Shrinktech: A Tool for the Robustness Analysis of Timed Automata

Ocan Sankur

LSV, ENS Cachan & CNRS, France.
sankur@lsv.ens-cachan.fr

Abstract. We present a tool for the robustness analysis of timed automata, that can check whether a given time-abstract behaviour of a timed automaton is still present when the guards are perturbed. The perturbation model we consider is *shrinking*, which corresponds to increasing lower bounds and decreasing upper bounds in the clock guards by parameters. The tool synthesizes these parameters for which the given behaviour is preserved in the new automaton if possible, and generates a counter-example otherwise. This can be used for 1) robustness analysis, and for 2) deriving implementations under imprecisions.

1 Introduction

Timed Automata and Robustness. Timed automata [3] are a well-established formal model for real-time systems. They can be used to model systems as finite automata, while using, in addition, a finite number of clocks to impose timing constraints on the transitions. Timed automata are, however, abstract models, and therefore make unrealistic assumptions on timings, such as perfect continuity of clocks, infinite-precision time measures and instantaneous reaction times. An important amount of work has been done in the timed automata literature to endow timed automata with a realistic semantics. The works [14] and [10] showed that perturbations on clocks, either imprecisions or clock drifts, and regardless of how small they are, may yield additional qualitative behaviours in some timed systems. On the other hand, assuming bounds on the reaction times can disable desired behaviours [8, 1]. These observations mean that there is a need for checking the *robustness* of timed automata models, that is, whether the behaviour of a given timed automaton is preserved in presence of small perturbations. Robustness is an important property of critical embedded systems [12], since it requires that the system will behave correctly when the environment's behaviour deviates slightly from the assumptions.

Clock Imprecisions and Shrinking. A prominent approach to model imprecisions in timed automata, initiated in [11], consists in introducing imprecisions in the model by syntactically *enlarging* all guards, that is, turning a guard $x \in [a, b]$ into $x \in [a - \Delta, b + \Delta]$ for some parameter $\Delta > 0$. Model-checking algorithms on timed automata have been revisited in order to take into account such imprecisions (see *e.g.* [10, 6]). These algorithms check whether any really new behaviour appears when timing constraints are relaxed by a small (parameterized) amount.

Recently we studied the dual notion of robustness, which consists in checking whether any behaviour is lost when the guards are *shrunk*, that is tightened by

a small (parameterized) amount. More precisely, shrinking means converting a guard $x \in [a, b]$ into $x \in [a + \delta, b - \delta]$ for some $\delta > 0$. In [15], we showed that one can decide whether all guards can be shrunk –by possibly different amounts, so that the resulting timed automaton can still time-abstract simulate the original automaton. In this case, one can also synthesize these shrinking parameters for each atomic guard. By checking shrinkability of timed automata, one ensures that the behaviour of the automaton does not depend on exact timings, or on its ability to take the transitions on the boundaries of the guards. A shrinkable timed automaton preserves all its behaviours when, for instance, task execution times are shorter than the worst-case, and waiting times are longer than the best-case. One can also detect unrealistic runs, including Zeno runs [15]. We believe that shrinkability complements the robustness approach based on guard enlargement of [11, 10].

Shrinkability can also be used for deriving implementations with imprecise clocks. In fact, if the guard $x \in [a, b]$ is shrunk into $x \in [a + \delta, b - \delta]$, then under imprecisions *à la* [11], this guard becomes $x \in [a + \delta - \Delta, b - \delta + \Delta] \subseteq [a, b]$, where the inclusion holds whenever $\Delta < \delta$. Hence, the behaviours of a shrunk timed automaton with bounded imprecisions (*i.e.* guard enlargement) are entirely included in the behaviours of the initial timed automaton. Further, using shrinkability, one can synthesize parameters δ for each guard, so that the resulting automaton still contains some useful time-abstract behaviour.

2 Shrinkability

Let us define shrinkability more formally. We assume that the reader is familiar with the syntax and semantics of timed automata, and refer to [3] for details. We only need the following definitions. Given a finite clock set $\mathcal{C}$, an *atomic guard* is an expression of the form $x \leq k \mid x \geq k \mid x - y \leq k \mid x - y \geq k$, where $x, y \in \mathcal{C}$ and $k \in \mathbb{Z}$. A *guard* is a conjunction of atomic guards. The *shrinking* of an atomic guard g by δ, denoted by $\langle g \rangle_{-\delta}$ is defined as follows.

$$\langle k \leq x \rangle_{-\delta} = k + \delta \leq x, \qquad \langle x \leq l \rangle_{-\delta} = x \leq l - \delta,$$
$$\langle k \leq x - y \rangle_{-\delta} = k + \delta \leq x - y, \qquad \langle x - y \leq l \rangle_{-\delta} = x - y \leq l - \delta.$$

Note that the only variables that appear in timed automata are clocks. Discrete variables with bounded domains can be considered, but we assume these are encoded in the locations.

Let $\mathcal{A}$ be a timed automaton, and let I be the vector of the atomic guards of $\mathcal{A}$. Given a vector $\boldsymbol{\delta}$ of nonnegative rational numbers indexed by I, we denote by $\mathcal{A}_{-\boldsymbol{\delta}}$, the automaton obtained from $\mathcal{A}$ by shrinking each atomic guard by the corresponding element of $\boldsymbol{\delta}$. We are going to write the vector $\boldsymbol{\delta}$ as $\boldsymbol{k}\delta$ for an integer vector $\boldsymbol{k}$ and rational δ. This is always possible since we are interested in rational parameters. Figure 1 is an example of shrinking.

We are interested in shrinking the atomic guards of a given timed automaton by positive values, while preserving *some* of the behaviours. We only consider timed automata with non-strict guards; in fact, using strict guards makes little sense when one is interested in shrinking (or enlarging) the guards [10]. We also assume that the edges have distinct labels, since we are interested in comparing

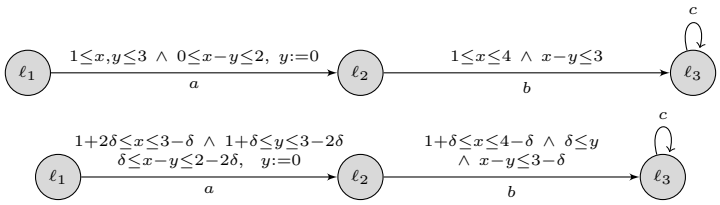

Fig. 1. A timed automaton $\mathcal{A}$ (above) and its shrinking $\mathcal{A}_{-\mathbf{k}\delta}$ (below). Timed automaton $\mathcal{A}_{-\mathbf{k}\delta}$ can time-abstract simulate $\mathcal{A}$ for all $\delta \in [0, \frac{1}{6}]$ ([15]).

two timed automata that have the same underlying structure. The problem is formulated as follows:

Definition 1 (Shrinkability). *Given a timed automaton $\mathcal{A}$, and a finite automaton $\mathcal{F}$ such that $\mathcal{F} \sqsubseteq_{ta} \mathcal{A}$, decide whether for some $\boldsymbol{\delta} > 0$, $\mathcal{F} \sqsubseteq_{ta} \mathcal{A}_{-\boldsymbol{\delta}}$.*

In this definition, $\sqsubseteq_{\mathrm{ta}}$ denotes *time-abstract delay simulation*. We say that $\mathcal{A}$ is *shrinkable w.r.t.* $\mathcal{F}$ if the above condition is satisfied. Thus, shrinkability requires that some behaviour $\mathcal{F}$, that is included in the initial model $\mathcal{A}$, should be still possible in the shrunk automaton. When $\mathcal{F}$ is the region graph of $\mathcal{A}$ [3], or a time-abstract bisimulation quotient [16], shrinkability implies that the shrunk automaton can time-abstract simulate the original automaton. In this case, we say that $\mathcal{A}$ is simply *shrinkable*. Shrinkability w.r.t. $\mathcal{F}$ is decidable in polynomial time if $\mathcal{F}$ is part of the input; shrinkability is decidable in exponential time, if the time-abstract bisimulation graph is not given. The vector $\boldsymbol{\delta}$ can be computed in the same time complexity.

The present tool builds on the theoretical results presented in [15]. There, we show that given a finite automaton $\mathcal{F}$, the shrinking parameter of each atomic guard can be expressed as a function of the other parameters using only maximization and sum. The problem is then reduced, in polynomial time, to solving nonlinear fixpoint equations in the max-plus algebra. We gave in [15], graph-based algorithms to solve these equations. Appendix A explains by a simple example why some timed automata are not shrinkable.

Partial shrinkability. Although we defined shrinkability by requiring that *all* atomic guards should be shrunk by a positive amount, one can relax this condition and shrink only some of the guards; our algorithms are valid also in this case. In our experiments, we shrunk all the guards but equality constraints.

3 The tool `shrinktech`

We present the tool `shrinktech` that analyzes the shrinkability of timed automata and synthesizes shrinking parameters. Given a network of timed automata, the tool either finds a counter-example to shrinkability, such as a path or a cycle that cannot be executed by any shrinking of the automaton, whatever the value of δ's are, or outputs a shrinking of the timed automata that witnesses the shrinkability.

Figure 3 shows an overview of the tool. To check the shrinkability of a timed automaton, the user can either provide a finite automaton $\mathcal{F}$, or let `shrinktech`

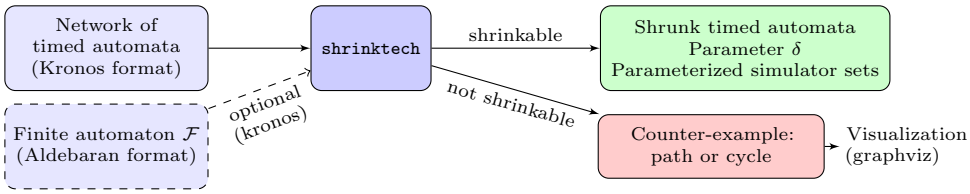

Fig. 2. Overview of `shrinktech`.

compute the full finite bisimilarity graph using Kronos[1]. Note that if the full bisimilarity graph is too big, one can also try to shrink with respect to a portion of it, or with respect to a randomly generated trace. This is to be compared with bounded model-checking, which is useful for detecting bugs, but also for "partially" proving the correctness of a system. The tool comes with scripts to compute the bisimilarity graph, extract some (random) portion of it, and generate random executions.

The tool `shrinktech` can be used for several kinds of systems modelled by timed automata. We believe it can be used mainly for two purposes:

1. Robustness analysis, to find out whether the behaviour of the system is preserved when the time bounds are disturbed (shrunk). This analysis complements the robustness checking by enlarging the guards as in [11, 2]. This can for instance help to detect unrealistic executions such as Zeno or other convergence phenomena, but also timing anomalies in scheduling problems (see [5]).
2. Deriving implementation from timed automata. As explained above, the behaviour of a shrunk timed automaton is included in that of the initial model in presence of imprecisions. So lower and upper bounds on the delays can be "shrunk" in the implementation to guarantee that these will be respected despite imprecisions. In fact, we proved in [15] that shrinkable timed automata can be implemented in a concrete semantics with imprecise clocks and reaction times.

Implementation details and availability. The tool is implemented in C++ and the source code has about 5Klocs. It uses the Uppaal DBM library [2], and implements a parameterized extension of this data structure, introduced in [15]. The input formats are (networks of) timed automata in the Kronos format, and finite automata in the Aldebaran format[3]. The tool Kronos can be plugged in the tool-chain in order to compute the finite time-abstract bisimilarity graph of a given timed automaton, to be used as the finite automaton $\mathcal{F}$. Shrinktech is

[1] Kronos is a model-checker for timed automata [7], that can *minimize* the region graph of a timed automaton as described in [16]. It is available at `http://www-verimag.imag.fr/DIST-TOOLS/TEMPO/kronos/`

[2] `http://people.cs.aau.dk/~adavid/UDBM/`

[3] This is a graph description format of the CADP tool suite, also used by Kronos. See `http://www.inrialpes.fr/vasy/cadp/`

open source software and is distributed under GNU General Public Licence 3.0. It is freely available at: `http://www.lsv.ens-cachan.fr/Software/shrinktech`

4 Experimental Results

We used `shrinktech` on several case studies found in the literature. The table 1 summarizes the results. The Lip Synchronization Protocol has been the subject of robustness analysis (by guard enlargement) before [13]. This is an algorithm that synchronizes video and sound streams that arrive in different frequencies. The model is not shrinkable neither for video frames arriving in exact frequency, nor for those arriving within a bounded interval. Observe that the model is shrinkable w.r.t. a small subgraph with 501 nodes, but it is not shrinkable w.r.t. the whole graph which has 4484 nodes. Shrinkable models include Philips Audio Retransmission protocol [9], and some asynchronous circuit models. We were able to analyze Fischer's Mutual Exclusion Protocol upto 4 agents; while for 5 agents we could only partially analyze w.r.t. a randomly generated trace. The non-shrinkability of most models is due to equality constraints. In fact, although we only shrink non-punctual guards, some behaviours may still disappear immediately, however small the shrinking parameter is (see Appendix A).

Note that some of these models were designed at a level of abstraction where imprecisions were not taken into account. So, our results do not necessarily imply that these systems are not robust, but rather that the present models are not good for direct implementation. This is best illustrated in the Latch Circuit models, where the exact model that extensively uses equalities is not shrinkable, but its relaxation to intervals is. Notice also how most of the circuit models, which define bounds on stabilization times are shrinkable.

Table 1. The column sim-graph is the number of states and the number of transitions of the finite automaton $\mathcal{F}$ w.r.t. which the shrinkability is checked. An asterisk indicates bounded shrinkability, where only a subgraph of the time-abstract bisimulation graph (given by a BFS) or a random trace was used. The tests were performed on an Intel Xeon 2.67 GHz. All models are available on the tool's website.

Model	**states**	**trans**	**clocks**	**sim-graph**	**time**	**shrinkable**
Lip-Sync Prot. (Exact)	230	680	5	4000/8350* (subgraph)	9s	No
Lip-Sync Prot. (Interval)	230	680	5	501/1282* (subgraph)	9s	Yes*
Lip-Sync Prot. (Interval)	230	680	5	4484/48049	28s	No
Philips Audio Prot.	446	2097	2	437/2734	46s	Yes
Root Contention Prot.	65	138	6	500/3455* (subgraph)	7s	No
Train Gate Controller	68	199	11	952/8540	34s	No
Fischer's Protocol 3	152	464	3	472/4321	20s	Yes
Fischer's Protocol 4	752	2864	4	4382/65821	310min	Yes
Fischer's Protocol 5	3552	16192	5	10000/10000* (trace)	42s	Yes*
And-Or Circuit	12	20	4	80/497	1.3s	Yes
Flip-Flop Circuit	22	34	5	30/64	0.9s	Yes
Latch Circuit (Interval)	32	77	7	105/364	1.6s	Yes
Latch Circuit (Exact)	32	77	7	100/331	0.6s	No

Our approach depends on the computation of the finite automaton $\mathcal{F}$, thus it is limited by the feasability of this computation. To deal with this problem, one could consider adapting the algorithm of [16] to compute on-the-fly some bounded part of the graph. To be able to treat even larger systems, we will consider extending the theoretical results of [15], in order to use under- and over-approximations of the automaton $\mathcal{F}$, and refine these by counter-examples.

5 Related Work

Existing verification tools for timed automata may be used for *non-parameterized* robustness checking by modeling explicitly the imprecisions, at the cost of increasing the size of the models [2]. The semi-algorithm of HyTech was used to synthesize guard enlargement parameters in timed automata in [11]. An extension of Uppaal for robustness against guard enlargement was used in [13]; this feature is no longer available in Uppaal. Note that shrinkability cannot be solved by existing model-checkers for timed automata since we are interested in parameter synthesis so as to ensure time-abstract simulation. Other similar work includes (the undecidable problem of) parameter synthesis in timed automata, where guards are written using parameters, and one tries to find the valuations for which the system satisfies some specification, as in [4]. This is difficult to realize due to the large number of parameters (upto millions) and the time-abstract simulation condition we consider. Robustness against large decreases in task execution times using simulation was considered in [1].

References

1. Tesnim Abdellatif, Jacques Combaz, and Joseph Sifakis. Model-based implementation of real-time applications. In EMSOFT'10, p. 229–238. ACM, 2010.
2. Karine Altisen and Stavros Tripakis. Implementation of timed automata: An issue of semantics or modeling? In FORMATS'05, LNCS 3829, p. 273–288. Springer.
3. Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, April 1994.
4. Étienne André, Laurent Fribourg, Ulrich Kühne, and Romain Soulat. IMITATOR 2.5: A tool for analyzing robustness in scheduling problems. In FM'12, LNCS 7436. Springer, 2012.
5. Patricia Bouyer, Nicolas Markey, and Ocan Sankur. Robust reachability in timed automata: A game-based approach. In ICALP'12, LNCS 7392, p. 128–140. Springer.
6. Patricia Bouyer, Nicolas Markey, and Ocan Sankur. Robust model-checking of timed automata via pumping in channel machines. In FORMATS'11, LNCS 6919, p. 97–112. Springer, 2011.
7. Marius Bozga, Conrado Daws, Oded Maler, Alfredo Olivero, Stavros Tripakis, and Sergio Yovine. Kronos: A model-checking tool for real-time systems. In CAV'98, p. 546–550. Springer, 1998.
8. Franck Cassez, Thomas A. Henzinger, and Jean-François Raskin. A comparison of control problems for timed and hybrid systems. In HSCC'02, LNCS 2289, p. 134–148. Springer, March 2002.
9. C. Daws and S. Yovine. Two examples of verification of multirate timed automata with kronos. In RTSS'95, p. 66–75. IEEE Computer Society Press, 1995.
10. Martin De Wulf, Laurent Doyen, Nicolas Markey, and Jean-François Raskin. Robust safety of timed automata. *FMSD*, 33(1-3):45–84, December 2008.
11. Martin De Wulf, Laurent Doyen, and Jean-François Raskin. Almost ASAP semantics: From timed models to timed implementations. *Formal Aspects of Computing*, 17(3):319–341, 2005.
12. Thomas A. Henzinger and Joseph Sifakis. The embedded systems design challenge. In FM'06, LNCS 4085, p. 1–15, Hamilton, Canada, 2006. Springer.
13. Piotr Kordy, Rom Langerak, and Jan Willem Polderman. Re-verification of a lip synchronization protocol using robust reachability. In *FMA*, p. 49–62, 2009.
14. Anuj Puri. Dynamical properties of timed automata. *Discrete Event Dynamic Systems*, 10(1-2):87–113, January 2000.
15. Ocan Sankur, Patricia Bouyer, and Nicolas Markey. Shrinking timed automata. In FSTTCS'11, LIPIcs 13, p. 90–102. Leibniz-Zentrum für Informatik, 2011.
16. Stavros Tripakis and Sergio Yovine. Analysis of timed systems using time-abstracting bisimulations. *Form. Methods Syst. Des.*, 18(1):25–68, January 2001.

A Example: Non-shrinkability

Non-shrinkability of timed automata can be due either to the disappearance of a finite behaviour, or that of an infinite behaviour. We illustrate the first kind of non-shrinkability here; the next section gives an example to the second one.

Let us describe a simple system that is not shrinkable. We consider the timed automaton $\mathcal{A}_2$ of Fig. 3. The system has to give a feedback (fb) at least every 2 time units, through the self-loop on ℓ_0. At any time, it can receive a message m, which it can transfer to another system after a delay of at least 1 and at most 2. In fact, let us assume that the target system only listens between this time interval. If the message is sent too early or too late, it will be lost. In addition, while the system is in the state ℓ_1 preparing for transferring the message, it can decide to send the message to a buffer by realizing the $\mathsf{btransfer}$ action as long as $x \leq 1$. The sytem may also receive a dupplicate (m') of the message while $1 \leq x \leq 2$.

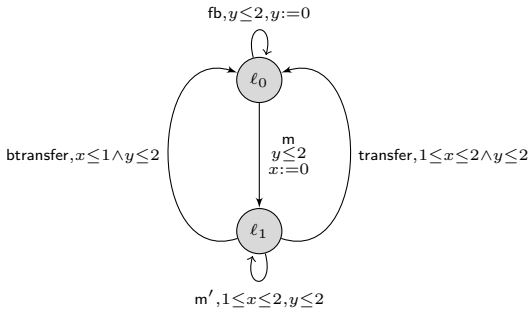

Fig. 3. Timed automaton $\mathcal{A}_2$.

A good implementation must only $\mathsf{transfer}$ the message when $x \in [1+\delta, 2-\delta]$ for a safe $\delta > 0$, and should always give the feedback when $y \leq 2-\delta$. Hence it makes sense to do shrinkability analysis on this system to check whether any significant behaviour is lost under shrinking.

To simplify the discussion, we will only concentrate on a simple behaviour of $\mathcal{A}_2$, and summarize it as the timed automaton $\mathcal{A}_3$ of Fig. 4. Here, $\mathcal{A}_3$ consists of the unfolding of $\mathcal{A}_2$ for two transitions from ℓ_1. By investigating the state space of $\mathcal{A}_2$, one can see that it contains a branching: at ℓ_1, both $\mathsf{btransfer}$ and $\mathsf{transfer}$ actions are available. The state space is illustrated in Fig. 5 (on the left). On the other hand, under any shrinking, the branching $\mathsf{btransfer}$ and $\mathsf{transfer}$ is disabled (the right of Fig. 5). This means that a controller that waits for the repetition of the message before making the decision between $\mathsf{btransfer}$ and $\mathsf{transfer}$ is not robust. Therefore, the timed automaton $\mathcal{A}_3$ (hence $\mathcal{A}_2$) is not shrinkable w.r.t. its own time-abstract behaviour. Note however that the shrunk automaton does contain the behaviours $\mathsf{m} \cdot \mathsf{m}' \cdot \mathsf{transfer}$ and $\mathsf{m} \cdot \mathsf{m}' \cdot \mathsf{btransfer}$ so if the branching is not important, one can eliminate the branching in the time-abstract bisimulation quotient and prove shrinkability w.r.t. this smaller finite automaton.

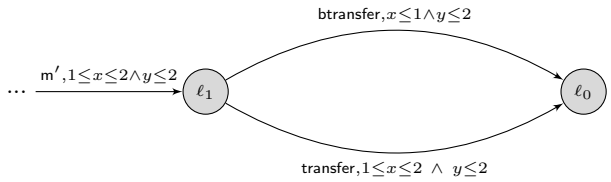

Fig. 4. Timed automaton $\mathcal{A}_3$, modelling a part of $\mathcal{A}_2$.

Non-shrinkability is not always due to a finite path that is disabled because of shrinkings. In the next section, we give an example of a timed automaton whose shrinkings cannot simulate a cycle of the bisimilarity graph, but can simulate any finite unfolding of it.

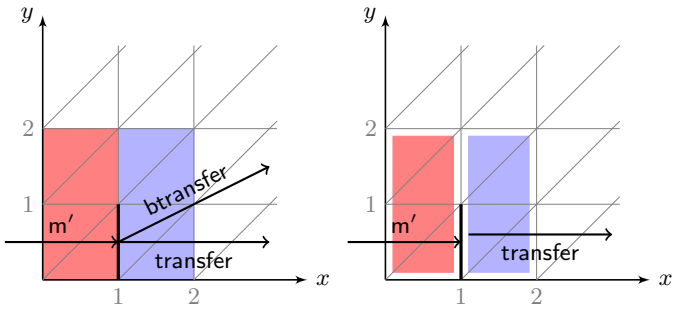

Fig. 5. The figure on left illustrates the behaviour in the original timed automaton, while on the right, the behaviour of the shrunk automaton is given. The red area is the guard of btransfer, while the blue area is the guard of transfer. The thick black segment is a set of states reachable after the action m′. From this set, both actions transfer and btransfer are available in the original timed automaton, while the latter is disabled in the shrunk automaton. Hence, the branching disappears under any shrinking.

B Using `shrinktech`

Simple usage We explain here how to use the **`shrinktech`** tool on a simple example. We would like to check whether the timed automaton $\mathcal{A}_3$ of Fig. 6 below is shrinkable.

This automaton can be described in Kronos timed automaton format as given in Fig. 7. Assuming Kronos is installed on the system, the simplest way to check shrinkability is the following:

```
> shrinktech automaton.tg
[...]
Starting shrinkability analysis on files automaton.tg and automaton.aut
Warning: Some edges have equality constraints and will not be shrunk (see log file)

Timed automaton is NOT SHRINKABLE.  Counter-example generated in automaton.log and automaton.png
```

Shrinktech first calls kronos (the output is omitted) to compute the time-abstract bisimulation graph of the timed automaton into `automaton.aut`. It then answers that the automaton is not shrinkable. Note also that a warning is generated since not all guards were shrunk (`shrinktech` shrinks all the guards but equality constraints). The counter-example, that is a subgraph of `automaton.aut` that cannot be simulated by any shrinking of `automaton.tg` is generated and it is shown in Fig. 7 (this is the file `automaton.png`). The analysis reveals that the cycle BC cannot be simulated in the shrunk automaton, whatever the values of the shrinkings are. In fact, for any shrinking, the cycle can be taken only a finite number of times after which the automaton is blocked.

When the program terminates, the file `automaton.log` contains the text representation of the counter-example, also other useful information such as the shrunk guards and the parameterized simulation sets. The counter-example is also generated as a dot file in `automaton.dot`, and a PNG image is generated in `automaton.png` provided that Graphviz is installed on the system.

Note that `shrinktech` works (when necessary) on a copy of the given timed automaton (the default value is `ta.tg`) because it adds annotations to the edge labels. One can change this file name by passing `-out myfile.tg` as argument to the program.

With a custom finite automaton One can also give a custom finite automaton `automaton1.aut` as an input to the program. For instance, one can try to unfold the loop BC several times to check whether some shrinking of $\mathcal{A}_3$ can at least simulate a finite number of iterations. Consider the finite automaton of Fig. 8. This automaton can be simulated by `automaton.aut`, thus also by $\mathcal{A}_3$ (note that the timed automaton must be able to simulate the given finite automaton; an error will be generated otherwise). See also the `--gentrace` option in Section B.1 to randomly generate such a trace.

One can run `shrinktech` providing both the timed automaton and the finite automaton:

```
> shrinktech automaton.tg automaton1.aut
Starting shrinkability analysis on files automaton.tg and automaton1.aut
Warning: Some edges have equality constraints and will not be shrunk (see log file)

Timed automaton is SHRINKABLE.

Run again with --simulator_sets option to compute solution.
See automaton.log for details.
```

This time the automaton is shrinkable with respect to the given finite automaton. By default, `shrinktech` does not compute the simulator sets of the shrunk automaton, *i.e.* the set of states of the shrunk timed automaton that

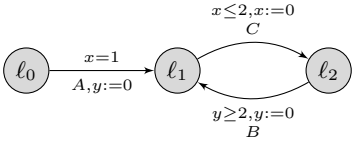

Fig. 6. Timed automaton $\mathcal{A}_3$.

```
#states 3
#trans  2
#clocks  2
X
Y

state: 0
invar: TRUE
trans:
X = 1 => A; RESET{Y}; goto 1

state: 1
invar: TRUE
trans:
X <= 2 => B; RESET{X}; goto 2

state: 2
invar: TRUE
trans:
Y >= 2 => C; RESET{Y}; goto 1
```

```
des(0,10,6)
(0, "A",1)
(0, "A",2)
(1, "B",3)
(1, "B",4)
(1, "B",5)
(3, "C",1)
(3, "C",2)
(4, "C",2)
(5, "C",1)
(5, "C",2)
```

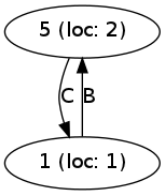

Fig. 7. The Kronos timed automaton file `automaton.tg` (on the left), the time-abstract bisimulation graph `automaton.aut` generated with Kronos, and the counter-example generated by `shrinktech` (on the right). Nodes are labelled by i(Loc: j) where i is the node of the given automaton $\mathcal{F}$ and j is the unique location of the timed automaton that corresponds to this node i. Edges are labelled by the edge labels of the timed automaton.

```
des(0,7,8)
(0, "A",1)
(1, "B",2)
(2, "C",3)
(3, "B",4)
(4, "C",5)
(5, "B",6)
(6, "C",7)
```

Fig. 8. Finite automaton `automaton1.aut`

can simulate each state of the finite automaton. But this can be done using the `--simulator_sets` option. In this case, the log file will contain the description of the simulator sets. The log file will also report an upper bound on δ below which the simulation holds. Furthermore, a shrinking of the given timed automaton is generated in a local folder named `shrunk`.

Here is an excerpt from the log file.

```
------------------ PARAMETERIZED SHRUNK SIMULATOR SETS ----------------------
-- Substituting delta = 0 gives the simulator sets of the original automaton
-------------------------------------------------------------------------------
Simulator set of the finite automaton node(0)
0 +(p[0])delta <= X <= 1 -(p[2])delta  and
0 +(p[1])delta <= Y  and
X - Y  <= 1 -(p[3])delta

Simulator set of the finite automaton node(1)
0 +(p[4])delta <= X <= 2 -(p[6])delta  and
0 +(p[5])delta <= Y  and
X - Y  <= 2 -(p[7])delta

[...]
```

These describe the set of (parameterized) states that can simulate the nodes 0 and 1 of the finite automaton. Here, $p[i]$'s are positive integers to be determined by shrinkability analysis. The log file also contains an instantiation of these simulator sets for the parameter values computed by `shrinktech` and for the largest possible δ:

```
------------------------ DISCRETE SOLUTION --------------------------
-- Scaling factor: 100
-----------------------------------------------------------------------
Discrete simulator set of the finite automaton node(0)
X <= 100 and
X - Y  <= 99

Discrete simulator set of the finite automaton node(1)
X <= 199 and
X - Y  <= 194

[...]
```

Note that we scale here the constants so that we only have to deal with integers, instead of floating point numbers.

See also Section B.1 for additional tools and scripts that can be helpful to extract a finite automaton.

Using multiple files In Kronos timed automaton format, several components communicating by synchronization can be given to describe one large system. In this case, one can simply give `shrinktech` the list of all the components:

```
> shrinktech component1.tg component2.tg .... componentK.tg
```

In this case, `shrinktech` will invoke Kronos to compute the product, and then to compute the bisimulation graph. By default, the product automaton will be written in `ta.tg` and shrinkability analysis will be run on this file. One can modify this target file using the option `-out target.tg`.

B.1 Command-line Options and Additional Tools

`--aut` When run with this option, shrinktech will only compute the product of the given timed automata (if more than one is given), and output the time-abstract bisimilarity graph. The output file can be specified using `-out target.aut`; the default is `ta.aut`. This feature uses Kronos.

`--simulator_sets` This option tells `shrinktech` to compute all simulator sets of the shrunk timed automaton, if the given timed automaton is shrinkable. The

shrunk timed automata will also be written in a local directory 'shrunk' in case of shrinkability. This option is disabled by default since it is not always needed.

`--shrink` This option is used to syntactically shrink a given timed automaton. When called `shrinktech --shrink automaton.tg 100`, the program will multiply all constants by 100 and shrink the guards by 1. This is equivalent to shrinking by 1/100; but the time scale is changed so that all constants are integers. By default, the file is output as `automaton-shrunk.tg`, but another target file can be specified with `-out automaton2.tg`.

`--gentrace` Generates a trace, in form of a finite automaton, by a random simulation of the given timed automaton. When called with `shrinktech --gentrace automaton.tg 1000`, this option will output in standard output a random trace of length 1000. One can then either direct this to a file, or specify an output file by `-out automaton.aut`.

`graph_extractor` This is a Python script that can extract a subgraph of a given graph, using DFS or BFS, either deterministically or randomly. The options `-dfs`, `-randdfs` and `-bfs` can be used for this purpose. Run the program without arguments for usage.

Note that CADP can also be useful for instance to check bisimulation, minimize or visualize finite automata (`http://www.inrialpes.fr/vasy/cadp/`).

C Presentation

The tool demonstration will start by presenting timed automata and robustness issues with its semantics. We will explain how behaviours disappear under shrinking by an example similar to one in Section A. The presentation will then be based on the following scheduling example in order to illustrate the purpose of shrinkability checking. Consider the model in Fig. 9. This timed automaton models a specification for a system in which an event is generated at least every 2 time units (the component on the left), and at most an event is consumed every 2 time units (the component in the middle). The communication is ensured by a bounded FIFO channel, modelled as a finite aumaton (on the right). Let us consider the instantiation of this system for $N = 1$.

Can we refine this system so that no error occurs (*i.e.* the location `err` is never visited)? The answer is yes: one can replace both inequalities $x \leq 2$ and $y \geq 2$ by equalities. One can then show that location `err` is not reachable in the resulting system (for $N = 1$).

How about robustness? Intuitively, in order to obtain the above refinement, we need to take all transitions "at the last moment", that is, at the boundaries of the guards. This may not be a desirable refinement due to robustness concerns. Shrinkability analysis shows that there is no refinement that avoids the boundaries of the guards. In fact, the timed automaton of Fig. 9 can time-abstract simulate the timed automaton of Fig. 6. But we already saw that the latter is not shrinkable.

In our presentation, we will also give a short summary of previous work on different notions of robustness, and the approaches that were adopted to check robustness in case studies in the literature.

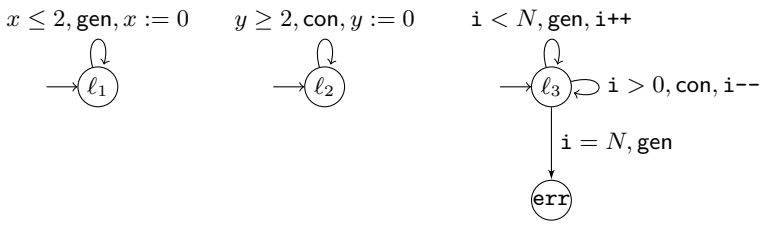

Fig. 9. A timed automaton modelling a producer-consumer system, given as a network of timed automata with a bounded integer variable `i` which is initially 0 (Our definition of timed automata and the Kronos timed automata format do not contain discrete variables. But these can be easily encoded by a finite automaton. See for instance, the Lip Sync case study).