Advanced Automata Theory
Solutions for Exercise Sheet 6

Emanuele D'Osualdo
Sebastian Schweizer
TU Kaiserslautern
Summer term 2016

Exercise 2: Naive Interpretation of NFAs as NBAs
Let $A = (\Sigma, Q, q_0, \rightarrow, Q_F)$ be an NFA with $\emptyset \neq L(A) \subseteq \Sigma^+$ and, for any two states $q, q' \in Q$, define $L^{\neq \epsilon}_{q,q'} := \{w \in \Sigma^+ \mid q \xrightarrow{w} q' \text{ in } A\}$. If $L_\omega(A)$ is the ω-regular language accepted by A (interpreted as an NBA), one can **wrongly** believe that $L_\omega(A) = L(A)^\omega$.

c) Given an NFA A, provide a construction for an NBA A_ω such that $L(A_\omega) = L(A)^\omega$.

Solution:
We want to modify the automaton such that it accepts a word from $L(A)$ and then goes back to the start to accept the next one. Each iteration of this loop should see exactly one final state. Then an accepting run for the NBA will contain infinitely many of these loop iterations, i.e. accepts the concatenation of infinitely many words from $L(A)$.

Construction. Consider an NFA $A = (\Sigma, Q, q_0, Q_f, \delta)$. We construct an NBA $A_\omega = (\Sigma, Q', q'_0, Q'_f, \delta')$ that accepts $L(A)^\omega$. We first need to make sure that the initial state does not have any incoming transitions except for our loop. We therefore add a new state q'_0 and set $Q' = Q \cup \{q'_0\}$. The transition relation δ' contains all the transitions from δ plus the following added rules:

- $\delta'(q'_0, a) = \delta(q_0, a)$ for all $a \in \Sigma$
  Note: This is equivalent to adding an ϵ transition from q'_0 to q_0.
- $\delta'(q, a) \ni q'_0$ if $\delta(q, a) \cap Q_f \neq \emptyset$.
  Note: This is equivalent to adding an ϵ transition from q_f to q'_0 for every $q_f \in Q_f$ and represents the loop we mentioned before.

The only final state shall be our added initial state: $Q'_f := \{q'_0\}$.
Figure 1 shows an example for the construction.

Proof.

$L(A\omega) \subseteq L(A)^\omega$: Consider a word $x \in L(A_\omega)$. There is an accepting run in A_ω that contains infinitely many occurrences of the only final state q'_0:

$$q'_0 \xrightarrow{x_1} q'_0 \xrightarrow{x_2} q'_0 \xrightarrow{x_3} \ldots \quad \text{with } x = x_1 \cdot x_2 \cdot x_3 \ldots \text{ and } x_i \neq \epsilon$$

Chose the x_i such that the automaton does not visit q'_0 in between. We show that $x_i \in L(A)$, i.e. there is a run in A with $q_0 \xrightarrow{x_i} q_f \in Q_f$. This is sufficient to show $x \in L(A)^\omega$.

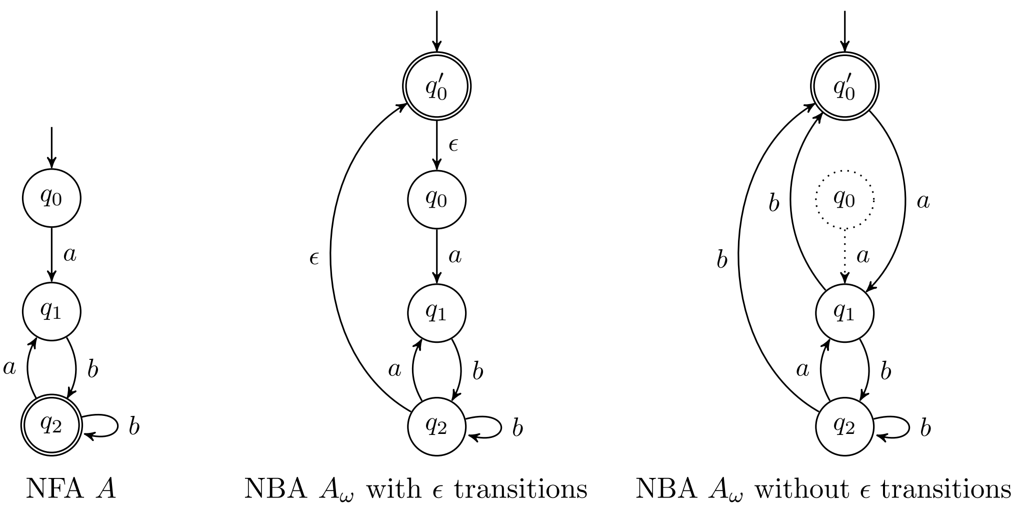

Figure 1: Example construction for the language $(ab^*)^+$

Split $x_i = a \cdot y$ with $a \in \Sigma$ and $y \in \Sigma^*$. If y is empty, then we have $\delta'(q_0', a) \ni q_0'$. Due to our construction, we also have $\delta(q_0, a) \cap Q_f \neq \emptyset$ such that A accepts $x_i = a$.

If y is not empty, then we can split $y = z \cdot b$ with $z \in \Sigma^*$ and $b \in \Sigma$. We find states $q_1, q_2 \in Q$ with $\delta'(q_0', a) \ni q_1$, $\delta'^*(q_1, z) \ni q_2$ and $\delta'(q_2, b) \ni q_0'$. Due to our construction, we immediately know that $\delta(q_0, a) \ni q_1$ and $\delta'(q_2, b) \cap Q_f \neq \emptyset$. But as we do not visit q_0' in between, we also have $\delta^*(q_1, z) \ni q_2$. This completes the required run in A.

$L(A\omega) \supseteq L(A)^\omega$: Consider a word $x \in L(A)^\omega$. Then $x = x_1 \cdot x_2 \cdot x_3 \ldots$ with $\epsilon \neq x_i \in L(A)$. For every x_i there is a run in A from q_0 to a final state $q_f \in Q_f$, i.e. $\delta^+(q_0, x_i) \ni q_f$. Split $x_i = y \cdot a$ with $y \in \Sigma^*$ and $a \in \Sigma$. We find a state $q \in Q$ with $\delta^*(q_0, y) \ni q$ and $\delta(q, a) \ni q_f$. Due to our construction, we also have $\delta^*(q_0', y) \ni q$ and $\delta(q, a) \ni q_0'$. This yields a run in A_ω as $q_0' \xrightarrow{x_i} q_0'$.
We can combine these runs to a run for x in A_ω:

$$q_0' \xrightarrow{x_1} q_0' \xrightarrow{x_2} q_0' \xrightarrow{x_3} \ldots$$

Since we visit the final state q_0' infinitely often, this proves that $x \in L(A_\omega)$.

□

Exercise 5: Reachability in Counter Machines (Optional)
Adapting Parikh's proof, show that reachability in counter machines with one unrestricted counter and n r-reversal bounded counters is decidable.

Solution:
Let $M = (\Sigma, Q, q_0, \rightarrow)$ be a counter machine with counters $C = \{c_0, c_1, \ldots, c_n\}$ and $\Sigma = \{\mathit{inc}, \mathit{dec}, \mathit{zero}\} \times C$ where c_0 is unrestricted and $c_1, \ldots, c_n$ are r-reversal bounded. Let $q \in Q$ be a state that we want to check for reachability. The overall plan is as follows:

1. Using the algorithm from the lecture, we construct an equivalent counter machine M' with one unrestricted and $n * \left(1 + \left\lfloor \frac{r}{2} \right\rfloor\right)$ one-reversal bounded counters.
2. Using the approach from exercise 1a, we construct a one counter machine M'' over Σ. The state space and transitions ensure that for each reversal bounded counter there is only one increment phase followed by a decrement phase. For c_0 we use the machine's unrestricted counter.
3. We show that for every one counter machine the set of reachable vectors $R(q)$ is semi-linear.
4. As in exercise 1a, we construct a semi-linear set S_{SL} that contains all vectors with valid numbers of increments, decrements and zero tests for each reversal bounded counter. We intersect S_{SL} with the semi-linear set $R(q)$. The resulting set is semi-linear and therefore reachability of q decidable.

Similarly to Parikh's proof, point 2 extracts a machine that generates a language of possible counter actions. The words of this language represent possible sequences of actions on all the counters; the actions on the reversal-bounded counters respect the 1-reversal bound, but may do more decreases than increases. This is corrected by representing the reachable configurations as a Presburger formula, which is restricted to natural numbers. The words in the language will however respect the actions on the unrestricted counter because they correspond to actual effects on the counter of M'', which can make sure we only increment, decrement and test for zero when it is legal to do so.

The difficult part is point 3. There are two possible approaches.

Alternative 1: Reduction to Parikh's theorem via CF grammars

One can interpret the one-counter machine as a pushdown automaton which generates words representing the possible actions on all the counters. A single special stack symbol $\#$ denotes the bottom of the stack. Counter value n can be represented with a stack containing n symbols A on top of $\#$. Increment and decrement will add respectively remove an A, the zero test will check that the top of the stack is $\#$. Then with the standard triplet construction the pushdown automaton can be transformed into a context-free grammar G generating the same language. The proof is then a direct consequence of Parikh's theorem applied to G.

Alternative 2: Adaptation of Parikh's proof

The idea for an alternative, more direct, proof is as follows: we modify Parikh's proof to fit one counter machines. A run in a one counter machine is a sequence of transitions. A configuration $c = (q, n)$ contains a state $q \in Q$ and the counter value $n \in \mathbb{N}$.

We can split a run into two phases. In the first phase, a pump shall induce a sequence $(q, n) \cdot \tau \cdot (q, n)$. For the second phase, we relax the condition for the counter value and a pump should induce a sequence $(q, n) \cdot \tau \cdot (q, n')$ with $n \leq n'$. The idea is that in the first phase pumps will restore the counter value. This ensures that a pump does not affect later *zero* tests. In the second phase the counter value can increase and therefore it must not contain any *zero* tests.

A pump is a triple (min, max, α) where

- $min \in \mathbb{N}$ is the minimal counter value that allows to apply the pump
- $max \in \mathbb{N} \cup \{\infty\}$ with $max \geq min$ is the maximal counter value that allows to apply the pump and
- $\alpha \in (Q \times \{inc, dec, zero\} \times C')^*$ is a sequence of states and actions on all counters of M'.

The first and last state of a pump must be the same. Applying the pump induces a sequence of a run where for each step the counter is adjusted accordingly. The minimal value must be chosen such that all decrements in the pump can be applied without dropping the counter to a negative value. The maximal value must be chosen such that the counter reaches 0 for *zero* tests. Without *zero* tests, the maximal value will be ∞.

For the first phase of the run, pumps need to contain an equal number of increment and decrement operations. For the second phase, pumps can contain more increments than decrements. The maximal value for pumps in the second phase is always ∞.

We define $\lhd$ as the smallest reflexive transitive closure containing the following:

- $(min_1, max, \alpha) \lhd (min_2, max, \alpha)$ where $min_1 \leq min_2$
- $(min, max_1, \alpha) \lhd (min, max_2, \alpha)$ where $max_1 \geq max_2$
- $(min_1, max_1, \alpha) \lhd (min_2, max_2, \beta \cdot \alpha \cdot \gamma)$ where $min_2 + inc(\beta) - dec(\beta) \leq min_1$ and $max_2 + inc(\beta) - dec(\beta) \geq max_1$ and $\beta \neq \epsilon \vee \gamma \neq \gamma$.

A basic pump is $\lhd$ minimal. We argue that our definition of $\lhd$ ensures that there are only finitely many basic pumps. This allows to complete the proof.

Assume there are infinitely many basic pumps. Then there is a state $q \in Q$ for which there are infinitely many basic pumps that begin and end with q. But since there are only finitely many states, these basic pumps must contain a repeating pattern that can be generated by other pumps. This is a contradiction. For a detailed proof, we need to handle several cases for both phases separately.

The rest of the proof follows the same scheme as Parikh's. $\square$