Mälardalen University
School of Innovation, Design and Engineering

TEN1 DVA337 2015

Formal languages, automata, and theory of computation

Thursday, November 5, 14:10 - 18:30
Teacher: Daniel Hedin, phone 021-107052

The exam has a total of 40 points and consists of 3 pages. No aids are allowed. Answers must be given in English and should be clearly justified.

1. Regular languages (14 p)

a) Floating point numbers are built by an optional sign followed by digits separated by a decimal point and end with an optional exponent part. Examples of floating point numbers are

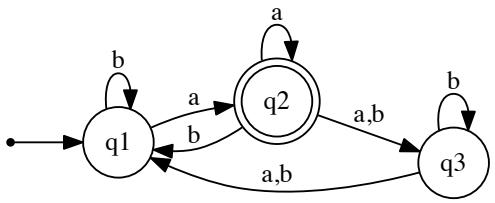
```
1.35e+10  +2.54  -5.2  245.3E-2
```

Write a regular expression that recognizes floating point numbers. (2 p)

Solution: `(-|+)?[0-9]+.[0-9]+( (e|E)(+|-)[0-9]+ )?`

---

b) Convert the following NFA to an equivalent minimal DFA. (6 p)



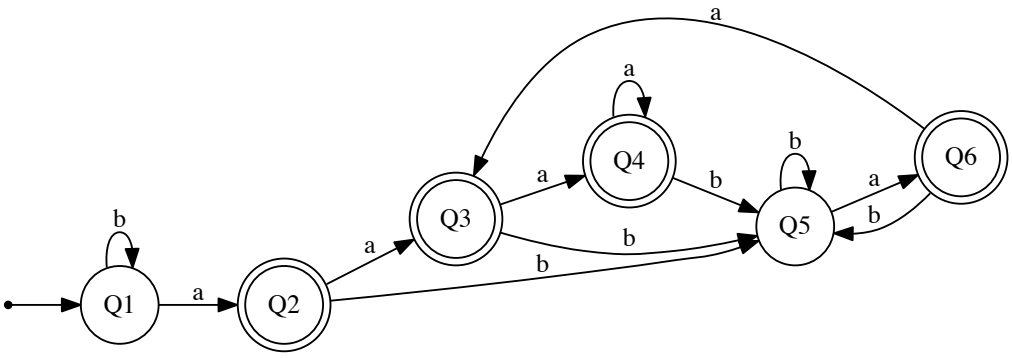
Solution: We first convert the NFA to a DFA using the subset construction algorithm. For book keeping we state the transition function δ

q_1	
a	$\{q_2\}$
b	$\{q_1\}$

q_2	
a	$\{q_2, q_3\}$
b	$\{q_1, q_3\}$

q_3	
a	$\{q_1\}$
b	$\{q_1, q_3\}$

Starting in state $\{q_1\}$ (the set of λ-reachable states from the starting state of the NFA), we can systematically construct the following DFA, where $Q_1 = \{q_1\}$, $Q_2 = \{q_2\}$, $Q_3 = \{q_2, q_3\}$, $Q_4 = \{q_1, q_2, q_3\}$, $Q_5 = \{q_1, q_3\}$, $Q_6 = \{q_1, q_2\}$ with Q_2, Q_3, Q_4 and Q_5 are final states, since they contain the at least one final state from the NFA.

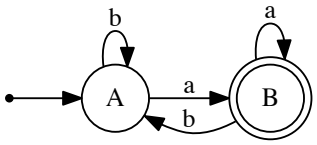


To minimize this DFA we apply state partitioning starting with two sets of states $A = \{Q_1, Q_5\}$ and $B = \{Q_2, Q_3, Q_4, Q_6\}$ corresponding to the set of non-final and the set of final states respectively and see that the transitions of the original DFA do not partition these sets further.

A	Q_1	Q_5
a	B	B
b	A	A

B	Q_2	Q_3	Q_4	Q_6
a	B	B	B	B
b	A	A	A	A

Based on this we get the following mininal DFA.



---

c) State the pumping lemma for regular languages and use it to prove that $L = \{a^n b^n \mid n \geq 0\}$ is not regular. (4 p)

Solution: The pumping lemma for regular languages states that for all regular languages L, there exists a positive number m such that for all strings $w \in L$ with $|w| \geq m$ it holds that the exists a decomposition of $w = xyz$ subject to $|xy| \leq m$ and $|y| \geq 1$ for which it holds that $xy^i z \in L$ for any $i \geq 0$.

To prove that $L = \{a^n b^n \mid n \geq 0\}$ is not regular we proceed with a proof by contradiction. Assume that L is regular, i.e., that the pumping lemma holds for L and show how to construct a string w in terms of m chosen so that we can show that for any decomposition $w = xyz$ subject to $|xy| \leq m$ and $|y| \geq 1$ there is at least one $i \geq 0$ for which $xy^i z \notin L$. This shows that the pumping lemma does not hold for L and we have reached a contradiction.

In this particular problem we can chose $w = a^m b^m$. Due to $|xy| \leq m$ we have that $x = a^j$, $y = a^k$, for $k \geq 1$, and $z = a^l b^m$ for $j + k + l = m$. According to the pumping lemma if L is regular then for every $i \geq 0$ it should hold that $xy^i z \in L$. But for any $i \neq 1$ we can show that $xy^i z \notin L$. Consider for instance $i = 0$ then $xy^i z = xy = a^{j+l} b^m$, but

we have that $j+l \neq m$, since $k \geq 1$, and we can draw the conclusion that $a^{j+l}b^m \notin L$. Hence, L is not regular. □

---

d) Consider the problem of nested C-like comments. A comment is well-formed if the number of starting symbols `/*` matches the number of ending symbols `*/` and in any given prefix the number of starting symbols is greater than or equal to the number of ending symbols. To illustrate consider the following examples of well-formed nested comments

```
/* Well-formed comment */
/* Also /* a well-formed nested comment */ !! */
```

and the following examples of malformed nested comments

```
/* Not a */ well-formed comment */
/* Also not /* a well-formed comment  */
```

Give a convincing argument that it is not possible to create a DFA that accepts precisely the well-formed nested comments. (2 p)

Solution: Recall that regular languages are closed under intersection, i.e., that if L_1 and L_2 are regular languages then so is $L_1 \cap L_2$. Thus, if it can be shown that $L_1 \cap L_2$ is not regular and that, e.g, L_2 is regular then we can draw the conclusion that L_1 is not regular.

Let L_1 denote the language of well formed nested C-like comments and let $L_2 = L((/*)^*|(*/)^*)$, i.e., the regular language any number of `/*` followed by any number of `/*`.

Now consider the intersection of L_1 and L_2. It is clear that $L_3 = L_1 \cap L_2 = \{(/*)^n(*/)^n \mid n \geq 1\}$, but we have already proved (in 1.*c*) that $L = \{a^n b^n \mid n \geq 0\}$ is not regular for any a and b. This implies that L_3 is not regular, which together with the fact that L_2 is regular implies that L_1 is not regular.

---

2. Context-free languages (14 p)

a) Explain what it means for a grammar to be ambiguous. (2 p)

Solution: A grammar is ambiguous if there exists a string with more than one left-most derivation (or, equivalently, more than one rightmost derivation).

Alternatively, a grammar is ambiguous if there exist a string with more than one parse tree.

---

b) Consider the following small grammar for an expression language

$$E ::= E + E \mid E * E \mid NUM$$

where NUM represents numbers. Show that this grammar is ambiguous. (2 p)

Solution: Number the production rules 1, 2 and 3 in order from the left. Consider the string $1+2*3$ with the following two left-most derivations

$$\begin{aligned} &1) \quad E \xrightarrow{1} E+E \xrightarrow{3} 1+E \xrightarrow{2} 1+E*E \xrightarrow{3} 1+2*E \xrightarrow{3} 1+2*3 \\ &2) \quad E \xrightarrow{2} E*E \xrightarrow{1} E+E*E \xrightarrow{3} 1+E*E \xrightarrow{3} 1+2*E \xrightarrow{3} 1+2*3 \end{aligned}$$

corresponding to $1+(2*3)$ and $(1+2)*3$ respectively.

---

c) Rewrite the above grammar to be unambiguous and show by example that your grammar follows the standard precedence rules for addition and multiplication. (2 p)

Solution: We need to encode that multiplication binds harder than addition. This can be done in the following way

$$\begin{aligned} E &::= E+T \mid T \\ T &::= T*NUM \mid NUM \end{aligned}$$

Now, numbering the production rules E_1, E_2, T_1 and T_2 from left to right the string $1+2*3$ has a unique left-most derivation

$$E \xrightarrow{E_1} E+T \xrightarrow{E_2} T+T \xrightarrow{T_2} 1+T \xrightarrow{T_1} 1+T*3 \xrightarrow{T_2} 1+2*3$$

---

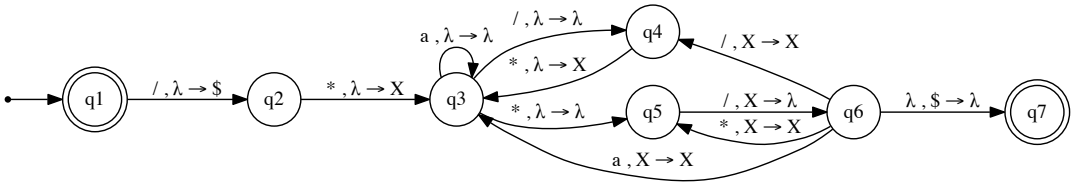
d) Consider the following grammar of well-formed nested comments

$$\begin{aligned} S &::= \lambda \mid /*A*/ \\ A &::= \lambda \mid aA \mid SA \end{aligned}$$

where a, $*$, and $/$ are terminals. Create a deterministic pushdown automaton that recognizes the above grammar, and show the sequence of instantaneous description corresponding to running the automaton on 1) `/*a/**/*/` and 2) `/**/a` (6 p)

Solution: Note that the methods for automatically translating context-free grammars to push-down automata generate non-deterministic push-down automata. By studying the language generated by S, however, it is relatively easy to create a DPDA.

The key idea is to use the stack to count the number of starting symbols `/*` and to make sure they match up with the number of ending symbols `*/`. Between these symbols any number of as can occur. A key thing to note is that once the outermost starting symbol has been matched nothing else can be matched. Below it is assumed that the stack is initially empty and use \$ as the distinguished symbol to signal the end of the stack.



1) For /*/**/*/ we get the following derivation

$(q_1, /*a/**/*/, \$) \mapsto (q_2, *a/**/*/, \$) \mapsto (q_3, a/**/*/, X\$) \mapsto (q_3, /**/*/, X\$) \mapsto$
$(q_4, **/*/, X\$) \mapsto (q_3, */*/, XX\$) \mapsto (q_5, /*/, XX\$) \mapsto (q_6, */, X\$) \mapsto (q_5, /, X\$) \mapsto$
$\mapsto (q_6, \lambda, \$) \mapsto (q_7, \lambda, \$)$ accept

2) For /**/a we get the following derivation.

$(q_1, /**/a, \$) \mapsto (q_2, **/a, \$) \mapsto (q_3, */a, X\$) \mapsto (q_5, /a, X\$) \mapsto (q_6, a, \$) \mapsto$
$\mapsto (q_7, a, \$)$ reject - input string not empty

---

e) Do deterministic pushdown automata have the same computational power as non-deterministic pushdown automata? Give a convincing argument for or against. (2 p)

Solution: Nondeterministic pushdown automata have are strictly more powerful than deterministic pushdown automata. As an example, consider the language of even length palindromes $\{ww^R \mid w \in \Sigma^*\}$ for $\Sigma = \{a, b\}$. In order for a deterministic pushdown automaton to check if a given string w is a palindrome it would have to know when it has reached the middle of the string in order to start matching. However, unlike the language $\{a^n b^n \mid n \geq 0\}$, where the transition from the last a to the first b signifies the middle there is no way for a deterministic pushdown automaton to know when it has reached the middle of the palindrome. A nondeterministic pushdown automaton, however, is able to do this, since the non-determinism allows it to search all possible splits.

---

3. Restriction-free languages and theory of computation (12 p)

a) Consider the Turing machine described by

$$M = (\{q_1, q_2, q_2, q_3, q_4, q_5, q_6, q_7\}, \{a, b\}, \{a, b, \Box\}, \delta, q_1, \Box, \{q_7\})$$

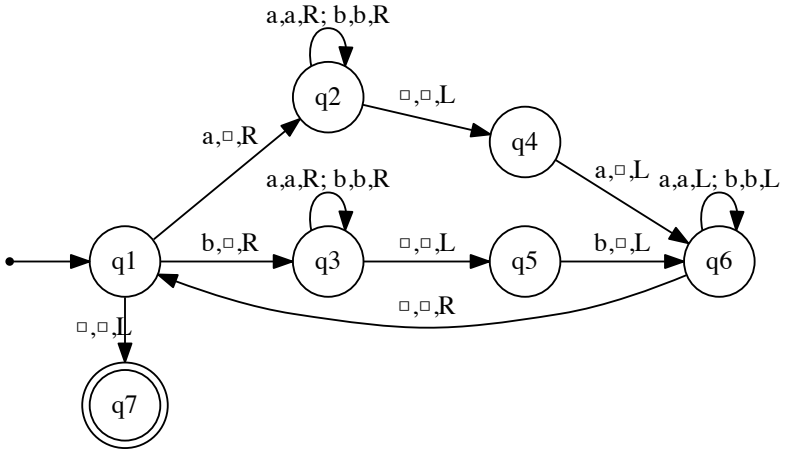
with δ defined as follows

$$\begin{array}{lcllcllcl}
\delta(q_1,a) & = & (q_2,\Box,R) & \delta(q_2,a) & = & (q_2,a,R) & \delta(q_3,a) & = & (q_3,a,R) \\
\delta(q_1,b) & = & (q_3,\Box,R) & \delta(q_2,b) & = & (q_2,b,R) & \delta(q_3,b) & = & (q_3,b,R) \\
\delta(q_1,\Box) & = & (q_7,\Box,L) & \delta(q_2,\Box) & = & (q_4,\Box,L) & \delta(q_3,\Box) & = & (q_5,\Box,L) \\
 & & & & & & & & \\
\delta(q_4,a) & = & (q_6,\Box,L) & \delta(q_5,b) & = & (q_6,\Box,L) & \delta(q_6,a) & = & (q_6,a,L) \\
 & & & & & & \delta(q_6,b) & = & (q_6,b,L) \\
 & & & & & & \delta(q_6,\Box) & = & (q_1,\Box,R)
\end{array}$$

1. Draw the transition graph of M. (2 p)
2. What language does M accept? (2 p)
3. Select a string $w \in \{a,b\}^4$ that is accepted by M and show the execution of M on w as a sequence of instantaneous descriptions. (2 p)

Solution:

1. The state machine corresponding to M is the following.



2. M recognizes the language of even length palindromes, $\{ww^R \mid w \in \Sigma^*\}$. To see this consider how M operates. First, M assume that M is started with some input w, i.e., corresponding to the instantaneous description $q_1 w$. First, the empty string is accepted, since if the first character is the blank symbol $\Box$, M accepts. Otherwise, the first symbol is either an a, or a b. In both cases, M overwrites it with the $\Box$ and searches for the last non-blank symbol if it exists. If it does not exist or if it does not match the overwritten symbol the machine halts. Otherwise, M overwrites the matching symbol and searches for the first non-blank symbol. If it exists, the process repeat until no more symbols remain. At this point M accepts.

   Phrased more distinctly, M shaves off matching symbols in both ends until only the empty string remains, which is accepted.

3. For $w = abba$ we have the following derivation

$\square\, q_1\, abba\square \vdash \square\square\, q_2\, bba\square \vdash \square\square b\, q_2\, ba\square \vdash \square\square bb\, q_2\, a\square \vdash \square\square bba\, q_2\square \vdash$
$\vdash \square\square bb\, q_4\, a\square \vdash \square\square b\, q_6\, b\square\square \vdash \square\square\, q_6\, bb\square\square \vdash \square\, q_6\, \square bb\square\square \vdash \square\square\, q_1\, bb\square\square \vdash$
$\vdash \square\square\square\, q_3\, b\square\square \vdash \square\square\square b\, q_3\, \square\square \vdash \square\square\square\, q_5\, b\square\square \vdash$
$\vdash \square\square\, q_6\, \square\square\square\square \vdash \square\square\square\, q_1\, \square\square\square \vdash \square\square\, q_7\, \square\square\square\square$, halt and accept

---

b) Explain the concept of reduction proof in the context of theory of computation, i.e., proving undecidability by reducing one problem to another. (2 p)

Solution: A proof by reduction is a proof where we reduce a known undecidable problem A to another problem B. The reduction is performed showing how to use a decision procedure for B as a decision procedure for A. This means that the existence of decision procedures for B implies the existence of decision procedure for A. However, since we know that there cannot exist any decision procedures for problem A we draw the conclusion that there cannot exist any decision procedures for B.

---

c) Recall the state-entry problem, i.e., given a Turing machine $M = (Q, \Sigma, \Gamma, \delta, q_0, \square, F)$ and any $q \in Q$, and $w \in \Sigma^+$, decide whether or not the state q is entered by M when it is applied to w. Prove that the state-entry problem is undecidable. (4 p)

Solution: Proof by reducing the halting problem into the state-entry problem.

We do this by creating a Turing machine $\hat{M}$ that halts in state $\hat{q}$ if and only if M halts.

Recall that a Turing machine halts in state q_i for input a when $\delta(q_i, a)$ is undefined (we demand that final states are halting for all input). Thus, we can create $\hat{\delta}$ by mapping any undefined transitions in to $\hat{q}$.

$$\hat{\delta}(q_i, a) = \begin{cases} \delta(q_i, a) & \text{if } \delta(q_i, a) \text{ defined} \\ (\hat{q}, a, R) & \text{otherwise} \end{cases}$$

Now, the state-entry problem for $\hat{M}$, $\hat{q}$, and w is equivalent to the halting problem for M and w.
