Collected Exercises and Exams (with Solutions), ACS 1, Fall 2004

Exercises for ACS 1, Fall 2004, sheet 1 – Solution sheet

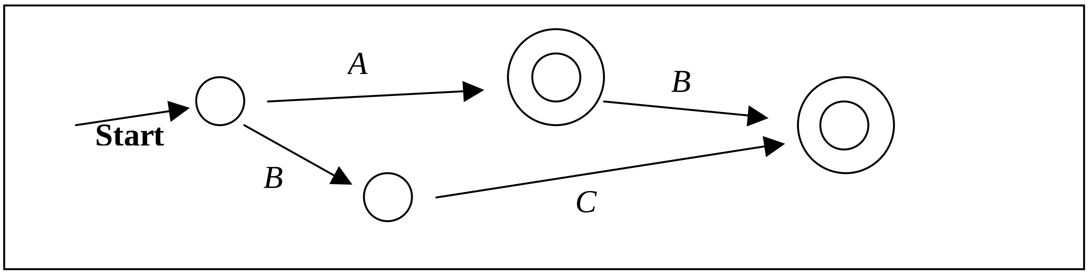
Exercise 1 (a) (5 points) How many words exist over the symbol set $S = \{1\}$? and over the symbol set $S = \{a, b\}$? **(b)** (2 points) How many words of length n exist over a symbol set of size k? **(c)** (5 points) How many languages exist over the symbol sets from (a)? **(d)** (3 points) How many languages of words of length n exist over a symbol set of size k?

Solution: (a) The words over $S = \{1\}$ are ε, 1, 11, 111, ... – that is, as many as there are integers, that is, countably many. The words over $S = \{a, b\}$ can be listed in a sequence, shortest first, sorted alphebetically for same size: ε, a, b, aa, ab, ba, bb, aaa, aab, ... – that is, again countably many. **(b)** k^n many. **(c)** In both cases, $|\Sigma^*| = \mathbb{N}$, that is, there are $2^{\mathbb{N}}$ many languages over these alphabets – indeed, over any finite alphabet there are $2^{\mathbb{N}}$ languages. **(d)** Since there are k^n many words of length n over a symbol set of size k, there are $2^{(k^n)}$ many such languages.

Exercise 2 (10 points). Design a DFA for the language $L = \{w \in \{0,1\}^* \mid w$ contains an uneven number of 0's$\}$. Present your DFA by a graph representation (transition diagram).

Solution. One solution is to take the DFA shown after Def. 3.3. in the script, redefining the set of accepting states to be $\{q_1, q_3\}$.

Exercise 3 (50 points). A *2-level-NFA* over a symbol set Σ is a generalization of NFAs. Informally, in a graph representation of a 2-level-NFA, transitions are labelled not by symbols but by ordinary NFAs (each over Σ). A transition graph of a 2-level-NFA might look like this:



In this graph, A, B, C correspond to three ordinary NFAs over Σ. **(a)** (25 points) Give a formal definition of 2-level-NFAs and their accepted languages. **(b)** (25 points) Prove that the languages accepted by 2-level NFAs are the regular languages.

Solution. (a) Definition of 2-level-NFAs:

Definition 1. Let Σ be an alphabet. Let $\Sigma_1 = \{A_1,...,A_m\}$, where A_i is an NFA over Σ ($i = 1,...,m$). A *2-level-NFA* is a quintuple $(Q, \Sigma , \Sigma_1, \delta, q_0, F)$, where $Q = \{q_0, q_1, ..., q_n\}$ is a finite set of *states*, Σ_1 is the set of *edge NFAs*, $\delta\colon Q \times \Sigma_1 \to \mathrm{Pot}(Q)$ is the *transition function*, q_0 is the *start state*, and $F \subseteq Q$ is the set of *accepting states.*

Definition of languages accepted by 2-level-NFAs: First define the extended transition function for 2-level-NFAs like for ordinary NFAs, then proceed to the "2nd-level language" accepted by 2-level-NFAs, then proceed to the language accepted by a 2-level-NFA. Altogether this makes 3 definitions:

Definition 2. The *extended transition function* $\hat{\delta} : Q \times \Sigma_1{}^* \to Pot(Q)$ *for a 2-level-NFA* (Q, Σ , Σ_1, δ, q_0, F) is defined by $\forall q \in Q :\ \hat{\delta}(q,\varepsilon) = \{q\}$ and $\forall q \in Q\ \forall w \in \Sigma_1{}^*\ \forall A \in \Sigma_1 :\ \hat{\delta}(q,wA) = \bigcup_{p \in \hat{\delta}(q,w)} \delta(p,A)$.

Definition 3. The *2nd-level language* accepted by a 2-level-NFA (Q, Σ , Σ_1, δ, q_0, F) is the set of all words w over Σ_1 for which $\hat{\delta}\,(q_0, w) \cap F \neq \emptyset$.

Definition 4. The language accepted by a 2-level-NFA B = (Q, Σ , Σ_1, δ, q_0, F) is the set set of all words w over Σ which can be written as $w = v_1 v_2 ... v_k$, where $v_j \in L(A_{i_j})$ and $A_{i_1} ... A_{i_k}$ is in the 2nd-level language accepted by B.

(b) We have to show that the set of regular languages is included in the set of languages accepted by 2-level-NFAs (part A) and vice versa (part B).

Proof of part A: Let A = (Q, Σ , δ, q_0, F) be an ordinary NFA with language $L(A)$. For each $a \in \Sigma$, let A_a be a ordinary NFA with $L(A_a) = \{a\}$. Put $\Sigma_1 = \{A_a \mid a \in \Sigma\}$. Then the 2-level NFA B = (Q, Σ , Σ_1, δ, q_0, F) obviously accepts $L(A)$.

Proof of part B: Let the 2-level NFA B = (Q, Σ , Σ_1, δ, q_0, F) accept the language $L(B)$, where $\Sigma_1 = \{A_1,...,A_m\}$ and where A_i = (Q_i, Σ , δ_i, q_{0i}, F_i). We may assume that the Q_i are pairwise disjoint. Idea: replace any transition $\delta(q, A_i) = p$ of B by the complete set of transitions from A_i, linking q to the start state of A_i by an ε-transition and all accepting states from A_i to p by ε-transitions. Formally, define an ε-NFA C = (Q', Σ , δ', q_0', F') by putting $Q' = Q \cup Q_1 \cup ... \cup Q_m$, $q_0' = q_0$, $F' = F$ and

$$\delta' = \delta_1 \cup ... \cup \delta_m \cup \{(q,\varepsilon, q_{0i}) \mid (q, A_i , p) \in \delta\} \cup \{(p_i,\varepsilon, p) \mid (q, A_i , p) \in \delta \text{ and } p_i \in F_i\}.$$

(Note that we interpret transition functions here as sets of transitions). It is obvious that C accepts the same language as B.

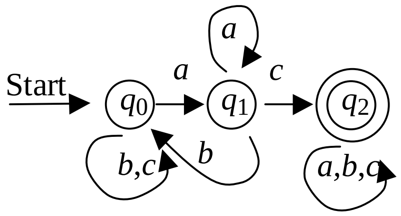
Exercise 4 (15 points). Give a regular expression that tries to catch in an electronic ad newspaper all ads where someone sells a car, like "Wanted: Mercedes model C, built 1999 or later", or "Want to buy: pickup in driveable condition, any make, cheap". As symbol set Σ, use the standard symbols that appear in newspapers (including interpunctuation and space). Minimal requirement for a solution: Your regexp should match the above two examples and at least 1000 others, but it should not match "For sale: Mercedes model C, built 1999" or "Want to pick up a friend". You may assume that some preprocessor has segmented the newspaper text into separate ads; your regexp should match complete car offer ads and reject other ads. Use the regexp syntax from the lecture. You don't have to use boldface for the regexps denoting single symbols. Full points are awarded to solutions that display some effort toward a useful regexp.

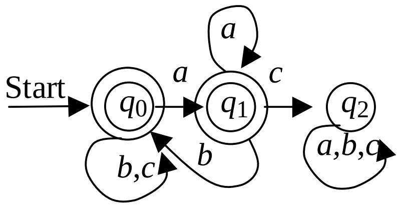
Solution. This calls for creativity. Let's use shorthand S for the regexp (a+b+...+A+...+Z+...+9+0) that matches any one symbol from our symbol set, and let ⎵ be the regexp denoting the space symbol. A minimalistic solution (which would meet the minimal requirements but would disappoint me and get few points) would be

$(\varepsilon+S)^*$ (Wanted: ␣Mercedes + driveable).

Exercise 5. Give a DFA [by way of its transition diagram] (10 points) and a regexp (10 points) for the language L of words over $\{a,b,c\}$ that do not contain *ac* as a subword. Explain by an informal description why your automaton and your regexp work.

Solution. The DFA drawn below (left) obviously accepts all words that do contain the subword *ac*. By exchanging the set $\{q_2\}$ of accepting states for its complement $\{q_0, q_1\}$, one obtains a DFA (right) that accepts L.



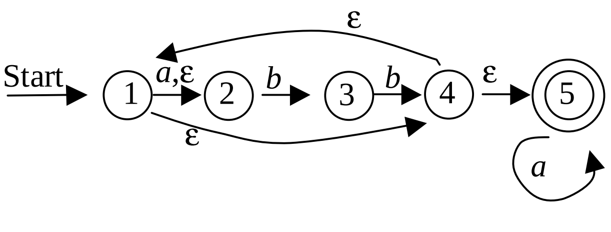


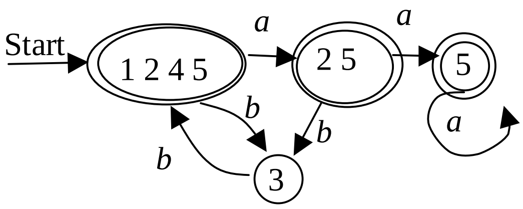
An equivalent regexp could be constructed from the solution DFA by the method from the lecture. Here is another one that is hand-crafted from the solution DFA, observing that L consists of all words whose state paths stay in states q_0, q_1; the regexp describes these paths in a way that is very similar to the regexp $E_i = (R + SU^*T)^*\, SU^*$ given in the proof of prop. 3.3 in the lecture notes:

$((b+c) +(a(b+a)))^*a + ((b+c) +(a(b+a)))^*$ [same as $((b+c) +(a(b+a)))^*(a+\varepsilon)$]

Exercise 6 (25 points). Design a DFA that accepts the language denoted by $(((\varepsilon+a)bb)^*)a^*$, by (i, 10 points) designing first an ε-NFA for this language (inspired by the methods from the proof of proposition 3.4, possibly with simplifications that suggest themselves), then (ii, 15 points) deriving an equivalent DFA from that by the subset construction. Represent your automata by transition diagrams.

Solution. The left diagram shows an ε-NFA for our language, the right the DFA derived from it by the subset construction.





Exercises for ACS 1, Fall 2004, sheet 2 – Solution sheet

Exercise 1. Let $\Sigma = \{0,1\}$. Prove or disprove the following two claims (R, S are language variables):

(a, 5 points) $(L_1 + L_2)^* L_2 =_\Sigma (L_1^* L_2)^*$

(b, 20 points) $(L_1 L_2 + L_1)^* L_1 =_\Sigma L_1 (L_2 L_1 + L_1)^*$

Solution. (a) Claim is false. Consider an interpretation I that assigns the language $\{0\}$ to L_1 and $\{1\}$ to L_2 . Then every word in $L_I((L_1 + L_2)^* L_2)$ must end with 1, but the word ε is in $L_I((L_1^* L_2)^*)$. Thus, the two regexps with language variables are not equivalent.

(b) Claim is true. Using Corollary 3.7, we have to show that $L((ab + a)^* a) = L(a(ba + a)^*)$.

First we show that for any $k \geq 0$, $(ab + a)^k a = a(ba + a)^k$ (note that "=" here denotes equivalence of regexps). For $k = 0$ this is obvious. For other k, use distributivity from Proposition 3.5:

$$\begin{aligned}
(ab + a)^k a &= (ab + a)^{k-1}(ab + a)\, a \\
&= (ab + a)^{k-1}(aba + aa) \\
&= (ab + a)^{k-1}a\, (ba + a) \\
&= (ab + a)^{k-2}(ab + a)a\, (ba + a) \\
&= (ab + a)^{k-2}a\, (ba + a)^2 \\
&= \ldots \\
&= a\, (ba + a)^k
\end{aligned}$$

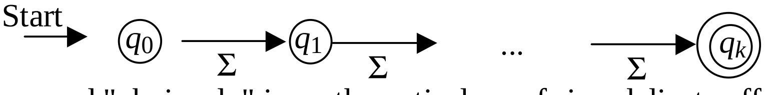
Now if $w \in L((ab + a)^* a)$, then $w \in L((ab + a)^k a)$ for some k. Using our insight just derived, we conclude $w \in L\, (a\, (ba + a)^k) \subseteq L(a(ba + a)^*)$. Thus $L((ab + a)^* a) \subseteq L(a(ba + a)^*)$. The reverse direction follows analogically.

Exercise 2 (10 points). Prove that the language $L = \{0^n \mid n \text{ is a power of } 2\}$ is not regular.

Solution. A clear case for the pumping lemma. Assume L is regular. Let m be a pumping lemma constant. Then $0^{2^m} \in L$. The P.L. implies that also $0^{2^m+k} \in L$, where $1 \leq k \leq m$. But 2^m+k is not a power of 2, so $0^{2^m+k} \notin L$, a contradiction.

Exercise 3 (15 points). Let L be a regular language specified by a DFA, NFA, ε-NFA, or regexp. Show that it is decidable whether $L = \Sigma^k$ for some $k > 0$.

Solution. There are many ways of how this can be decided. One elegant way is to first construct the minimal DFA A for L. Then obviously $L = \Sigma^k$ for some k iff A has the form

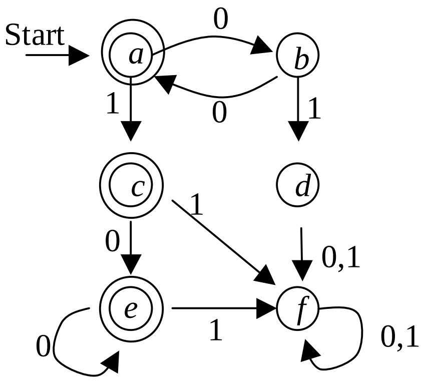


Note. The use of the word "obviously" in mathematical proofs is a delicate affair. One never knows what the reader is ready to accept as obvious. Here I think we have a borderline case, and one might feel the need to prove that if $L = \Sigma^k$ then the minimal automaton actually has the given form (it is really obviously obvious that this kind of DFA accepts $L = \Sigma^k$, so the only possibly questionable claim is its minimality). Minimality of DFAs of the shown form could be proven by going through the table-filling algorithm and showing that all the shown states are distinguishable. This would be a case for extra grading points.

Exercise 4 (5 points). Is the class of regular languages closed under infinite union?

Solution. Heaven, no!!!! Let L be a non-regular language (the pumping lemma has provided us with examples of non-regular langugages – and we also know that L must be infinite). Then $L = \bigcup_{w \in L} \{w\}$. But each $\{w\}$ is a regular language, so the regular languages cannot be closed under infinite union.

Exercise 5 (15 points) Minimize the DFA shown in the figure by using the table filling method. Deliverables: the filling table, the set of states of the minimal DFA, and a graph representation of the minimal DFA.



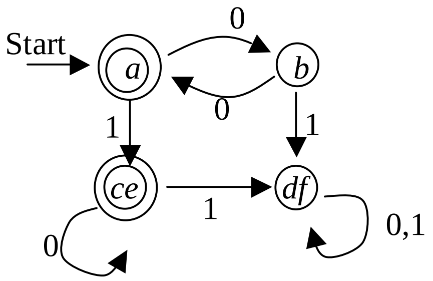
Solution. Manual labour by accurate following of the recipe...

Table:

b	x_1				
c	x_2	x_1			
d	x_1	x_2	x_1		
e	x_2	x_1		x_1	
f	x_1	x_2	x_1		x_1
	a	b	c	d	e

New states: $\{a\}$, $\{b\}$, $\{c,d\}$, $\{e,f\}$

Minimal DFA:



Exercise 6 (a, 10 points) Describe a general method by which one may transform any DFA A into an equivalent ε-NFA that has only a single accepting state. **(b, 10 points)** Transform your result minimal DFA from exercise 1 into a single-accepting-state ε-NFA, either using the general method you described in part **(a)** or by using insight.

Solution (a) First transform A into an equivalent regexp, then transform that into an equivalent ε-NFA using the method from the proof of proposition 3.4 from the script – which yields an ε-NFA with only one accepting state. **(b)** By insight: add an ε to the transition from state a to state ce in the minimal DFA shown in the solution of exercise 1. Then make state a non-accepting. The resulting ε-NFA clearly accepts the same language.

Exercise 7 (30 points). Consider a minimal DFA A that works on an alphabet Σ. Assume that you are told that it has m states, but you know nothing more about A – it is a "black box" and your only way of getting information about A is to feed in words and observe whether they are accepted. Give an algorithm for determining the transition diagram of A from (any finite number of) such observations. One extremely expensive method would be to first construct all possible minimal DFAs of size m, [a HUGE number of DFAs this would give you!] then start testing all words from Σ* in alphabetical enumeration, weeding out all DFAs that on some word behave different from your black box. Then at some point only one of your DFAs is left, --> problem solved. Don't do it this way, but reconstruct R_L from the Myhill-Nerode theorem – that gives a much faster reconstruction.

One possible solution. We know that R_L has m classes. First observe that in order to check whether for any words u,v it holds that uR_Lv, one only has to check whether $uw \in L(A) \Leftrightarrow vw \in L(A)$ for all words w of length $\leq m$ (two words u,v are not equivalent iff they can be distinguished by a word w of length $\leq m$, that is, $uw \in L(A)$ but $vw \notin L(A)$ or vice versa – this is the idea of the table-filling algorithm). So it is possible to determine whether uR_Lv by at most Σ^m many checks.

We next procure representatives $r_1, ..., r_m$ of the R_L-equivalence classes, such that we may write them as $[r_1],..., [r_m]$. We may assume that these representatives have length at most m. Choose $r_1, ..., r_m$ among all words u,v of length $\leq m$ such that the chosen r_i are pairwise not equivalent. Without loss of generality choose $r_1 = \varepsilon$.

Identify the states of the DFA you are about to reconstruct with $[r_1],..., [r_m]$. Assign $[r_1] = [\varepsilon]$ to be the starting state, according to the construction in the proof of the Myhill-Nerode proposition, and choose as accepting states all $[r_i]$ where $r_i \in L(A)$. Finish your construction by putting

$$\text{for } i = 1,...,m,\ a \in \Sigma\text{: } \delta([\ r_i], a) = [r_i\ a] = [\ r_j],$$

where you use the method to decide uR_Lv to find out to which representative r_j the word $r_i\ a$ is equivalent.

Exercises for ACS 1, Fall 2004, sheet 3: Solutions

Exercise 1. (10 points) Give a CFG for all words over the terminal alphabet $T = \{\boldsymbol{a}, \boldsymbol{b}, +, *, (,), \varepsilon, \varnothing\}$ that are regular expressions over $\Sigma = \{a, b\}$.

Solution. Put $V = \{E\}$ (which automatically makes E the start variable). Then simply replicate the inductive definition of regexps:

$E \rightarrow \boldsymbol{a} \mid \boldsymbol{b} \mid \varepsilon \mid \varnothing \mid (EE) \mid (E{+}E) \mid (E^*)$

Exercise 2. (a, 10 points**)** Give a CFG for the language $L = \{w \in \{a, b\}^* \mid w = a^n b^{2n}$ for some $n > 0\}$. **(b**, 20 points**)** Prove with a proof similar to the one from example 4.5. in the lecture notes that your grammar really generates the language L.

Solution. (a) Put $V = \{S, R\}$, then the following grammar G does the trick:
$S \rightarrow aRbb$
$R \rightarrow \varepsilon \mid aRbb$

(b) We show that $w = a^n b^{2n}$ iff $w \in L(G)$.
" $\Rightarrow$ " Let $w = a^n b^{2n}$. We show $w \in L(G)$ by induction on n. Basis: $n = 1$: $w = abb$ is in $L(G)$ because $S \Rightarrow aRbb \Rightarrow a\,\varepsilon\, bb = abb$. Induction: Let $a^n b^{2n} \in L(G)$. We show that $a^{n+1} b^{2(n+1)} \in L(G)$. There must be a derivation for $a^n b^{2n}$, which must start with $S \Rightarrow aRbb$, and be continued by $R \Rightarrow^* a^{n-1} b^{2(n-1)}$. If we replace the first derivation $S \Rightarrow aRbb$ by $S \Rightarrow aRbb \Rightarrow aaRbbbb$, and continue to replace R by $a^{n-1} b^{2(n-1)}$, we end up with $a^{n+1} b^{2(n+1)}$.

" $\Leftarrow$ " Let $w \in L(G)$, and let w have a derivation of length n. We show by induction on n that $w = a^{n-1} b^{2(n-1)}$. Basis: The shortest possible derivation has length $n = 2$ and is $S \Rightarrow aRbb \Rightarrow a\,\varepsilon\, bb$, which yields a word of the required form. Induction: Let w have a derivation of length $n+1$, where $n \geq 2$. The derivation of w must have the form $S \Rightarrow aRbb \Rightarrow aaRbbbb \Rightarrow \ldots \Rightarrow w = aa\gamma bbbb$, where γ is a word of terminals derived from R in n-1 steps. This implies $S \Rightarrow aRbb \Rightarrow \ldots \Rightarrow a\gamma bb$ in n steps. By induction hypothesis, $a\gamma bb = a^{n-1} b^{2(n-1)}$, that is, $\gamma = a^{n-2} b^{2(n-2)}$. Therefore, $aa\gamma bbbb = a^n b^{2n}$.

Exercise 3. (a, 15 points**)** Give a CFG for the language L over the the terminals $T = \{a, b\}$ whose words contain exactly twice as many b's as a's. **(b**, 20 points**)** Prove that your grammar actually generates exactly the words from L.

Solution. (a) The following grammar G does it:

$S \rightarrow \varepsilon \mid SaSbSbS \mid SbSaSbS \mid SbSbSaS$

(b) (sketch) It is obvious that G generates only words whith twice as many b's as a's. To show that every such word w can be generated by G, we proceed by induction on the length of w. If $|w| = 0$ or $|w| = 3$, it is clear that w can be generated by G. If w has length $3n$ greater than 3, observe that w must contain at least one subword v of the form abb or bab or bba (because otherwise all subwords of length 3 would contain more a's than b's, which would make it impossible for w to contain more b's than a's.) Replace this subword within w by ε, to obtain a word $w' = w''\varepsilon w'''$ of length $3(n$-1), which can be generated by G by induction hypothesis, and where the ε marks the place where the subword has been cut out. Conclude by arguing that the parse tree of w' must have had a final branch $S \to \varepsilon$ to obtain the highlighted ε. This branch can be replaced by a derivation of v.

Exercise 4. The grammar $E \to E{+}E \mid E{*}E \mid (E) \mid \mathbf{id}$ generates the arithmetic expressions with +, *, parentheses and **id**. The grammar is ambiguous because **id** + **id** * **id** has two different parse trees. **(a**, 5 points**)** Construct an equivalent unambiguous grammar. **(b**, 25 points**)** Construct an unambiguous grammar for all arithmetic expressions with no redundant parentheses. A pair of parentheses is redundant if its removal does not algebraically change the expression, e.g., the parentheses are redundant in **id** + (**id** * **id**) or **id** * (**id**) but not in (**id** + **id**) * **id**. Explain the idea behind your grammar in words. Give derivations for (**id** + **id**) * **id** and **id** * (**id** + **id**).

Solution. (a) We follow closely the recipe from Example 4.3. in the lecture notes:

$$\begin{array}{lcl}
E & \to & T \mid E + T \\
T & \to & F \mid T * F \\
F & \to & \mathbf{id} \mid (E)
\end{array}$$

(b)

$$\begin{array}{lcll}
E & \to & T \mid E + T & (1) \\
E' & \to & E + T & (2) \\
T & \to & \mathbf{id} \mid T' * F & (3) \\
T' & \to & F \mid T' * F & (4) \\
F & \to & \mathbf{id} \mid (E') & (5)
\end{array}$$

Explanation: Parentheses can be redundant for three reasons: (i) they embrace a product or an atomic **id**, or (ii) they embrace the entire expression, or (iii) they embrace a sum that is not preceded or followed by a *. Our grammar precludes all three possibilities: (i) parentheses can only be introduced with rule (5), which leads to rule (2), which enforces a + inside parentheses. (ii): Parentheses can only be introduced by rule (5), which needs a prior derivation of F, which can only be introduced together with a * by (3) or via T', which itself comes with a * in (3). (iii): same argument as for (ii); ensures that a * must come before or after parentheses.

Derivation of (**id** + **id**) * **id**: $E \Rightarrow T \Rightarrow T' * F \Rightarrow F * F \Rightarrow^* (E') * \mathbf{id} \Rightarrow^* (\mathbf{id} + \mathbf{id}) * \mathbf{id}$

Derivation of **id** * (**id** + **id**): $E \Rightarrow T \Rightarrow T' * F \Rightarrow F * F \Rightarrow^* \mathbf{id} * (E') \Rightarrow^* \mathbf{id} * (\mathbf{id} + \mathbf{id})$

Exercise 5. (15 points) Design a PDA for the language L of words over the terminal alphabet $T = \{(,)\}$ that belong to the language of the grammar $S \to S\,S \mid (S) \mid \varepsilon$. (This is the language of all "balanced parenthesis" words). The PDA should accept by going into an accepting state. Specify your PDA by its transition function, and describe the principles behind your design in intuitive terms.

Solution. Intuitive description: the PDA may, at any time and regardless of top stack symbol, read in an opening "(" and memorizes the opening by pushing one ")" on the stack. It may read in a ")" only if an ")" is the top stack symbol, which is then deleted. When the bottom stack symbol Z_0 is seen, the PDA may enter an accepting state with ε input.

The state set is $\{q,p\}$, where q is the start state and p the accepting state. The stack symbols are $\{Z_0,)\}$. Here is the transition function:

$\delta(q, \varepsilon, Z_0) = \{(p, Z_0)\}$
$\delta(q, (, \#) = \{(q,)\#)\}$ for any stack symbol #
$\delta(q,),)) = \{(q, \varepsilon)\}$

Exercises for ACS 1, Fall 2004, sheet 4: Solutions

Return solutions in paper form on Friday Nov. 12, in the lecture

Note: a maximum of 100 points is accredited for this sheet.

Exercise 1. (20 points) Give a PDA to accept $L = \{0^n1^m2^k \mid n, m, k \geq 1 \text{ and } (n \neq m \text{ or } m \neq k)\}$ by accepting state. Describe the idea behind your PDA in words and specify its transition function.

Solution (partial). L is the union of $L_1 = \{0^n1^m2^k \mid n, m, k \geq 1 \text{ and } n \neq m\}$ with $L_2 = \{0^n1^m2^k \mid n, m, k \geq 1 \text{ and } m \neq k\}$. If we have PDAs P_1 and P_2 for L_1 and L_2, where the state and stack symbol sets of P_1 and P_2 are disjoint and have start states q_1 and q_2 and top stack symbols Z_1 and Z_2, we can combine them into a single PDA P for L by joining all states and rules, declaring some new state q_0 and new top stack symbol Z as start state for P and add the transition rule $\delta(q_0, \varepsilon, Z) = \{(q_1, Z_1), (q_2, Z_2)\}$. Then P initially takes a random choice between P_1 and P_2, after which it carries out a run of the chosen PDA. Obviously P accepts L.

It remains to provide PDAs for L_1 and L_2. Here I only describe the first case. Idea: The PDA for L_1 always ends dead when the input is not of the form $0^n1^m2^k$, where $n, m, k \geq 1$. This can be achieved by ensuring that the set S_0 of states that are entered after reading 0 are disjoint from the set S_1 of states that are entered after reading 1, and again both sets are disjoint from the 2-reachable states S_2. Furthermore, states from S_1 can only be reached from states of S_0 or S_1, and states from S_2 only from S_1 or S_2 states. Accepting states are all in S_2. To check the conditions $n \neq m$, the PDA first memorizes the number of read 0's by copying them on the stack. When it starts reading 1's, it cancels 0's from the stack until one of the following occurs:

(i) It reads a 1 but the stack has no more 0's. Then $n \neq m$. The PDA enters a mode where it only checks whether the rest of the word is of form $1^{m'}2^k$, where $m' \geq 0$, $k \geq 1$.

(ii) It reads the first 2 but the stack still has 0's. Then again $n \neq m$. The PDA enters a mode where it only checks whether the rest of the word is of form $2^{k'}$, where $k' \geq 0$.

(iii) It reads the first 2 exactly after it has deleted the last 0 from the stack. Then $n = m$ and the PDA is halted in a dead end.

Here is a list of the required transitions. The stack top symbol is Z_1, and the start state is q_1. The single accepting state is q_{accept}.

1. $\delta(q_1, 0, Z_1) = \{(q_1, 0Z_1)\}$
2. $\delta(q_1, 0, 0) = \{(q_1, 00)\}$
3. $\delta(q_1, 1, 0) = \{(q_2, \varepsilon)\}$; entering the downcounting of 0's
4. $\delta(q_2, 1, 0) = \{(q_2, \varepsilon)\}$
5. $\delta(q_2, 1, Z_1) = \{(q_3, Z_1)\}$; entering mode (i)
6. $\delta(q_3, 1, Z_1) = \{(q_3, Z_1)\}$; continue reading 1's
7. $\delta(q_3, 2, Z_1) = \{(q_{\text{accept}}, Z_1)\}$
8. $\delta(q_{\text{accept}}, 2, \#) = \{(q_{\text{accept}}, \#)\}$; # is any stack symbol
9. $\delta(q_2, 2, 0) = \{(q_{\text{accept}}, \varepsilon)\}$; entering mode (ii)
10. $\delta(q_2, 2, Z_1) = \{\}$; case (iii)

Exercise 2 (20 points) The PDAs that we use (and everybody else) have a single stack memory. One might wish to increase the power of PDAs by adding more such memories. Give a formal definition of an "*n*-stack PDA", its configurations, and the languages accepted by them (by final state). Follow the definitions 4.9 – 4.12 of the lecture notes. Your definition should be so general as to include the special cases where $n = 0$ (and then your definition should be equivalent to an NFA); for $n = 1$ your *n*-stack PDA should be equivalent to our familiar PDAs. Note: there isn't a unique "correct" such definition. *n*-stack PDAs can be defined in various ways, not necessarily equivalent. The purpose of this exercise is that you train writing clean definitions, not to find the "correct" definition. Hint: to cover the case $n = 0$ without any extra case distinctions, you may use the convention that for any set X, X^0 is $\{\varnothing\}$.

Solution. (1) (one possible definition of *n*-stack PDAs) Let $n \geq 0$. An *n*-stack PDA is a 7-tuple $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, where

- Q is a finite set of states,
- Σ is a finite set of input symbols,
- Γ is a finite stack alphabet,
- $\delta: Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma^n \rightarrow \text{Pot}_0(Q \times (\Gamma^*)^n)$ is the transition function,
- $q_0 \in Q$ is the start state,
- $Z_0 \in \Gamma$ is the start stack symbol,
- $F \subseteq Q$ is the set of accepting states.

(2) (configuration) A *configuration* of an *n*-stack PDA $(Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ is triple (q, w, γ), where

- $q \in Q$ is the current state of the PDA,
- $w \in \Sigma^*$ is the remaining input word,
- $\gamma \in (\Gamma^*)^n$ is the *n*-tuple of current stack contents (*i*-th word of γ = *i*-th stack).

(3) (move)) For an *n*-stack PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, all $q, q' \in Q$, $a \in \Sigma \cup \{\varepsilon\}$, $w \in \Sigma^*$, $(X_1,..., X_n) \in \Gamma^n$, and $(\alpha_1,..., \alpha_n), (\beta_1,..., \beta_n) \in (\Gamma^*)^n$ define

$$(q, aw, (X_1 \beta_1,..., X_n \beta_n)) \vdash_P (q', w, (\alpha_1 \beta_1,..., \alpha_n \beta_n)) \quad \text{iff}$$

$(q', (\alpha_1, ..., \alpha_n)) \in \delta(q, a, (X_1, ..., X_n))$.

As usual, define by $\vdash_P$* the transitive-reflexive closure of $\vdash_P$ (that is, zero or any number of moves).

(4) (languages accepted by final state) Let $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ be an n-state PDA. Then the *language accepted by P by final state* is

$L(P) = \{w \in \Sigma^* \mid (q_0, w, (Z_0, ..., Z_0)) \vdash_P^* (q, \varepsilon, (\alpha_1, ..., \alpha_n))$, where $q \in F$ and $(\alpha_1, ..., \alpha_n) \in (\Gamma^*)^n \}$.

Exercise 3. (20 points) Show that the language $L = \{0^n 1^n 2^n \mid n \geq 1 \}$ can be accepted by a 2-stack PDA, by specifying the transition function and explaining the working principle in words. Note: this language is not a CFL (can be shown via the CFL pumping lemma). Thus, introducing 2-stack PDAs properly extends the class of recognizable languages.

Solution. Idea: while reading 0's, the first stack is filled with 0's to count them. When reading 1's, the first stack is successively emptied in order to check that there are as many 1's as 0's; at the same time, the second stack is filled with 1's. Finally, when reading 2's, their number is checked using the 1's from the second stack. The stack alphabet is $\Gamma = \{Z_0, 0,1\}$; q_3 is the only accepting state. Here is a possible transition function:

$\delta(q_0, 0, (X, Y)) = \{(q_0, (0X, Y), (q_1, (0X, Y))\}$ for any $X, Y \in \Gamma$;
$\delta(q_1, 1, (1, Y)) = \{(q_1, (\varepsilon, 1Y))\}$ for any $Y \in \Gamma$;
$\delta(q_1, 2, (Z_0, 1)) = \{(q_2, (Z_0, \varepsilon))\}$
$\delta(q_2, \varepsilon, (Z_0, Z_0)) = \{(q_3, (Z_0, Z_0))\}$

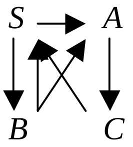
Exercise 4 (30 points) Convert the following grammar $G = (V, T, P, S)$ into CNF, by (i) eliminating ε-productions, (ii) eliminating unit productions, (iii) eliminating useless symbols, (iv) putting the resulting grammar in CNF. Each of the steps (i) to (iv) counts 10 points.

$S \to 0A0 \mid 1B1 \mid AB$
$A \to C$
$B \to S \mid A$
$C \to S \mid \varepsilon$

Solution: (i) **a.** Finding nullable variables: NULL(1) = $\{C\}$, NULL(2) = $\{C, A\}$, NULL(3) = $\{C, A, B\}$, NULL(4) = NULL(5) = $\{C, A, B, S\}$. **b.** For $S \to 0A0$ add $\{S \to 0A0, S \to 00\}$ to P', for $S \to 1B1$ add $\{S \to 1B1, S \to 11\}$ to P', for $S \to AB$ add $\{S \to AB, S \to B, S \to A \}$ to P', for $A \to C$ add $\{ A \to C \}$ to P', for the remaining rules add $\{B \to S, B \to A, C \to S\}$ to P'. This gives a new set P'

$S \to 0A0 \mid 00 \mid 1B1 \mid 11 \mid AB \mid B \mid A$
$A \to C$
$B \to S \mid A$
$C \to S$

(ii) **a.** Finding unit pairs: PAIRS(1) = {(A, A), (B, B), (C, C), (S, S)}, PAIRS(2) = {(A, A), (B, B), (C, C), (S, S), (S, B), (S, A), (A, C), (B, S), (B, A), (C, S)}, PAIRS(3) = {(A, A), (B, B), (C, C), (S, S), (S, B), (S, A), (A, C), (B, S), (B, A), (C, S), (S, C), (A, S), (B, C), (C, A), (C, B)}, PAIRS(4) = PAIRS(5) = {(A, A), (B, B), (C, C), (S, S), (S, B), (A, C), (B, S), (B, A), (C, S), (S, A), (A, S), (B, C), (C, B), (S, C), (A, B), (C, A)}. An easier way to see that here *all* pairs are unit pairs is to check the following directed graph created by the unit transitions from P' and see that it is cyclic, that is, every node is transitively reachable from every other node:



$S \to 0A0 \mid 00 \mid 1B1 \mid 11 \mid AB \mid B \mid A$
$A \to C$
$B \to S \mid A$
$C \to S$

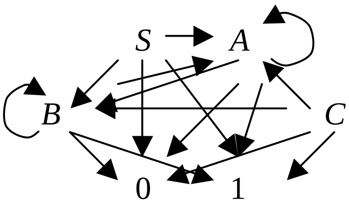
b. Stripping from P' all unit productions and then adding all productions of the form $A \to \alpha$, where $B \to \alpha$ is a non-unit production in P' and (A, B) is a unit pair, yields P'' =

$S \to 0A0 \mid 00 \mid 1B1 \mid 11 \mid AB$
$A \to 0A0 \mid 00 \mid 1B1 \mid 11 \mid AB$
$B \to 0A0 \mid 00 \mid 1B1 \mid 11 \mid AB$
$C \to 0A0 \mid 00 \mid 1B1 \mid 11 \mid AB$

(iii) **a.** We first detect all generating symbols. GEN(1) = {0,1}, GEN(2) = GEN(3) = {0, 1, A, B, C, S}.

b. Deleting from G all nongenerating symbols and productions in which such symbols occur, yields G_2 = (V, T, P'', S), because there are no non-generating symbols or productions.

c. Next we find all reachable symbols of G_2. The graph described in the lecture notes is



From this we see that the reachable symbols are {0, 1, S, A, B}.

d. Finally we eliminate from G_2 all non-reachable symbols and productions in which such symbols occur, to obtain G_1 = ({S, A, B }, {0, 1}, P''', S), where P''' =

$S \to 0A0 \mid 00 \mid 1B1 \mid 11 \mid AB$
$A \to 0A0 \mid 00 \mid 1B1 \mid 11 \mid AB$

$B \to 0A0 \mid 00 \mid 1B1 \mid 11 \mid AB$

(iv) In the last step, we obtain a CNF grammar by carrying out the two steps given in the proof of theorem 4.10 in the lecture notes.

a. Arrange that all bodies of lenght 2 or more consists only of variables. This gives us productions P'''' =

$S \to A_0AA_0 \mid A_0A_0 \mid A_1BA_1 \mid A_1A_1 \mid AB$
$A \to A_0AA_0 \mid A_0A_0 \mid A_1BA_1 \mid A_1A_1 \mid AB$
$B \to A_0AA_0 \mid A_0A_0 \mid A_1BA_1 \mid A_1A_1 \mid AB$
$A_0 \to 0$
$A_1 \to 1$

b. Break productions with all-variable bodies of length 3 or more into sequences of productions of the form $A \to BC$. This gives us the final rule set P_{CNF} =

$S \to A_0A' \mid A_0A_0 \mid A_1B' \mid A_1A_1 \mid AB$
$A \to A_0A' \mid A_0A_0 \mid A_1B' \mid A_1A_1 \mid AB$
$B \to A_0A' \mid A_0A_0 \mid A_1B' \mid A_1A_1 \mid AB$
$A' \to AA_0$
$B' \to BA_1$
$A_0 \to 0$
$A_1 \to 1$

Exercise 5. (30 points) Write an unrestricted grammar for $L = \{0^{(2^n)} \mid \text{n} > 0\}$, and explain in words how it functions.

Solution. This is a classic. Here I copy the solution from an old edition of the HMU book (then only a HU book). Here's the grammar:

1. $S \to AC0B$
2. $C0 \to 00C$
3. $CB \to DB$
4. $CB \to E$
5. $0D \to D0$
6. $AD \to AC$
7. $0E \to E0$
8. $AE \to \varepsilon$

And here is how it works. The idea is that A and B serve as end markers for strings of 0's which are iteratively doubled in length by a "cursor" C that moves through the strings of 0's between A and B, doubling their number by production 2. When C hits the right end marker B, it becomes a D or E by productions 3 or 4. If a D is chosen, it migrates left by production 5 until the left endmarker A is reached. At that point D becomes a C again by production 6 and the process starts again. If an E is chosen, the right end marker is consumed (in 4.) and the E wanders left by production 7 until it hits the left endmarker, which is consumed along with E in production 8. By that time, a word of the desired form $0^{(2^n)}$ is left.

Exercises for ACS 1, Fall 2004, sheet 5 – Solution sheet

Return solutions in paper form on **Wednesday** Nov. 24, in the lecture

Note: a maximum of 100 points is accredited for this sheet.

Exercise 1. (30 points) Give FOL propositions that formally state the following natural-language sentences about personal relationships. Provide a symbol set *S* that you use for all the sentences, and declare what type each symbol is (constant, predicate/relation, function; also state arity). Use the exact FOL syntax as introduced in class.

a. Scott is the author of Waverley.
b. Waverley is a famous classical English novel.
c. A novel is a novel.
d. A novel is a special type of written art.
e. If I am the author of Waverley, and the author of Waverley is Scott, then I am Scott.
f. I like Waverly better than any other novel.
g. There exist at least two men by the name of "Scott".

Which of your propositions are tautologies, which are contradictions?

Solution. (not unique - - there are always many ways to formalize natural language statements) Symbol set: constants Scott_The_Author, Waverley, I, Scott_Name; unary predicate symbols: Famous, Classical, English, Novel, Written_Art; ternary relation symbols: Who_Likes_What_Better_Than; unary function symbols: Author_of, Name_of.

a. Author_of Waverley = Scott_The_Author
b. Famous Waverley $\wedge$ Classical Waverley $\wedge$ English Waverley $\wedge$ Novel Waverley
c. $\forall x_1$ (Novel $x_1 \rightarrow$ Novel x_1)
d. $\forall x_1$ (Novel $x_1 \rightarrow$ Written_Art x_1)
e. (I = Author_of Waverley $\wedge$ Scott = Author_of Waverley $\rightarrow$ I = Scott)
f. $\forall x_1$ (Novel $x_1 \wedge \neg\, x_1$ = Waverley $\rightarrow$ Who_Likes_What_Better_Than I x_1 Waverley)
g. $\exists\, x_1 \exists\, x_2(\neg\, x_1 = x_2\ \wedge$ (Name_of x_1 = Scott_Name $\wedge$ (Name_of x_2 = Scott_Name))

c and e are tautologies, none is a contradiction.

Exercise 2 (20 points) For your symbol set *S* of the previous exercise, describe an *S*-structure in which all the statements of Exercise 1 hold.

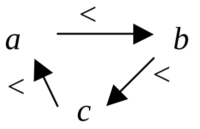
Solution. Put *A* = {Scott, Waverley, I, Scott-the-name }; this set contains two persons, a novel, and a name. Put Famous^A = Classical ^{A} = Novel^A = English^A = Written_Art^A = {Waverley}, Who_Likes_What_Better_Than^A = ∅, Author_of^A = {(Waverley, Scott)}, Name_of^A = {(Scott, Scott-the-name)}.

Exercise 3 (20 points) Let *S* = {<}, where < is a binary relation symbol. Characterize in words the class of all *S*-structures $\mathcal{A}$ which are models of

$\varphi = \forall x_1 \forall x_2 \forall x_3(((((\neg\, x_1 = x_2\ \wedge \neg\, x_2 = x_3) \wedge \neg\, x_1 = x_3) \wedge <x_1\, x_2) \wedge <x_2\, x_3) \rightarrow \neg <x_3\, x_1)$

and give two concrete S-structures, one of which is a model of φ and the other isn't. Present your structures (i) in an intuitive graph-like representation, (ii) formally as sets. How many non-isomorphic models does φ have?

Solution. The models of φ are exactly those {<}-structures that contain no <-cycle of length 3. The simplest S-structure that is a model of φ is given by a singleton set A and empty <, that is, $A = \{a\}$ and $<^A = \varnothing$ (graph-like representation: a single point). The simplest S-structure that is not a model of φ is an isolated 3-cycle of <, that is, $A = \{a, b, c\}$ and $<^A = \{(a,b), (b,c), (c,a)\}$:



Exercise 4 (20 points) For $S = \{<\}$ design a proposition φ such that any model of φ is isomorphic to the 3-cycle $\mathcal{A} = (A, <^A) = (\{a, b, c\}, \{(a,b), (b,c), (c,a)\})$. Note: Questions of this kind – find propositions that characterize structures up to isomorphism – are quintessential in the analysis of mathematical axiom systems; an entire, highly active field of logics called "model theory" is mainly concerned with questions of this kind.

Solution: φ is the conjunction of a proposition φ_1 that states that any model has exactly three elements, and of a proposition φ_2 that states that there exist three elements that occur in a <-cycle:

$$\varphi_1\ =\ \exists x_1 \exists x_2 \exists x_3(((\neg\, x_1 = x_2\ \wedge \neg\, x_2 = x_3) \wedge \neg\, x_1 = x_3) \wedge \forall x_4((x_4 = x_1 \vee\ x_4 = x_2) \vee\ x_4 = x_3))$$

$$\varphi_2\ =\ \exists x_1 \exists x_2 \exists x_3((((((((< x_1\, x_2 \wedge < x_2\, x_3) \wedge < x_3\, x_1\,) \wedge \neg < x_1\, x_1\,) \wedge \neg < x_1\, x_3) \wedge \neg < x_2\, x_2\,) \wedge \neg < x_2\, x_1) \wedge \neg < x_3\, x_3) \wedge \neg < x_3\, x_2)$$

Exercise 5 a. (20 points) Show that for any S and any S-expression φ, $\forall x\, \varphi \models \neg\ \exists\, x \neg\ \varphi$.
b. (20 points) Show that for a ternary relation symbol R, $\forall x\ \forall y\ \exists\, z\, Rxzy \models \exists\, z\ \forall x\ \forall y\, Rxzy$ does not hold.

Solution. a. . Let $(\mathcal{A}, \beta) \models \forall x\, \varphi$ for some $\mathcal{A}$ with domain A. We have to show that $(\mathcal{A}, \beta) \models\ \neg\ \exists\, x \neg\ \varphi$. For all $a \in A$ it holds that $(\mathcal{A}, \beta\frac{a}{x}) \models\ \varphi$. That implies that for no $a \in A$ it holds that not $(\mathcal{A}, \beta\frac{a}{x}) \models\ \varphi$, that is, there exists no $a \in A$ such that $(\mathcal{A}, \beta\frac{a}{x}) \models\ \neg\,\varphi$, that is, it does not hold that there exists some $a \in A$ such that $(\mathcal{A}, \beta\frac{a}{x}) \models\ \neg\,\varphi$, that is, it does not hold that $(\mathcal{A}, \beta) \models \exists\, x \neg\,\varphi$, that is, $(\mathcal{A}, \beta) \models\ \neg\ \exists\, x \neg\ \varphi$.

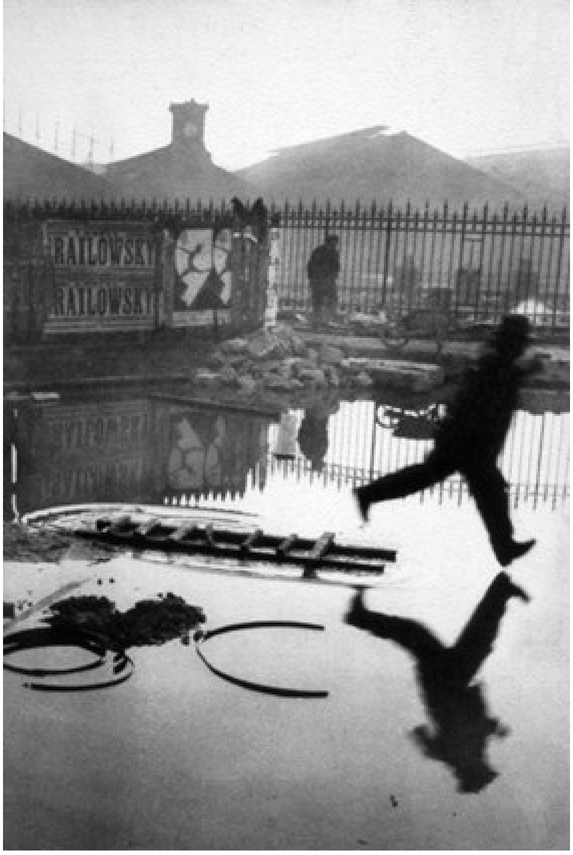
b. We give a counterexample, that is, an $\{R\}$-structure (A, R^A) where $(A, R^A) \models \forall x\ \forall y\ \exists\, z\, Rxzy$ but not $(A, R^A) \models \exists\, z\ \forall x\ \forall y\, Rxzy$. There are many such counterexample structures. One is to take $A = \mathbb{N}$ and choose $R^{\mathbb{N}}$ to be the relation $R^{\mathbb{N}} = \{(k,l,n) \in \mathbb{N}^3 \mid k \leq l \leq n \text{ or } k \geq l \geq n\}$, that is, *Rkln* means that l lies between k and l. Then clearly $(\mathbb{N},R^{\mathbb{N}}) \models \forall x\ \forall y\ \exists\, z\, Rxzy$ but not $(\mathbb{N},R^{\mathbb{N}}) \models \exists\, z\ \forall x\ \forall y\, Rxzy$ (because there exists no natural number that lies between all possible choices of two natural numbers – for instance, 1 does not lie between 6 and 8.)

Exercises for ACS 1, Fall 2004, sheet 6

Return solutions in paper form on Friday Dec. 03, in the lecture

Note: a maximum of 100 points is accredited for this sheet.

Exercise 1. Here is a famous photo[1]:



(a, 15 points) Imagine you would have to describe this picture to a blind friend in 5 sentences. Write these 5 sentences down in FOL. You don't have to specify the type and arity of your symbols; for simplicity and clarity, use "xyz_of" for function symbols; all other symbols are constants or relations.
(b, 15 points) This photo invites aesthetic and philosophical thinking (take a look at the website it was taken from!). Think of one such "deep" thought and argue why it can't be formalized in FOL. (Alternatively, argue that all aesthetic and philosophical thoughts can be expressed in FOL). You think this is a strange exercise? Well, it has been tried to formalize legal reasoning in FOL, for juridical expert systems... and legal reasoning is "deep".

No solution given, because possible solutions are extremely varied.

Exercise 2. You know *many* facts that can easily be expressed in FOL. A *base fact* is a fact that is not logically entailed by other facts that you know. **(a, 10 points)** Give one example of a base fact from your personal knowledge and argue informally that it is a base fact. **(b, 20 points)** Give an estimate of the number of base facts that you know, and explain the reasoning behind your estimate. [Background of this exercise: in Artificial Intelligence, *knowledge bases* for *expert systems* are essentially large collections of base facts from some specific domain of expertise]

No solution given, because possible solutions are extremely varied.

[1] http://www.dienes-and-dienes.com/Cartier-Bresson.html

Exercise 3. (20 points) Consider the following propositions which express that the binary relation R is an equivalence relation:

$\varphi_1 = \forall x\, Rxx$ $\qquad\qquad$ $\varphi_2 = \forall x \forall y\, (Rxy \rightarrow Ryx)$ $\qquad\qquad$ $\varphi_3 = \forall x \forall y \forall z\, ((Rxy \wedge Ryz) \rightarrow Rxz)$

Show that none of these propositions is entailed by the others by presenting $\{R\}$-structures that are models of two of the propositions, but not of the third.

Solution. Here is one possibility.
(i) A model of φ_2 and φ_3 but not of φ_1: $\mathcal{A} = (\{1\}, \emptyset)$
(ii) A model of φ_1 and φ_3 but not of φ_2: $\mathcal{A} = (\{1,2\}, \{\{1,1\},\{1,2\},\{2,2\}\})$
(iii) A model of φ_1 and φ_2 but not of φ_3:
$\mathcal{A} = (\{1,2,3\}, \{\{1,1\},\{2,2\},\{3,3\},\{1,2\},\{2,1\},\{2,3\},\{3,2\}\})$

Exercise 4. (30 points) A DFA can be seen as a structure $\mathcal{D} = (A, S^A, Q^A, F^A, \delta^A, q_0{}^A)$, where the carrier A consists of the states and symbols, S is a unary predicate (intention: S denotes the symbols), Q is a unary predicate (denoting the statesF is a unary predicate (denoting the accepting states),), δ is a binary function (denoting the transition function), and q_0 is a constant symbol (denoting the start state). Give a collection Φ of FOL propositions such that every finite S-structure $\mathcal{D}$ is a model of Φ iff $\mathcal{D}$ corresponds to a DFA. In other works, axiomatize the DFAs in FOL. Explain each of your propositons in words.

Solution. Here is one possibility:

$\forall x\, ((Sx \vee Qx) \wedge \neg\, (Sx \wedge Qx))$	every thing must be either a state or a symbol
$(\exists x\, Sx \wedge \exists x\, Qx)$	state and symbol sets are not empty
Qq_0	the start state is actually a state
$\forall x\, (Fx \rightarrow Qx)$	the accepting states are actually states
$\forall x \forall y \forall z\, (((Qx \wedge Sy) \wedge \delta xy = z\,) \rightarrow Qz)$	δ maps state-symbol pairs on states

Note: because in FOL we only know total functions, in any S-structure $\mathcal{D}$ the function δ^A is totally defined. For the purposes of interpreting $\mathcal{D}$ as a DFA, it is not relevant which type of values δ^A has on argument pairs that are not of type (state, symbol).

Exercise 5. (10 points each) Give a rigorous derivation of the following sequence rules:

a. $\dfrac{}{(\varphi \vee \neg\, \varphi)}$ $\qquad\qquad$ **b.** $\dfrac{\Gamma\ \varphi \neg\, \varphi}{\Gamma \quad \neg\, \varphi}$ $\qquad\qquad$ **c.** $\dfrac{\Gamma\ \varphi \neg\, \psi}{\Gamma\ \psi \neg\, \varphi}$

in the sequence calculus! (Notes: **b.** is the easiest**.** The sequence **a.** has empty antecedent $\Gamma = \emptyset$).

Solution: a.

1.	$\varphi \quad \varphi$	(Pre)
2.	$\varphi \quad (\varphi \vee \neg\, \varphi)$	($\vee$ Con a.) on 1.
3.	$\neg\, \varphi \quad \neg\, \varphi$	(Pre)
4.	$\neg\, \varphi \quad (\varphi \vee \neg\, \varphi)$	($\vee$ Con b.) on 3.

5.	$(\varphi \vee \neg \varphi)$	(Cas) on 2. and 4.

b.

1.	$\Gamma\ \varphi \quad \neg\varphi$	(premise)
2.	$\Gamma\ \neg\varphi \quad \neg\varphi$	(Pre)
3.	$\Gamma \quad \neg \varphi$	(Cas) on 1. and 2.

c.

1.	$\Gamma\ \varphi \quad \neg \psi$	(Premise)
2.	$\Gamma\ \psi\ \ \varphi \quad \psi$	(Pre)
3.	$\Gamma\ \psi\ \ \varphi \quad \neg \psi$	(Ant) on 1.
4.	$\Gamma\ \psi\ \ \varphi \quad \neg \varphi$	(Con) on 2. and 3.
5.	$\Gamma\ \psi\ \ \neg \varphi \quad \neg \varphi$	(Pre)
6.	$\Gamma\ \psi \quad \neg \varphi$	(Cas) on 4. and 5.

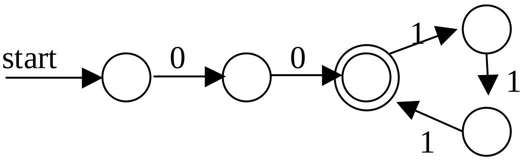
Advanced Computer Science 1
Midterm, October 13, 2004

Solution sheet

Group A

1. (15 points) Design a DFA that accepts $L = \{w \in \{0,1\}^* \mid w = 00(1^{3n})$ for some $n \geq 0\}$. Present your DFA by a transition diagram.

Solution.



(all transitions that remain are not shown, they lead to dead state, likewise not shown)

2. (5 points) Give a regexp for the language from problem 1.

Solution. 00(111)*

3. (20 points) Show that the language $L = \{0^n1^{2n}2^{3n} \in \{0,1,2\}^* \mid n \geq 0\}$ is not regular.

Solution. Pumping lemma! Assume L is regular with pumping constant k. Consider $w = 0^k1^{2k}2^{3k} \in L$. By PL, $w = xyz$, with $|xy| \leq k$, $|y| > 0$. Because $|xy| \leq k$, y must consist entirely of 0's. By PL, then also $0^{k-|y|}1^{2k}2^{3k} \in L$, a contradiction.

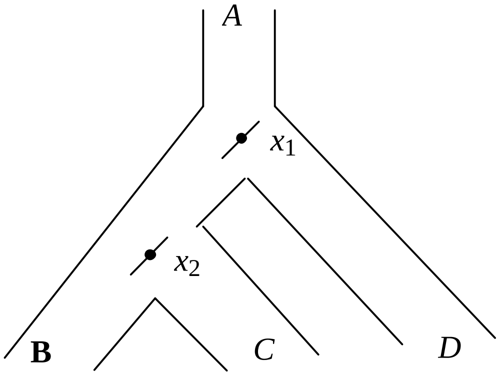
4. (20 points) Show that if $L \subseteq \{0,1\}^*$ is regular, then also $L' = \{w \in L \mid |w| \leq 1$ or the second symbol of w is 1$\}$ is regular.

Solution. $L' = L \cap L_1 \cap L_2$, with $L_1 = \{w \in \{0,1\}^* \mid |w| \leq 1\}$ and $L_2 = \{w \in \{0,1\}^* \mid |w| > 1$ and the second symbol of w is $1\}$. L_1 is finite and thereby regular. L_2 can clearly be represented by the regexp (**0**+**1**)**1**(**0**+**1**)* and is thus also regular. Thus L' is the intersection of three regular languages and thereby regular.

5. (10 points) Give a CFG for the language from problem 1.

Solution. One possibility is $S \to 00T$, $T \to 111T \mid \varepsilon$.

6. (30 points) Consider the toy shown in the figure below. A marble or a marshmallow can be dropped in at A. At the junctions there are levers x_1, x_2 which guide the dropped object to the left or to the right. When a (heavy) marble passes a lever, it flips direction after the object has passed, while a (leight) marshmallow doesn't affect the lever. Initially the levers are directed as in the figure. Denote a marble into A as input 0 and a marshmallow as input 1. A *winning sequence* of inputs is one where the last object dropped in comes out at C. Model this toy by a DFA that accepts the language of all winning 0/1-sequences. Specify your DFA by a transition table.



Solution. The important idea is to code DFA states by the possible flipstates of the two levers. Let *ll* code the situation where both levers direct objects to the left (as in the figure), *lr* code the situation where the first lever points left and the second right, and *rl* and *rr* accordingly. Furthermore, we must also code whether the previous input came out at C: we write, for instance, *ll+* for a toy state where currently both levers point left and the previous drop ended in C, an *ll-* if it came out elsewhere. *ll-* is also the start state. Here is the transition table:

state	input 0	input 1	
ll-	*rr-*	*ll-*	
ll+	*rr-*	*ll-*	*; inaccessible from start state, superfluous*
lr-	*rl+*	*lr+*	
lr+	*rl+*	*lr+*	
rl-	*ll-*	*rl-*	
rl+	*ll-*	*rl-*	
rr-	*lr-*	*rr-*	
rr+	*lr-*	*rr-*	*; inaccessible from start state, superfluous*

Among these states, only *ll- rr- lr- rl+ rl- lr+* are accessible from the start state, the remaining two states are superfluous. The "+"-states are the accepting states.

Advanced Computer Science 1
Final, December 14, 2004

Note: points per problem reflect expected difficulty.

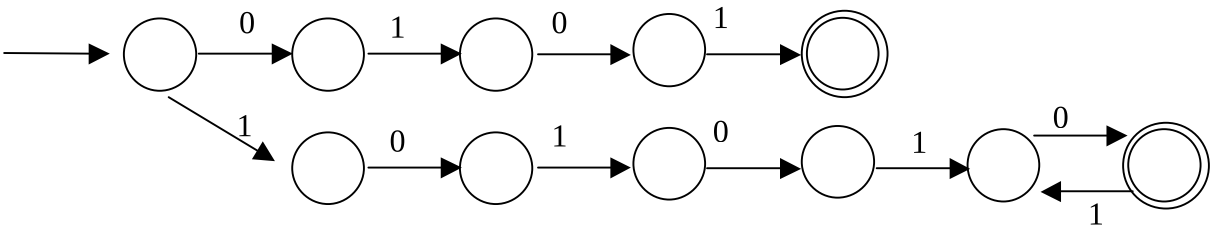
Group A
Solutions

Problem 1. Show that the language $L = \{0^n \in \{0\}^* \mid n$ is not of the form $n = a^a$ for some integer $a\}$ is not regular.

Solution. Assume L is regular. Then $L^{\mathrm{c}} = \{0^n \in \{0\}^* \mid n = a^a$ for some integer $a\}$ is also regular, because the regular languages are closed under complement. Let m be a pumping lemma constant for L^{c}. Then $0^{m^m} \in L$. The P.L. implies that also $0^{m^m+k} \in L$, where $1 \leq k \leq m$. But m^m+k is not of the form a^a, bceause $m^m < m^m+k < (m+1)^{(m+1)}$, so $0^{m^m+k} \notin L^{\mathrm{c}}$, a contradiction. Thus L^{c} is not regular and thereby neither is L. *Note:* for group B, similarly use the PL on the complement language.

Problem 2. Design an automaton (DFA, NFA or ε-NFA) A over the alphabet $\{0,1\}$ such that (i) $L(A) \cap \{0,1\}^4 = \{0101\}$ and (ii) $L(A) \cap L((\mathbf{10})^*)$ is infinite.

Solution. The following DFA accepts only the word 0101 in its upper branch and the inifinite sublanguage $L(\mathbf{101010}(\mathbf{01})^*)$ of $L((\mathbf{10})^*)$ in its lower.



Problem 3. Give a CFG for the language of the regexp $\mathbf{101010}(\mathbf{11})^*$.

Solution. $S \to 101010 \mid 101010A,\ \ A \to 11 \mid 11A$ does it.

Problem 4. Let Σ be an alphabet. An ordered Σ-tree is a (possibly empty) finite ordered[2] tree whose nodes are labelled by symbols from Σ. Let Σ^{T} denote the set of all ordered Σ-trees. A

[2] A tree is *ordered* if a linear ordering is declared for the childs of any node, that is, if they can be uniquely listed "from left to right".

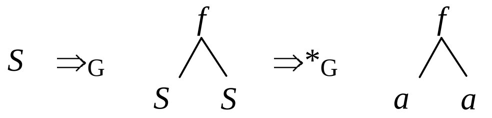
tree language[3] over Σ is a subset of Σ^{T}. Invent a generalization of context-free grammars that can describe tree languages and formally define the grammar and the tree language denoted by such a grammar. Demonstrate your definition by giving a concrete example of a tree grammar and a simple derivation of an ordered Σ-tree in that grammar. *Note.* The purpose of this problem is not to find "the correct" definition of tree grammars – you are free to invent what you find intuitive – but to demonstrate that you can distil intuitive concepts into formal definitions.

Solution. Here is one possibility:

Definition (tree grammars) A tree grammar is a quadruple $G = (V, \Sigma, P, S)$, where V is a finite set of variables, Σ is a set of terminals, P is a finite set of rules of the form $A \rightarrow (\alpha, a)$ where $\alpha \in (V + \Sigma)^*$ and $a \in \Sigma$, and $S \in V$ is the start symbol. P must include at least one rule whose l.h.s. is S. V and Σ are disjoint.

Definition. (derivations and tree languages). For a tree grammar G, define a binary relation $\Rightarrow_G \subseteq (V + \Sigma)^{\mathrm{T}} \times (V + \Sigma)^{\mathrm{T}}$ by $t \Rightarrow_G s$ iff t has a leaf node n labelled with a variable A, and $A \rightarrow (\alpha, a) \in P$, such that s is the same tree as t except that the node n is re-labelled with a and has as many new child nodes (each a leaf in s) as the length of α, and these child nodes are labelled with the symbols from α, from left to right. Let $\Rightarrow^*_G$ be the transitive closure of $\Rightarrow_G$. If $t \Rightarrow^*_G s$, we say that s can be *derived* from t. We identify S with the single-node tree labelled by S and define *the tree language* of G by $L(G) = \{t \in \Sigma^{\mathrm{T}} \mid S \Rightarrow^*_G t\}$.

Example: Let $V = \{S\}$, $\Sigma = \{a, f\}$, $P = \{S \rightarrow (SS, f), S \rightarrow (\varepsilon, a)\}$. Then



is a derivation of a Σ-tree in G.

Problem 5. List all subformulas and terms that occur in

$$((Qx \vee \exists x\, \forall y\, (\neg Pffxa \rightarrow Qa)) \vee Rxxa),$$

where Q is a unary predicate symbol, P is a binary predicate symbol, f is a unary function symbol, a is a constant symbol and R is a ternary predicate symbol. Determine for each occurrence of a variable whether it is free or bound.

Solution. The terms that occur are a, x, y (arguably, not rigoroously covered by our definitions), fx, and ffx. The subexpressions are Qx, $Pffxa$, $\neg Pffxa$, Qa, $(\neg Pffxa \rightarrow Qa)$, $\forall y\, (\neg Pffxa \rightarrow Qa)$, $\exists x\, \forall y\, (\neg Pffxa \rightarrow Qa)$, $(Qx \vee \exists x\, \forall y\, (\neg Pffxa \rightarrow Qa))$, $Rxxa$, $((Qx \vee \exists x\, \forall y\, (\neg Pffxa \rightarrow Qa)) \vee Rxxa)$. Free/bound occurrence, from left to right: x free, x bound [in $Pffxa$], x twice free [in $Rxxa$]. y does not occur at all according to our definition!

[3] Tree languages are an important field of theoretical computer science with many applications; the concept of tree languages used in that field is slightly more involved (it uses symbols with arity) than the simplistic concept used here.

Problem 6. Prove or disprove the following claims, where < is a binary relation symbol, *f* a unary function symbol and *a* a constant symbol (shorthand and infix notation is used):

1. $\{\forall x\, y\, z\, ((x < y \wedge y < z) \rightarrow x < z)\} \vDash \exists x\, \forall y\, x < y$
2. $\{\forall y\, fa < y\ \} \vDash \exists x\, \forall y\, x < y$

If you feel the need to describe an *S*-structure, explicitly present the carrier and the interpretation of the symbols as sets. If you feel the need to invoke derivation in the sequence calculus, rigorously follow the scheme from the lecture notes (as in the derivation of the chain rule).

Solution. 1. is false. A simple counterexample is, for instance, the *S*-structure $A = (\{0\}, <^A, f^A, a^A) = (\{0\}, \varnothing, \{(0,0)\}, 0)$. Clearly $A \vDash \forall x\, y\, z\, ((x < y \wedge y < z) \rightarrow x < z)$, because the antecedent of the implication is void, but likewise clearly not $A \vDash \forall x\, \exists y\, x < y$.
2. is true, as can be proven with our sequence calculus:

1.	$\forall y\, fa < y$	$\forall y\, fa < y$	(Pre)
2.	$\forall y\, fa < y$	$\exists x\, \forall y\, x < y$	($\exists$Con) applied on 1.
			(Note: $\forall y\, fa < y = \forall y\, x < y\, \frac{fa}{x}$)