INF2080
Oblig 1

Elsie Mestl

February 18, 2016

Problem 1: Regular languages

Let A and B be regular languages defined by DFAs $\mathcal{A}$ and $\mathcal{B}$. Let $n_{\mathcal{A}}$ and $n_{\mathcal{B}}$ be the number of states in $\mathcal{A}$ and $\mathcal{B}$ respectively.

Problem 1a

$\boldsymbol{A \cap B}$: Since the DFA $\mathcal{A}$ has $n_{\mathcal{A}}$ states and the DFA $\mathcal{B}$ has $n_{\mathcal{B}}$ states, the states in the intersection of the two DFAs are elements in $\mathcal{A} \times \mathcal{B}$ and therfore have at most $|\mathcal{A} \times \mathcal{B}|$ states, which is equivalent to $n_{\mathcal{A}} \cdot n_{\mathcal{B}}$ states.

$\boldsymbol{A^*}$: The atomaton that accepts the language A has $n_{\mathcal{A}}$ states. But for A^* the DFA also needs to accept the empty string and any number of concatinations. In an NFA this wound have been solved by adding a new start state that accepts the empty string and has an epsilon transaction to the old start state, and adding epsilon-transactions from every accepting state to the new start state. Assumming we do that, we now have a NFA that accepts A^* and we know there exists an DFA that accepts the same language as the NFA. We also know that an DFA created from a NFA has at most $\mathcal{P}(A^*) = 2^{n_{\mathcal{A}}+1}$ states.

Problem 1b

Assume A, B are the same languages as before, and they can be represented by the DFAs $\mathcal{A}$ and $\mathcal{B}$, repectively.
Then the largest nr. of states needed to represent $A \cap B$, AB and A^* are:

$\boldsymbol{A \cap B}$:

Nr. of states for	$\lvert A \cup B \rvert = n_{\mathcal{A}} + n_{\mathcal{B}} + 1$[1]
And we know	$A \cap B = \overline{\overline{A \cap B}} = \overline{\neg A \cup \neg B}$

The negation is negating which states in the DFAs whom are accepting states. The total nr. of sates stays the same. Therfore we can see the total number of states are $|A \cap B| = |A \cup B| = n_{\mathcal{A}} + n_{\mathcal{B}} + 1$.

$\boldsymbol{AB}$: Let $\mathcal{M}$ be the DFA that accepts the language AB. The start state in $\mathcal{M}$ equals the start sate in $\mathcal{A}$. Futhermore $\mathcal{M}$ works in the way that every accepting state in the original $\mathcal{A}$ has an epsilon transaction to the start state in $\mathcal{B}$. But $\mathcal{M}$ only has the same accepting states as $\mathcal{B}$. This way we have not added any extra nodes, only extra transactions between states already existing in the original state machines. And the largest number of states in AB is $n_{\mathcal{A}} + n_{\mathcal{B}}$.

$\boldsymbol{A^*}$: This problem is already answered in 1a, but here is a more detailed description:
The number of states in an atomaton, $\mathcal{M}$, that accepts the language A^* is one more than for the atomaton $\mathcal{A}$. The extra state added is a new accepting start state that has epsilontransactions to the old start state. To make $\mathcal{A}$ into $\mathcal{M}$, add, as described, a new start state. In addition, add an epsilontransaction(s) from the accepting state(s) to the new start state. Make the old accepting states non-accepting.[2]

Problem 1c

To find the regular expression that describes the language the DFA accepts, we have to make the DFA into a GNFA. The start state can only have outgoing edges and the accepting state can only have incomming edges. All the states also needs to be connected by a transaction.

[1] The pluss one comes form the new start state wich is added to the new sate machine. Where the transactions from this state to the next are epsilon transitions to the old start states form the original atomaton.

[2] This last part is not strictly speaking necessary, but because of the epsilon transactions they are no longer needed as accepting.

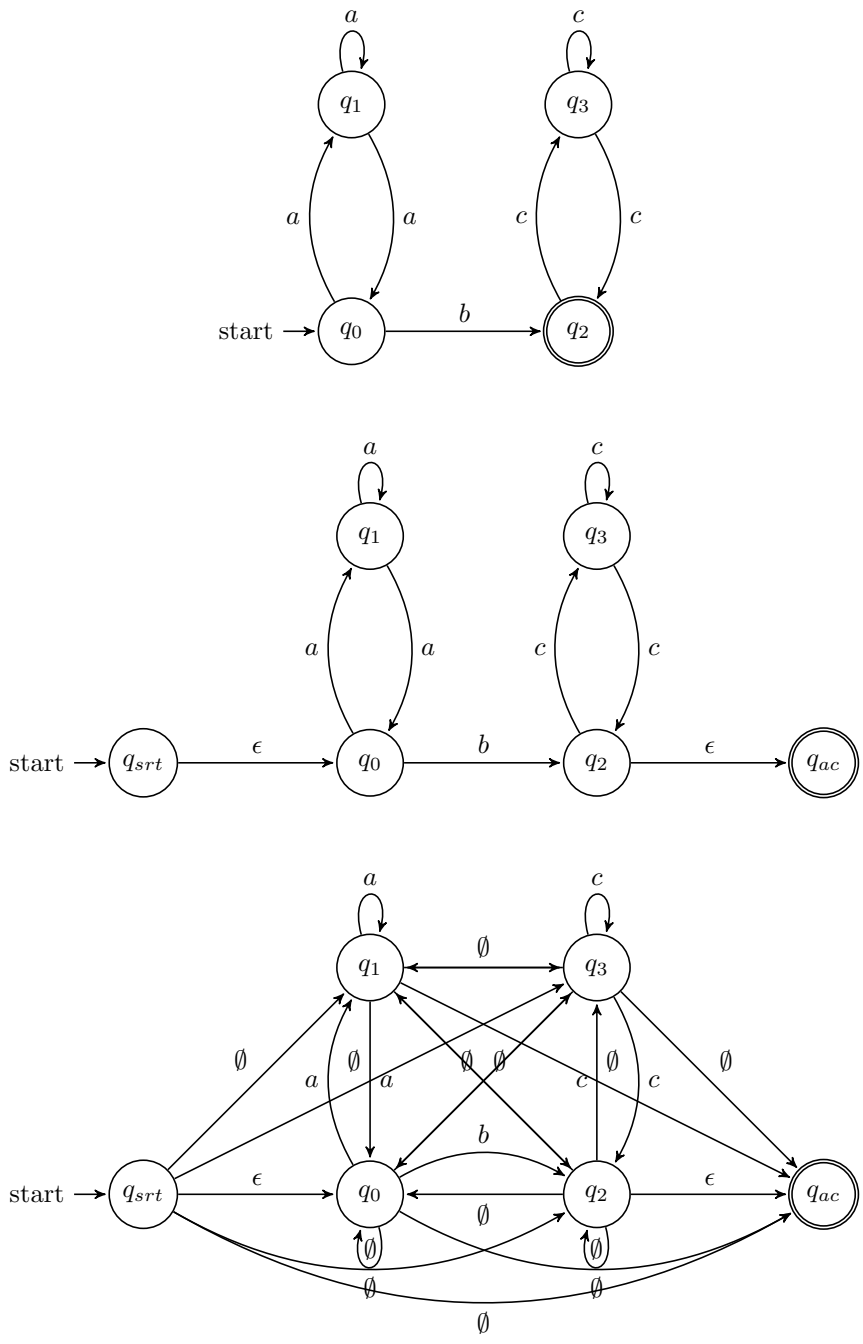
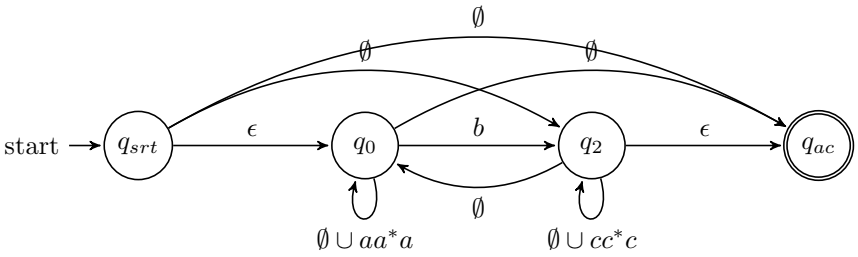
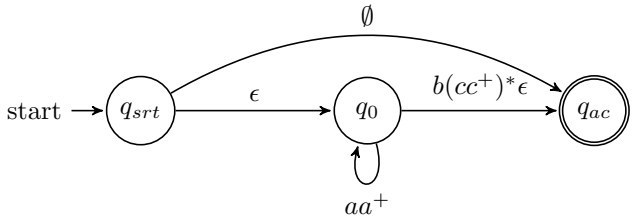


Figure 1: Three-step guide to constructing the GNFA from the DFA

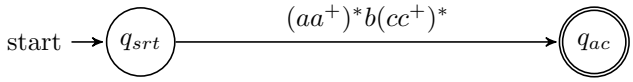
Now as that is done, we match triangular figures [3] and remove edges.
Look at the edges going between $q0$ and $q1$ where $q0$ is both a and c to reference the above. And since all the other edges are empty transactions, theese do not have to be put into account. Do the same with $q2$ and $q3$, and get the following atomaton.



Now removing $q2$ by the same method



At last removing $q0$ and getting



And there we have it, $(aa^+)^*b(cc^+)^*$, the regular expressing defining the language that the DFA accepts.

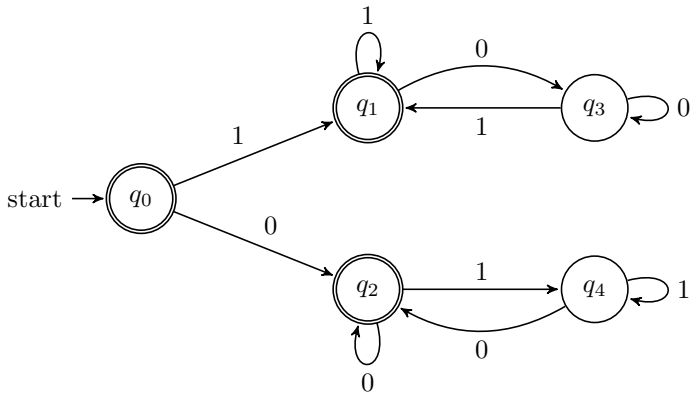
Problem 1d

$$\{w \mid w \text{ contains equally many occurences of the substrings 01 and 10}\}.$$

Assume that this means that 010 is accepted expression since both 01 and 10 are substrings of 010. What this really means is that there is an equal number of changes from 0 to 1 and from 1 to 0. We can easily see that one can not go more than one ahead in change. Since the change $0 \to 1$ is not possible more than one time before the change $1 \to 0$ has to be done to performe the $0 \to 1$

[3] Where node a is connected to c and b, and b is connected to c. We can in such a case remove node b and extend the expresion between a and c with the union of every that earlier was on the path a-¿b-¿c.

change again. One can also think the automaton accepts all strings that start and end with the same symbol.
The DFA will also accept the empty string since the occurance of 01 and 10 both are 0, and thereby equal.



Problem 2: all-NFAs

Assume that you have an all-NFA G that you want to convert to a DFA $D = (Q_D, \Sigma, \delta_D, q_{1_D}, F_D)$.

Let $G' = (Q, \Sigma, \delta, q_1, F')$ be an NFA with the same properties as G, except for F, the finish state. The difference between G an G' is that G accepts only those inputs where all the computational branches are at an accept state, whereas G' accepts as long as at least one is.

Since G' is an NFA we know we can convert G' to a DFA $D' = (Q_{D'}, \Sigma, \delta_{D'}, q_{1_{D'}}, F_{D'})$ where D' has the following properties:

$$
\begin{aligned}
Q_{D'} &= \mathcal{P}(Q) \\
\delta_{D'}(R, a) &= \{q \in Q | q \in \delta(r, a) \text{ for some } r \in R\} \qquad R \in Q_{D'} \\
q_{0_{D'}} &= \{q_0\} \\
F_{D'} &= \{R \in Q_{D'} | R \text{ contains an accept state } N\}
\end{aligned}
$$

Figure 2: Credit to book

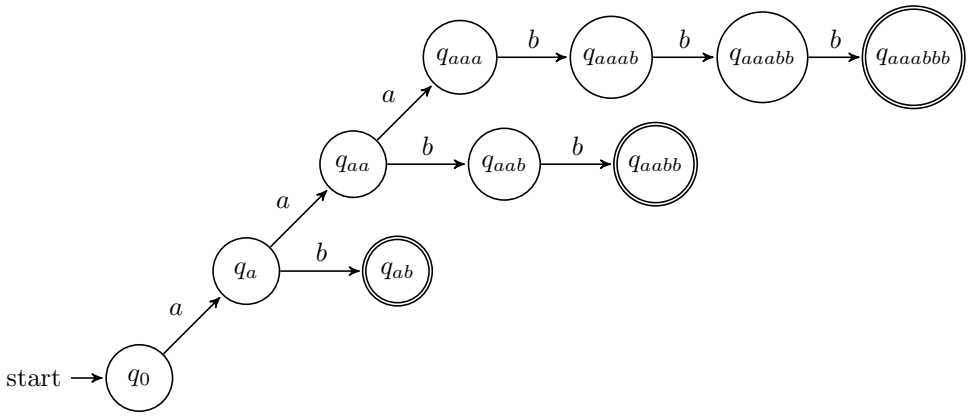
To make D' into D, we want to remove all states that have lead to at least one branch dying out, and the connecting transactions to this state.

1. Let D = D', and we will start removing posibillites from D.

2. For every state $q \in Q_D$ and $p \in Q_D$, given $a \in \Sigma$, $\delta_{D_{q,p}}(q, a) = p$ then for every $q' \in q$ look at all leagal transitions given a. If any of them leads to a termenating branch, make q a sink state and remove the transition $\delta_{D_{q,p}}(q, a) = p$.
3. Remove any state but the start state that has no incomming transactions.
4. Repeat step 2 and 3 until done.
5. For all states $f \in F$, if there is an $f' \in f$ that is not an accept state, make f non-accepting as well.

Problem 3: Non-regular languages

Problem 3a



defines the language $\{ab, aabb, aaabbb\}$.

Problem 3b

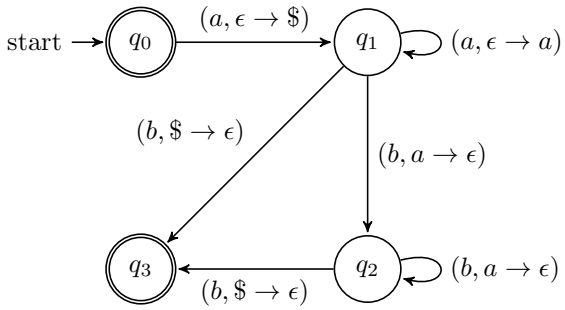
I assume that by determenistic infinite automaton, a determenistic pushdown automaton suffice, since the stack in such have an infinite storage amount.

$$\{a^n b^n \mid n \in \mathbb{N}\}.$$

Let the PDA be described as follows:

$$
\begin{aligned}
Q &= \{q_0, q_1, q_2, q_3\} \\
\Sigma &= \{a, b\} \\
\Gamma &= \{\$, a\} \\
q_0 &= q_0 \\
Z^{[4]} &= \epsilon \\
F &= \{q_0, q_3\}
\end{aligned}
$$

Let $\delta : (Q \times \Sigma \times \Gamma) \to (Q \times \Gamma)$ be defined as the transaction function visible in the representation below.



One might at first glance think this automaton is non-determenistic because of the multiple transactions for b both at q_1 and q_2. But with a closer look we see that pop-requests to the stack are all uniqe given an inputsymbol, and no "choice" is possible anywhere in the automaton, i.e. the PDA is also an DPDA.

And there you have it; an determenistic infinite automaton that accepts the language $\{a^n b^n | n \in \mathbb{N}\}$ [5]

Problem 3c

The pumpinglemma states that any string longer than the pumping length can be pumped. I.e. this means that the automaton contains at least one loop.

Given the language $\{a^n b^n | n \in \mathbb{N}\}$ prove that at least one string in this language cannot be pumped.

Proof. Asssume $\{a^n b^n | n \in \mathbb{N}\}$ is a regular language. Let the pumping length be of length p. Lets look at the string; $a^p b^p$, which has a length $2p$. Since $2p > p$,

[4] Initial state in the stack

[5] Strictly speaking it is not a perfect DPDA since not all possible inputs at any state are acounted for, forr exsample a in q_2. All transactions that are not drawn all go to a sick state. But due to readability I have chosen to leave this out of the oblig.

this string should, by the pumpinglemma, be pumpable. We will, however, show that it cannot be pumped, and thereby proove that the language is irragular. Since the string $a^p b^p$ can be pumped, it can be devided into 3 parts $xy^i z$ where y is the part that gets pumped. We also know that $|y| > 0$ and $|xy| \leq p$. With this in mind we can look at all possible ways $a^p b^p$ can be divided.

Case 1: y = some number of a's.
Since y can be pumped, the number of a's increase, but the number of b's stay the same. The number of a's is greater than the number of b's after pumping. Case 1 is not pumpable.
Case 2: y = some number of a's and b's.
Since $|xy| \leq p$ and $|a^p| = p$ then this is not a valid selection of y.
Case 3: y = some number of b's.
This is only a more extreme version of Case 2 and is no leagal selection of y.

Since we have found one occurance of a string in the language $\{a^n b^n | n \in \mathbb{N}\}$ with length longer than the pumping length, but cannot be pumped, we have prooven that the language is non-regular.

□

Problem 3d

Show that $\{a^n b^n | n \in \mathbb{N}\}$ is context-free.

Proof. A language is context free iff there exists a PDA that accepts the same language. And since we in Problem 3b found such a PDA (DPDA) we know that the language $\{a^n b^n | n \in \mathbb{N}\}$ is context-free. □