Formal verification - Tutorial 06

23/03/2026

Timed automata

Exercise 1. *Show that timed regular languages are effectively closed under*

1. *union,*
2. *intersection,*
3. *projection.*

Solution. For union and intersection, we can perform standard product constructions. For clocks we take the disjoin union of clocks. For projection, just project the alphabet. □

Exercise 2 (Diagonal constraints)**.** *A* diagonal constraint *is a constraint of the form*

$$x - y \sim c,$$

for some clock variables $x, y \in X$, comparison operator $\sim \in \{<, \leqslant, =, \geqslant, >\}$ and integer constant $c \in \mathbb{Z}$. Show that adding diagonal constraints to the syntax of timed automata does not increase their expressiveness (as recognisers of timed languages).

Solution. The main intuition is that $x - y$ is invariant under time elapse. Thus we can store in the state whether $x - y \sim c$ holds or not. When y is reset, we need to know whether $x \sim c$ holds in order to update the state. We can add additional such tests to the guard of the transitions. When x is reset, the reasoning is similar. The construction is exponential in the number of diagonal constraints to be removed. □

Exercise 3. *Fix a set of clocks $X = \{x_1, \ldots, x_k\}$ and a maximal constant $M \in \mathbb{N}$. Find an upper bound on the number of regions.*

Solution. A region is completely determined by:

1. The integral part of each clock up to M, or unbounded. There are $M + 2$ possibilities for each clock, thus $(M + 2)^k$ possibilities in total.
2. The total order of the fractional parts of the clocks, of which there are $k!$ possibilities.
3. For each clock, whether its fractional part is zero or not, which gives 2^k possibilities.

This gives the upper bound

$$(M+2)^k \cdot k! \cdot 2^k. \qquad \square$$

Exercise 4. *Show that the nonemptiness problem for timed automata is PSPACE-hard.*

Solution. We reduce from the nonemptiness problem for the intersection of k regular languages, which is PSPACE-hard (in fact PSPACE-complete). The idea is that a discrete-time clock can store the current state of a finite automaton. State updates are simulated by suitable time elapses and clock resets.

There are many ways to achieve this. To fix ideas, consider finite automata $A_1, \ldots, A_k$ with pairwise disjoint sets of states $Q_1, \ldots, Q_k$, with a common set of states $Q = Q_1 \cup \cdots \cup Q_k = \{q_1, \ldots, q_m\}$. We use clocks $x_1, \ldots, x_m$ to store the current state of the finite automata. The idea is that x_i is zero if the relevant automaton is not in q_i and that x_i is one if the relevant automaton is in q_i. We need to efficiently update this information. More generally, we show how we can set x_i to a new value in $\{0, 1\}$, while keeping the other clocks unchanged. Setting x_i to zero is easy: Just reset it. Setting x_i to one (while keeping the other clocks unchanged) is more tricky. We show how to do this with an example. Suppose we have three clocks x_1, x_2, x_3, starting at $(0, 0, 1)$, and we want to set x_1 to one while keeping x_2 and x_3 unchanged, as to reach $(1, 0, 1)$. More generally, we want to go from $(0, x_2, x_3)$ to $(1, x_2, x_3)$.

	I	II	III	IV	V
x_1	0	1	0	1	1
x_2	0	1	1	2	0
x_3	1	2	0	1	1

In the first step, we elapse one time unit. In the second step, we reset x_1 and all clocks with value 2. In the third step, we elapse one time unit. In the fourth step, we reset all clocks with value 2.

We can also implement a similar simulation using dense-time clocks, by encoding the state of the finite automaton in the total ordering of the fractional part of the clock values. $\square$

Exercise 5. *Consider timed automata over a single clock $X = \{x\}$. What is the computational complexity of the nonemptiness problem?*

Solution. Regions for a single clock are considerably simpler. In fact, it suffices to keep track of the following information:

1. The integral part of the clock up to M, or unbounded.
2. Whether the fractional part of the clock is zero or not.

When clock constraints are encoded in binary, M is still exponential in the size of the input. However, let $c_1 < \cdots < c_k$ be the pairwise distinct constants appearing in the input. It suffices just to keep track of whether the integral part of the clock is equal to c_i for some i, or whether it is between c_i and c_{i+1} for some i, or whether it is greater than c_k. This gives a polynomial number of regions. Thus the nonemptiness problem for timed automata with a single clock is in NL. Reachability is NL-hard already for finite graphs, so altogether the problem is NL-complete. $\square$

Exercise 6. *What is the computational complexity of the universality problem for deterministic timed automata?*

Solution. Deterministic timed automata are efficiently closed under complementation, so the universality problem reduces to the nonemptiness problem in PTIME, and thus it is in PSPACE. The problem is also PSPACE-hard, since the nonemptiness problem for nondeterministic timed automata is PSPACE-hard already for deterministic timed automata, and the latter reduces to the universality problem via complementation. □