Forward Analysis of Timed Automata

Pierre-Alain Reynier

LSV, CNRS UMR 8643, ENS de Cachan
61, av. du Prés. Wilson, 94235 Cachan Cedex - France
Email: reynier@lsv.ens-cachan.fr

SUPERVISORS: Patricia Bouyer, François Laroussinie
KEYWORDS: Timed automtata, reachability analysis, diagonal constraints

Abstract. Timed automata constitute a well adapted model for real-time aspects. However, a classical forward analysis algorithm used to verify safety properties has been showed to be incorrect in the general framework of timed automata. Nevertheless, it is correct when we restrict the class to timed automata with only non-diagonal guards. The aim of this work is to understand the role of diagonal guards and propose a method that provides correct algorithms in the general case.

1 Introduction

Safety properties constitute basic and fundamental properties in verification. Several tools implement algorithms which decide whether a system satisfies such properties. Most of them implement a classical version of a forward analysis algorithm. It has been proved recently that this classical forward analysis algorithm is not correct for the whole class of timed automata [Bou04]. This has been a big surprise since tools have been used by the past successfully.
Several solutions have been proposed, which basically remove diagonal guards from the automaton, but this suffers from a combinatorics explosion. In our approach, we will propose a new scheme based on a selection of diagonal guards which are responsible of spurious results in order to avoid this combinatorial explosion.
The structure of the paper is the following: we first recall some background on timed automata and we describe the implementation of the forward analysis algorithm that we study (section 2); we then motivate our choice of a counter-example guided refinement method (section 3) and show that the crucial step will be the detection of guilty guards, step that we develop in section 4. Full explanations can be found in my DEA report [Rey04].

2 Timed Automata and Forward Analysis

A *timed automaton* is a classical finite automaton with a set of variables called clocks. We consider a timed domain $\mathbb{T}$ such as $\mathbb{Q}^+$ or $\mathbb{R}^+$. In the following, we fix a set of clocks X. A *clock valuation* is a mapping v that assigns to each clock a time value.

Guards. Given a set of clocks X, the set of clock constraints (we simply say *guards*) over X, denoted $\mathcal{C}(X)$, is defined by the following grammar:

$\varphi ::= x \sim c \mid x - y \sim c \mid \varphi \wedge \varphi$ where $x, y \in X, c \in \mathbb{Z}, \sim \in \{<, \leq, =, \geq, >\}$.

The constraints of the form $x - y \sim c$ are called *diagonal guards*.

A clock constraint is k-*bounded* if it involves only constants between $-k$ and $+k$.

Timed Automata. A *timed automaton* over $\mathbb{T}$ is a tuple $\mathcal{A} = (\Sigma, Q, T, I, F, X)$, where Σ is a finite alphabet of actions, Q is a finite set of states, X is a finite set of clocks, $T \subseteq Q \times \mathcal{C}(X) \times \Sigma \times 2^X \times Q$ is a finite set of transitions, $I \subseteq Q$ is the subset of initial states and $F \subseteq Q$ is the subset of final states.

A configuration of a timed automaton is a pair (q, v) where q is a state and v is a valuation of clocks. Two evolutions are possible. Firstly, we can stay in the same location and let time elapse: every clock increases of the same duration $d \in \mathbb{T}$. Secondly, we can fire a discrete transition $t = (q, g, a, C, q') \in T$. This require that t is enabled, *i.e.* that the valuation satisfies the guard g. Then we move to state q' and reset the clocks of C.

Symbolic Representation. Because of the large number of possible configurations, algorithms and tools use a symbolic representation of sets of valuations: a *zone* is a set of valuations defined by a clock constraint. A zone is k-bounded if it is defined by a k-bounded constraint. We define an operator on zones called the k-approximation and denoted Approx_k. Given a zone Z, it computes the smallest (for the inclusion) k-bounded zone containing Z.

The Forward Analysis Algorithm. The problem of deciding emptiness of the language accepted by a timed automaton is PSPACE-complete ([AD94,AL02]). However, in practice, on-the-fly zone algorithms are implemented. In order to enforce termination, different kinds of abstractions are proposed to limit computations to a finite number of zones. One of the forward analysis algorithms for classical timed automata uses the k-approximation we have defined before. This algorithm is presented here as Algorithm 1. The operator *Post* computes the successor in one step by letting time elapse and then fire enabled transitions. Algorithm 1 simply iterates this operator following a breadth-first search.

Correctness Problem. The question of correctness has been well studied in [Bou04]. A first important remark is that Algorithm 1 computes an overapproximation of reachable states since we have $Z \subseteq \text{Approx}_k(Z)$. The following results allow us to well understand the situation. Theorem 1 is exactly the result we have anounced in the introduction:

Theorem 1 There exists a timed automaton with 4 clocks and with diagonal constraints such that Algorithm 1 is not correct on this input, whatever the integer k is.

The next theorem explains why this bug has not been discovered before while many case studies have been successfully analyzed [HSLL97,BBP02]: most case studies didn't use diagonal guards.

Algorithm 1 Classical forward analysis algorithm for timed automata.

Require: $\mathcal{A}$ a timed automaton, k the constant of extrapolation
Ensure: $\mathcal{L}(\mathcal{A}) = \emptyset$?
1: Visited $:= \emptyset$;
2: Waiting $:= \{(q_0, \text{Approx}_k(Z_0))\}$; /* Z_0 initial zone */
3: **repeat**
4: Get and Remove (q, Z) from Waiting;
5: **if** q is a final state **then**
6: Return "A final state is reachable.";
7: **else**
8: **if** there is no $(q, Z') \in$ Visited such that $Z \subseteq Z'$ **then**
9: Visited $:=$ Visited $\cup \{(q, Z)\}$;
10: Successor $:= \{(q', \text{Approx}_k(\mathit{Post}(Z, e))) \mid e \text{ transition from } q \text{ to } q'\}$;
11: Waiting $:=$ Waiting $\cup$ Successor;
12: **end if**
13: **end if**
14: **until** Waiting $= \emptyset$;
15: Return "No final state is reachable.";

Theorem 2 Algorithm 1 is correct for timed automata that have no diagonal guards.

This latter theorem leads naturally to study the role of the diagonal guards. The following result has been a great help in the work we will present here after.

Theorem 3 ([BDGP98]) Given a timed automaton $\mathcal{A}$ containing a diagonal guard g, we can construct a timed automaton $\mathcal{B}$ not containing anymore the guard g such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{B})$. The size of the new automaton is twice the size of the previous one.

Theorem 3 gives naturally a solution to our problem. Indeed, it is easy to see that given a timed automaton $\mathcal{A}$, we can compute a new timed automaton $\mathcal{C}$, containing no diagonal guards, and which is equivalent to $\mathcal{A}$ with respect to the accepted language. We can then apply Algorithm 1 to $\mathcal{C}$ and by Theorem 2, we obtain correctness of this method. Unfortunately, the size of $\mathcal{C}$ is exponential in the size of $\mathcal{A}$. Therefore, this solution is not efficient in terms of complexity.

3 Counter-Example Guided Refinement

Motivations. Algorithm 1 computes an overapproximation of reachable states. The only problem thus stands in that some states may be found reachable whereas they are not. It is easy to check if a path found by the algorithm is a spurious or a real path. We could think it is thus sufficient to reuse the classical algorithm 1 and add a procedure that checks its answers. Unfortunately, we can show that this method doesn't work because of the inclusion test of line 8. Indeed, approximations made during the discover of a spurious counter-example can prevent the algorithm from discovering later a real counter-example. This leads us to use a more complicated method.

Counter-example guided refinement. We have seen in section 2 that there exists a procedure that builds an equivalent automaton which does not contain anymore diagonal guards but this has an expensive cost in terms of complexity. The idea we will use here relies on the fact that the bug may depend only on one or two diagonal guards of the system, and surely not on all diagonal guards. Therefore, we will try to choose "in a clever way" the guards we remove. The general scheme is depicted on figure 1.

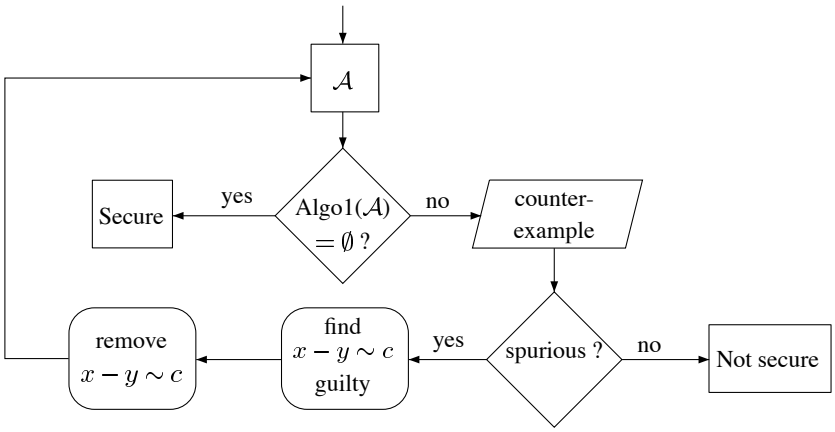

Fig. 1. Counter-example guided refinement.

As we claimed before, the test whether a counter-example is spurious is not difficult. We have also mentioned the procedure for removing a given diagonal guard. The only part that remains to be explained is the detection of guilty diagonal guards. This constitutes the difficult part of this method. Different approaches can be planned. On one side, we can try to find a guard, or a set of guards, that ensures that after having removed this guards, the algorithm will not obtain the same spurious counter-example anymore. On the other side, we can prefer having an algorithm with a lower complexity, but which may not have the strong property above.

4 Detection of Guilty Guards

We focus now on the analysis of a spurious counter-example to find "guilty" guards. It means that we consider a path in a timed automaton such that the computation made by the algorithm leads to a non-empty zone at the end of the path, whereas the exact computation of successors along this path leads to an empty zone. As we know that the algorithm is correct when there is no diagonal guards, we have that there is at least one diagonal guard along this path. Our goal is then to detect "guilty" diagonal guards along this path. Therefore we develop a method that checks whether a diagonal guard can be "simulated" by non-diagonal ones.

Definition 1 Let g be the first diagonal guard of a transition t on a spurious counter-example P in a timed automaton. We assume that g is a simple guard and that t is the

last transition of P. The diagonal guard g is said to be *non-diagonal equivalent* if there exists a timed automaton with no diagonal guards and a path Q in this automaton such that the zones computed by algorithm 1 at the end of P and Q are the same.

Since we have seen that the computations made by algorithm 1 on automata with no diagonal guards are correct with respect to reachability, this definition guarantees that Algorithm 1 has not made yet any mistake.

To check whether this criteria is satisfied, we can use the construction associated to Theorem 3. We consider the first diagonal guard appearing along the path P of a spurious counter-example, and we remove this guard by adding constraints on the last reset of one of the two clocks appearing in the guard. This procedure is illustrated on figure 2. If this construction does not lead to a satisfying automaton, we can consider that the guard g is suspicious.

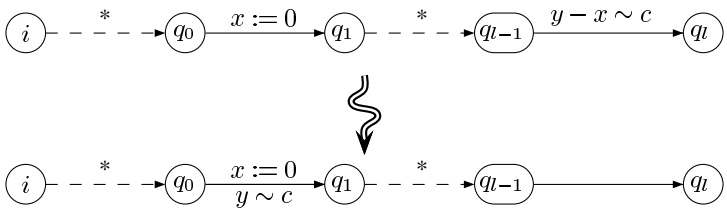

Fig. 2. Local refinement step.

This test gives us a way to detect suspicious guards. Morover, we can obtain a procedure which has the strong property described above. Indeed, we can iterate this procedure in order to build a set of guards which ensures to eliminate the spurious counter-example studied in this refinement step. In fact we just have to remove successively each diagonal guard according to the local refinement step described above. At the end, we have no more diagonal guard on this path and because of Theorem 2, the new computation will announce that the final state along this path is no more reachable since the studied counter-example is spurious.

To conclude this section, we discuss an other aspect. As we explained before, it can also be interesting to have a simple and low cost procedure to choose diagonal guards, even if this procedure does not build a relevant or complete set of guilty guards. For example, we can choose randomly a diagonal guard appearing along the spurious counter-example, or take the first or the last one. There are thus several way to implement such a procedure.

5 Conclusion

The problem we have addressed here is related to verification of timed systems. It has been proved that a classical forward analysis algorithm used to decide safety and reach-

ability properties is not correct as soon as diagonal guards are allowed. Our aim was to understand the role of diagonal guards and to propose correct algorithms in the general framework.

We have presented in section 2 the precise context of the problem we want to tackle: ensure correctness of a forward computation for timed automata using diagonal guards. The construction of [BDGP98] gives raise to a first algorithm which consists in removing all the diagonal guards. Unfortunately, this has an exponential cost. Another algorithm based on a related idea has been proposed in [BY03]. But it also suffers from the same drawback. Whereas there seems to be very few bugs in timed automata built from real systems, and whereas it seems that not every guard is responsible, the two methods above systematically apply the procedure which removes a diagonal guard to every guard appearing in the automaton. That's why we have tried to develop a method that reduces the number of calls to this procedure by using counter-example guided refinement. We have explained this general approach in section 3, and detailed a possible refinement step in section 4. This leads to numerous solutions whose efficiency has not been tested in practice yet.

Future work will consist in developping another procedure for detecting guilty diagonal guards based on a real-time history of errors made by algorithm 1. We also would like to compare all the different algorithms we have proposed.

References

[AD94] Rajeev Alur and David Dill. A theory of timed automata. *Theoretical Computer Science (TCS)*, 126(2):183–235, 1994.

[AL02] Luca Aceto and François Laroussinie. Is your model-checker on time ? on the complexity of model-checking for timed modal logics. *Journal of Logic and Algebraic Programming (JLAP)*, 52–53:7–51, 2002.

[BBP02] Béatrice Bérard, Patricia Bouyer, and Antoine Petit. Analysing the PGM protocol with UPPAAL. In *Proc. 2nd Workshop on Real-Time Tools (RT-TOOLS'02)*, 2002. Proc. published as Technical Report 2002-025, Uppsala University, Sweden.

[BDGP98] Béatrice Bérard, Volker Diekert, Paul Gastin, and Antoine Petit. Characterization of the expressive power of silent transitions in timed automata. *Fundamenta Informaticae*, 36(2–3):145–182, 1998.

[Bou04] Patricia Bouyer. Forward analysis of updatable timed automata. *Formal Methods in System Design*, 24(3):281–320, 2004.

[BY03] Johan Bengtsson and Wang Yi. On clock difference constraints and termination in reachability of timed automata. In *Formal Methods and Software Engineering, 5th International Conference on Formal Engineering Methods (ICFEM'03)*, volume 2885 of *Lecture Notes in Computer Science*. Springer-Verlag, 2003.

[HSLL97] Klaus Havelund, Arne Skou, Kim G. Larsen, and Kristian Lund. Formal modeling and analysis of an audio/video protocol: An industrial case study using UPPAAL. In *Proc. 18th IEEE Real-Time Systems Symposium (RTSS'97)*, pages 2–13. IEEE Computer Society Press, 1997.

[Rey04] Pierre-Alain Reynier. Analyse en avant des automates temporisés. Master's thesis, DEA Algorithmique, Paris, 2004.