University of Warsaw, PhD Open, Exam

Analysis of Systems with Infinite-State Spaces

Javier Esparza, Technical University of Munich
esparza@in.tum.de

1. Show that the reachability problem for timed automata is NP-hard.

Given: A timed automaton A, a control state q.
Decide: Is some configuration of A of the form $\langle q, \mathbf{t}\rangle$ reachable from the initial configuration?

You get extra points if your reduction assumes that the constants of the timed automaton must be given in unary; in other words, if you assume that the size of the constant c is c.

Remark: The problem is PSPACE-complete. A proof can be found in Section 4.5 of Alur and Dill's paper, accessible online at
`http://www.cs.aau.dk/ srba/courses/MCS-07/TA.pdf`
You can answer by "copying" Alur and Dill's proof, but then you have to fill out the details omitted in the paper. Directly finding a reduction from some NP-complete problem, say SAT, is easier.

2. This question is about Petri nets. The configurations of a Petri net are usually called *markings*. A marking is a vector of natural numbers, one for each place of the net. The i-th component of the vector indicates the number of tokens in the i-th place. An execution of the net is a (finite or infinite) sequence of markings, each one reachable from the previous one through the firing of a transition.
Show that the following problem is decidable.

Given: A Petri net N with initial marking M_0.
Decide: Does N have an infinite execution from M_0?

Remark: If you never heard about Petri nets before, go to
`http://en.wikipedia.org/wiki/Petri_net`
and read the sections "Syntax" and "Execution semantics".

Hint: Use Dickson's lemma.

3. In the course we have seen in detail an algorithm that, given an NFA recognizing a set C of configurations of a pushdown automaton P, constructs another NFA for the set $pre^*(C)$ of predecessors of C. A description of the algorithm can also be found in Section 4 of

`http://www7.in.tum.de/um/bibdb/esparza/cav00.pdf`

Section 6 presents a very similar algorithm for the set $post^*(C)$ of successors of C. In this question we call these algorithms the *pre**- and *post**-algorithms for pushdown automata.

(a). A *monadic rewrite system* over a finite alphabet Σ is a set R of rules of the form $a \to w$, where $a \in \Sigma$ and $w \in \Sigma^*$. Given $w_1, w_2 \in \Sigma^*$, we say that w_2 is an *immediate successor* of w_1 if there is a rule $a \to w$ in R and $u, v \in \Sigma^*$ such that $w_1 = uav$ and $w_2 = uwv$. As usual, the successor relation between words of Σ^* is defined as the reflexive and transitive closure of the immediate successor relation.
Give a *pre**-algorithm for monadic rewrite systems by suitably modifying the *pre**-algorithm for pushdown automata.

(b). Give a monadic rewrite system R and a word w such that $post^*(\{w\})$ is *not* a regular language. This shows that the *post**-algorithm for pushdown automata cannot be extended to monadic rewrite systems. Explain why.

(c). The *pre**- and *post**-algorithms for pushdown automata are polynomial both in the size of the pushdown automaton and in the size of the NFA. In this question we see that the use of NFAs as data structure to represent sets of configurations is crucial for polynomiality.
Show that there is no polynomial algorithm that, given a pushdown automaton and a DFA recognizing a set of configurations C, returns another DFA recognizing $pre^*(C)$.
Hint: Show that the minimal DFA for $pre^*(C)$ has exponential size.