and the schedulable analyser TIMES. These tools are becoming more and more mature, but are still all academic research tools. Below is an example of a simple timed automaton:

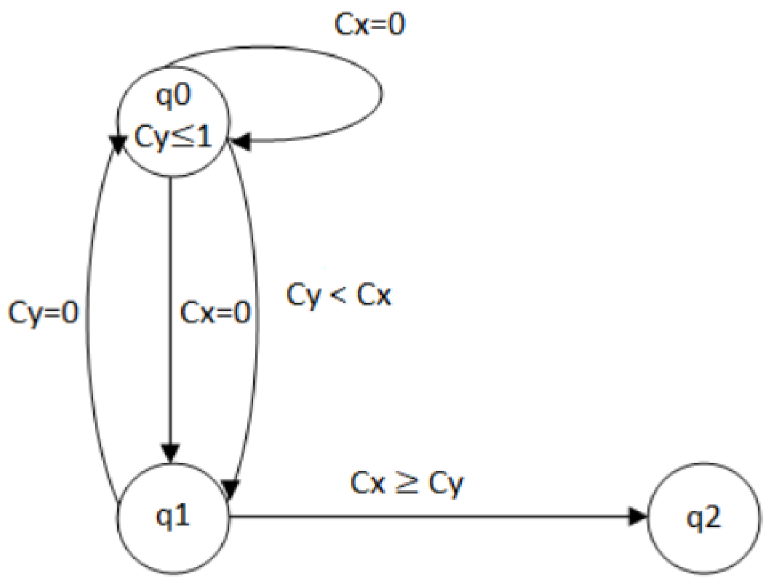

Figure 1: Example of a simple timed automaton

This automaton has two clock variables (Cx,Cy) and three states q_1, q_2, q_3. More formally, a timed automaton is a tuple A = (Q, Σ, C, E, q0) that consists of the following components:

• Q is a finite set. The elements of Q are called the states of A.

• Σ is a finite set called the alphabet or actions of A.

• C is a finite set called the clocks of A.

• $E \subseteq Q \times \Sigma \times B(C) \times P(C) \times Q$ is a set of edges, called transitions of A, where

• B(C) is the set of Boolean clock constraints involving clocks from C, and

• P(C) is the power set of C.

• q_0 is an element of Q, called the initial state.

An edge (q, a, g, r, q') from E is a transition from state q to q' with action a, guard g and clock resets r.

In addition to inequalities, some solvers may accept a number of additional constraint types, such as indicator constraints which have the form b where b is a binary variable and C is a linear inequality that has to be satisfied by the solution only if b has the value 1. This is the only form of non-linear constraint used in our formulation of the reachability problem.

3 Bounded model checking

Model checking [14] is a verification technique with three fundamental features. First, it is automatic; It does not rely on complicated interaction with the user for incremental property proving. If a property does not hold, the model checker generates a counterexample trace automatically. Second, the systems being checked are assumed to be finite. Typical examples of finite systems, for which model checking has successfully been applied, are digital sequential circuits and communication protocols. Temporal logic is used for specifying the system properties. Thus, model checking can be summarized as an algorithmic technique for checking temporal properties of finite systems.

BMC relies on encoding the next-state relationship as a logical formula a bounded number of times to encode all possible runs of depth equal to the bound. Then, a valid run corresponds to an assignment of the variables that satisfies the formula. The verification of properties on runs is hence reduced to (the SAT problem for the logic formula), encoding possible runs [15]. BMC was initially introduced for discrete state-transition systems and formulas are expressed in plain propositional logic. State space is a set of all possible states of the system. Each coordinate is a state variable, and the values of all the state variables completely describes the state of the system. In other words, each point in the state space corresponds to a different state of the system.

The original motivation of bounded model checking was to leverage the success of SAT [16] in solving Boolean formulas to model checking. During the last few years there has been a tremendous increase in reasoning power of SAT solvers. They can now handle instances with hundreds of thousands of variables and millions of clauses. Symbolic model checkers with BDDs, on the other hand, can check systems with no more than a few hundred latches. Though clearly the number of latches and the number of variables cannot be compared directly, it seemed plausible that solving model checking with SAT could benefit the former. A similar approach has been taken in tackling the planning problem in Artificial Intelligence. As in BMC, the search for a plan is restricted to paths with some predetermined bound. The possible plans in a given bound are described by a SAT instance, which is polynomial in the original planning problem and the bound. Compared to model checking, deterministic planning is only concerned with simple safety properties: whether and how the goal state can be reached [17]. In model checking we want to check liveness properties and nested temporal properties as well.

To formulate bounded reachability in MILP (Gurobi) [18], consider a system of timed automata. Reachable states and transitions of the system are encoded as variables and constraints of a MILP problem. Several options are available for the encoding and the formulation concerns the states of the system that can be reached through a sequence of transitions of bounded length. To simplify the denitions, one has to consider rst that there is a total order between the states and between the transitions, although this constraint will be relaxed later on.

The dierence between an SMT-based bounded model checker [19]/reachability analyzer and one based on MILP is that the latter may integrate an optimization objective. The objective has the form of a linear expression on model variables. It may prove useful for selecting a system run out of the set of feasible ones based on minimizing/maximizing various criteria. For example, for model debugging it is often convenient to obtain the shortest run that leads to the searched state, i.e. the run that contains the smallest number of (non-skip) discrete transitions.

3.1 LTL Semantics in BMC

A Kripke structure is a variation of the transition system, originally proposed by Saul Kripke, used in model checking to represent the behavior of a system. It is basically a graph whose nodes represent the reachable states of the system and whose edges represent state transitions.

A Kripke structure is a tuple M = (S, I , T , L) with a finite set of states S, the set of initial states I $\subseteq$ S, a transition relation between states T $\subseteq$ S x S, and the labeling of the states L: S $\rightarrow$ P(A) with atomic propositions A. We use Kripke structures as models in order to give the semantics of the logic.

They are characterized by Boolean encoding of a state. Each state has a successor state. The next step is to define the Semantics of LTL [20] with a Kripke structure M, where LTL formula f is valid along path π (π $\models$ f).]

An LTL formula f is universally valid in a Kripke structure M (in symbols M $\models$ Af) iff π $\models$ f for all paths π in M with $\pi(0) \in$ I. An LTL formula f is existentially valid in a Kripke structure M (in symbols M $\models$ Ef) iff there exists a path π in M with π $\models$ f and $\pi(0) \in$ I.

We consider searching for a counter example for the existential model checking problem. However, we are considering a bounded sequence so, we can derive bounded semantics with and without a loop. After that, we translate LTL into propositional formulae and determine the bound [21].

3.2 BMC activity diagram

Fig. 2 below shows the activity diagram of bounded model checking:

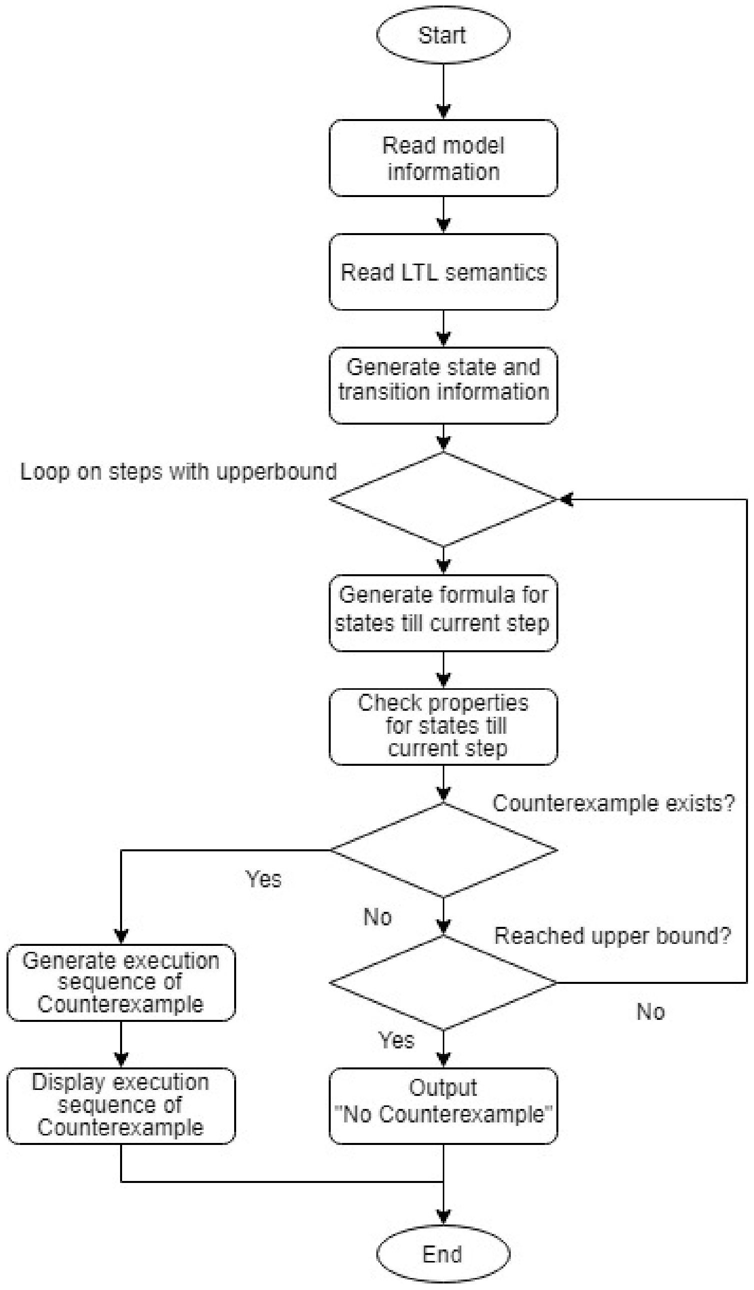
Start
Read model information
Read LTL semantics
Generate state and transition information
Loop on steps with upperbound
Generate formula for states till current step
Check properties for states till current step
Counterexample exists?
Yes
No
Reached upper bound?
No
Yes
Generate execution sequence of Counterexample
Display execution sequence of Counterexample
Output "No Counterexample"
End

Figure 2: BMC Activity Diagram

To promote practical use of model checking techniques in on-site software development, we realized that graphicalized modeling languages (e.g., representatively, UML) are more easily acceptable compared to model-checker-specific program-like formal description languages. An activity diagram is mostly much more readable and understandable [22]. This step by step process ensures that every counterexample is checked within a bound. At first the model information and LTL semantics are read. The state and transition information are then generated. The formulae for each step till current step is generated. The properties for all states till the current state are also checked. As long as a counter example exists, its execution sequence is generated and displayed. This activity ends when there is no counterexample to be checked.

4 Timed automata bounded model checking

Given a BMC problem [23] for a timed automaton, an LTL formula [24] with linear arithmetic constraints, and a bound of the length of counterexamples to be searched for, we describe a sound and complete reduction to the satisfiability problem [25] of Boolean constraint formulas. The encoding of the transition relations of the given automaton follows the now-standard approach. We we use Buchi automata for this encoding. This simplifies substantially the notations and the proofs, but a direct translation can sometimes be more succinct in the number of variables needed. We use the common notions for finite automata over finite and infinite words, and we assume as given a theory A of linear arithmetic constraints with a satisfiability solver.

4.1 System verification

A timed automaton S, can easily be described in terms of a program with linear arithmetic constraints over state variables *V= at*, x$_1, ..., x_n$, where *at* is interpreted over the set *L* of locations and clock variables $x_1, ..., x_n \subseteq$ *Cl* are interpreted over *IR*$_0 +$. We also define the state transition steps, delay steps and transition relations.]

Definition 4.1.1 [Encoding of C-Programs] The encoding [[M]]k of the *k*th unfolding of a C-program *M = (I; T)* in Prg(C(x$_1, ..., x_n$)) is given by a Boolean constraint formula [[M]]k.

After that, We have the encoding of Buchi formula. The system is checked taking acount of the Soundness theorem, Completeness for Finite state systems theorem and Completeness for Timed Automata theorem.

Theorem 4.1.2 (BMC for networks of timed automata) Consider two timed automata with disjoint set of clocks, A$_i =$ $(L_i, l_i, \Sigma_i, Cl_i, E_i, Inv_i)$, for i = 1, 2. Let *M' = (I', T')* be the program corresponding to the network of automata A1 $\|$ $A2$, and *M = (I, T)* be the program encoding the product automaton A1 x A2 as described above. Then for a k $\subseteq IN$, the kth unfolding of *M'* and *M* are equisatisfiable, that is $[[M']]_k \equiv [[M]]_k$.
Proofsketch: By induction over k we show that [[M]]$_k$ and $[[M']]_k$ are equisatisfiable.

5 Conclusion

In this paper, we have explicated a timed automata network model and ensured the coupling on the server-side with an existing prototype BMC, capable of computing feasible execution traces. BMC for networks of timed automata has been also explained. The approach is symbolic, and allows for (Bounded) Model Checking of LTL formulas. Furthermore, the MILP solver can be rather efficient in terms of run-times,but it is sometimes sensitive to rounding errors. Our approach is intended to be complementary rather than alternative to the current ones.

In the future, we plan to extend and improve our work along the following directions. First, we want to improve and test new kinds of encodings. We will also find new ideas to decrease the sensitivity of the MILP solver to rounding errors. Then, we will investigate the customization of MILP and MathSAT for encoded timed automata. Finally, we want to perform an extensive experimental evaluation, also on synchronous domains, to identify the bottlenecks and the strengths of the approach.

6 Acknowledgement

We would like to thank the anonymous referees for their helpful comments for improving this paper. Iulian Ober provided many useful inputs.

References

[1] Rajeev Alur and David L Dill. A theory of timed automata. *Theoretical computer science*, 126(2):183–235, 1994.

[2] Farn Wang. Efficient data structure for fully symbolic verification of real-time software systems. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 157–171. Springer, 2000.

[3] Armin Biere, Alessandro Cimatti, Edmund Clarke, and Yunshan Zhu. Symbolic model checking without bdds. In *International conference on tools and algorithms for the construction and analysis of systems*, pages 193–207. Springer, 1999.

[4] Parosh Aziz Abdulla, Per Bjesse, and Niklas Eén. Symbolic reachability analysis based on sat-solvers. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 411–425. Springer, 2000.

[5] Kim G Larsen, Justin Pearson, Carsten Weise, and Wang Yi. Clock difference diagrams. *Nord. J. Comput.*, 6(3):271–298, 1999.

[6] Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Automated configuration of mixed integer programming solvers. In *International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming*, pages 186–202. Springer, 2010.

[7] Iulian Ober. Revisiting bounded reachability analysis of timed automata based on milp. In *International Workshop on Formal Methods for Industrial Critical Systems*, pages 269–283. Springer, 2018.

[8] Jerry R Burch, Edmund M Clarke, Kenneth L McMillan, David L Dill, and Lain-Jinn Hwang. Symbolic model checking: 1020 states and beyond. *Information and computation*, 98(2):142–170, 1992.

[9] Jeffrey T Linderoth and Andrea Lodi. Milp software. *Wiley encyclopedia of operations research and management science*, 2010.

[10] Janusz Malinowski and Peter Niebert. Sat based bounded model checking with partial order semantics for timed automata. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 405–419. Springer, 2010.

[11] Wolfgang Thomas. Languages, automata, and logic. In *Handbook of formal languages*, pages 389–455. Springer, 1997.

[12] Karine Altisen and Stavros Tripakis. Implementation of timed automata: An issue of semantics or modeling? In *International Conference on Formal Modeling and Analysis of Timed Systems*, pages 273–288. Springer, 2005.

[13] Gerd Behrmann, Alexandre David, Kim Guldstrand Larsen, Paul Pettersson, and Wang Yi. Developing uppaal over 15 years. *Software: Practice and Experience*, 41(2):133–142, 2011.

[14] Stephan Merz. Model checking: A tutorial overview. In *Summer School on Modeling and Verification of Parallel Processes*, pages 3–38. Springer, 2000.

[15] Alessandro Cimatti, Alberto Griggio, Bastiaan Joost Schaafsma, and Roberto Sebastiani. The mathsat5 smt solver. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 93–107. Springer, 2013.

[16] Kenneth L McMillan. Interpolation and sat-based model checking. In *International Conference on Computer Aided Verification*, pages 1–13. Springer, 2003.

[17] Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT press, 2008.

[18] Bob Bixby. The gurobi optimizer, 2011.

[19] Alessandro Armando, Jacopo Mantovani, and Lorenzo Platania. Bounded model checking of software using smt solvers instead of sat solvers. *International Journal on Software Tools for Technology Transfer*, 11(1):69–83, 2009.

[20] Moshe Y Vardi. An automata-theoretic approach to linear temporal logic. In *Logics for concurrency*, pages 238–266. Springer, 1996.

[21] Kristin Y Rozier. Linear temporal logic symbolic model checking. *Computer Science Review*, 5(2):163–203, 2011.

[22] Iulian Ober, Susanne Graf, and Ileana Ober. Validation of uml models via a mapping to communicating extended timed automata. In *International SPIN Workshop on Model Checking of Software*, pages 127–145. Springer, 2004.

[23] Maria Sorea. Bounded model checking for timed automata. *Electronic Notes in Theoretical Computer Science*, 68(5):116–134, 2003.

[24] Edmund Clarke, Orna Grumberg, and Kiyoharu Hamaguchi. Another look at ltl model checking. In *International Conference on Computer Aided Verification*, pages 415–427. Springer, 1994.

[25] Paul Stephan, Robert K Brayton, and Alberto L Sangiovanni-Vincentelli. Combinational test generation using satisfiability. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 15(9):1167–1176, 1996.