Collected Exercises and Exam Sheets with Solutions for ACS1, Fall 03

Exercises for ACS 1, Fall 2003, sheet 1

Return solutions in paper form on Wednesday Oct. 1, in the lecture

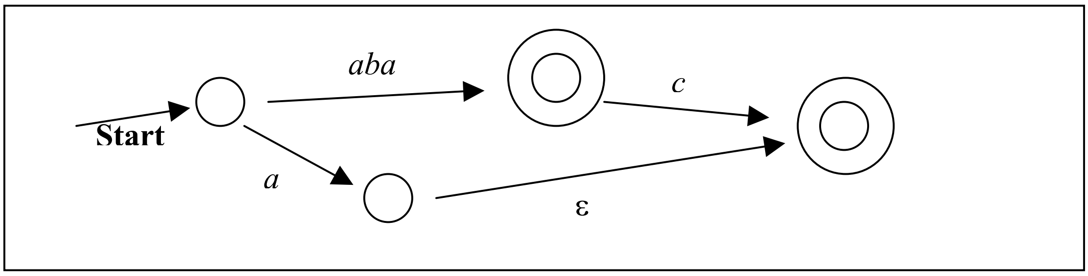
Note: a maximum of 100 points is accredited for this sheet.

Exercise 1 (30 points). Describe a method which directly transforms an ε-**NFA** into an equivalent **NFA**, by eliminating the ε-transitions.

Solution A: 1. Delete all reflexive ε-cycles . 2. merge all nodes that are connected by ε-cycles of length > 1. 3. Pick one ε-transition which has no precursor, and which goes from node *i* to *j*. Take all transitions into *i*, duplicate them and redirect the duplicates to *j*. Delete the ε-transition. Repeat until all ε-transitions are gone.

Solution B: Take the ε-**NFA** as is, and for any symbol *a* define $\delta(q,a)$ of the new NFA as the extended transition function δ^(q,a) of the ε-**NFA.** The new set of accepting states is the old one, except possibly adding the starting state, if in the original ε-**NFA** an accepting state lies in the Closure of the starting state.

Exercise 2 (30 points). A *transition system* is a generalization of ε-NFAs, in which additionally transitions labelled with words of length greater than 1 are admitted. A transition graph of a transition system might look like this:



Give a formal definition of transition systems (20 points) and their accepted languages and prove that the languages accepted by transition systems are accepted by DFAs (10 points).

Solution (for proof): simplest proof is to transform transition system into ε-NFA by changing each word-labelled arc into a sequence of symbol-labelled arc.

Exercise 3 (30 points). Give a regular expression that tries to catch in an electronic ad newspaper all contact ads where a man looks for a woman (or, if you prefer, the other way round), like "Lonely man looks for lovely woman, blahblab....", or "My nest is so empty ...blah blah... when will she call me ...". You may assume that each ad (of whatever sort) is enclosed by a blank line (that is, two newline commands \nl). Start with some simple initial regex, give an example of a valid contact ad that it does not catch or an unwanted contact ad of different type that it does catch, improve the initial regex to deal with that example, etc., through at least 8 cycles of improvement.

Exercise 4. Give (a) a formal set description of the kind $L = \{w \in \{0,1\}^* \mid w = u1\}$, (b) an ε-NFA and (c) a regex (using UNIX style if you like) for the following languages:

(30 points) L_1: The set of words over $\{0,1\}$ that do not contain 101 as a subword
(30 points) L_2: The set of words over $\{0,1\}$ with equal numbers of 0's and 1's such that no prefix has two more 0's than 1's, nor two more 0's than 1's (for (a), assume that the function f_0 counts the number of 0's in a word and the function f_1 that counts the number of 1's in a word)
(30 points) L_3: The set of nonempty words over $\{0,1\}$ whose number of 0's is divisible by 3 or divisible by 5.

In each case, prove that your solution is correct.

One solution for L_1: (b): first write a NFA that accepts the complement of L_1, transform it into a DFA, exchange accepting / nonaccepting states. (c): derive regex from your DFA or build regex directly from admissible concatenations of non-101 three-symbol words (taking care of leading two- and one-symbol words and the empty word)
One solution for L_2: Show first (by induction over even wordlength) that words of even length must have identical numbers of 1's and 0's. The rest is easy.
One solution for L_3: (b): design a DFA A1 whose states count the number 0's read in and accepts all words with $3n$ zeros, and another similar DFA A2 accepting words with $5n$ zeros. Join A1 and A2 in a NFA. (c): create regex from your NFA or write down directly, as $((1^*01^*)^3)^+ | ((1^*01^*)^5)^+$

Exercises for ACS 1, Fall 2003, sheet 2

Return solutions in paper form on Wednesday Oct. 15, in the lecture

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

Exercise 1 (20 points). Consider the DFA A given by the following transition table:

	0	1
$\rightarrow q_1$	q_2	q_3
q_2	q_3	q_1
$*q_3$	q_3	q_2

Construct the transition graph and construct a regular expression that describes $L(A)$, by eliminating state q_2. Provide the transition graph of the 2-state automaton that you obtain after elinimatint state q_2.

Solution: I found the regex (00 + (1+01)(01)*(1+00))*(1+01)(01)*

Exercise 2 (10 points). Convert 00(0+1)* to an ε-NFA.

Exercise 3. Prove (by exploiting Theorem 4.3 and possibly some further arguments, or by construction of minimal DFAs) or disprove (by using Theorem 4.4):

(i) $(R + S)^* S = (R^*S)^*$ [10 points]

(ii) $(RS + R)^*R = R(SR + R)^*$ [30 points]

Exercise 4 (10 points). Prove that the language $\{0^n \mid n \text{ is a power of } 2\}$ is not regular.

Exercise 5 (10 points). Show that the regular languages are closed under the *min* operation, where $min(L) = \{w \mid w \text{ is in } L, \text{ but no proper prefix of } w \text{ is in } L\}$.

Solution: Take a DFA for L and delete all arcs that leave accepting states.

Exercise 6 (30 points). Suppose that L is any language, not necessarily regular, over the alphabet $\{0\}$. Show that L^* is regular. Hint: you may use the fact from number theory that if integers $k_1, \ldots, k_n$ have greatest common divisor d, then for some l, all integers which are greater than l and which are divisibly by d can be written as a sum $\alpha_1 k_1 + \ldots + \alpha_n k_n$, where the α_i are nonnegative integers.

Exercises for ACS 1, Fall 2003, sheet 3: Solution sheet

Note: Solutions given here are sometimes more detailed than would be required for full grades.

Exercise 1 (30 points). A word w made from symbols "(" and ")" is called *balanced* if iterated deletion of substrings "()" ends in the empty word. Prove with a proof of the kind given in example 6.3 in the lecture notes: The language of the grammar $B \to BB \mid (B) \mid \varepsilon$ is the language of all balanced words.

Solution. Call our grammar G and call the language of balanced words L_b.We have to show (a) if $w \in L_b$, then $w \in L(G)$ and (b) if $w \in L(G)$, then $w \in L_b$.

(a): Let $w \in L_b$, $|w| = 2n$, $w = s_1,..., s_{2n}$, where $s_i \in \{(,)\}$. Then there exists a sequence of deletions $d_1, ..., d_n$ of innermost "()" that at the end deletes w altogether. Call a pair (s_i, s_j) the balance pair of d_k if (s_i, s_j) is deleted by d_k. For showing that $w \in L(G)$ we may without loss of generality assume that (s_1, s_{2n}) is a balance pair of d_n. [If it isn't, consider $w' = (w)$, where we may assume this, and show that w' is in $L(G)$; conclude that then also w is in $L(G)$ because if we have a derivation of w' then its first derivation must be $B \Rightarrow (B)$ and the remaining derivations for w' give a derivation for w]. We call a balance pair (s_i, s_j) a *descendant* of (s'_i, s'_j) if $s'_i < s_i$ and $s_j < s'_j$, and write $(s_i, s_j) < (s'_i, s'_j)$ for this. Clearly descendance is transitive and anti-symmetric, has (s_1, s_{2n}) as maximal element, and balance pairs of form (s_i, s_{i+1}) are the minimal elements of $<$. Hence the balance pairs are organized in a *tree* T by $<$, where the nodes are the balance pairs. The leaves are the minimal balance pairs and the root is (s_1, s_{2n}). When we annotate a node N of this tree by the bracket expression obtained from assembling all the balance pairs below and including N, we get a (unique) hierarchic tree representation of w, as in the following figure for $w = (_1(_2)_3(_4(_5)_6)_7)_8$:

$(_1(_2)_3(_4(_5)_6)_7)_8$

$(_2)_3$ $\quad$ $(_4(_5)_6)_7$

$(_5)_6$

We call these node annotations the *subexpressions* of w. For a node N let e[N] denote the subexpression annotating N.

In order to show that w can be generated by G, observe first that clearly $B \Rightarrow^* (B^n)$ for $n \geq 1$. We show by induction on the structure of trees that w can be generated by G, as follows. Claim: each subexpression in T can be generated by G (and therefore, the root w can be generated by G).

Basis: all subexpressions at the leaves of the tree can be obviously generated by $B \Rightarrow (B) \Rightarrow ()$.

Induction: consider some non-leave node N of T with direct tree childs $N_1, ..., N_k$. Then e[N] = (e[N_1] ... e[N_k]). By induction we know that each e[N_i] can be generated by G. Then e[N] can be generated by G, too, by first generating $B \Rightarrow^* (B^k)$ and then expanding each of the B on the rhs. by the generation for the corresponding e[N_i].

Note. For getting full grades, this detailed treatment is not required. You may have used the tree representation of a balanced parenthesis expression without deriving it – it is basic knowledge for computer scientists.

(b) We now show that if $w \in L(G)$, then $w \in L_b$. Induction on the length n of derivations of w.

Basis: $n \leq 2$: the only derivation in G of length at most 2 is $B \Rightarrow (B) \Rightarrow ()$ [can be found by systematic construction of all derivations of length at most 2], which is a balanced.

Induction: Assume we have derived w in G with a derivation D of length $n > 2$, and all w' derivable in G with derivations of length smaller than n are in L_b. The first step in D is $B \Rightarrow BB$ or $B \Rightarrow (B)$. In the first case, $w = uv$ with $u, v \in L_b$ by induction. Thereby, $w \in L_b$ because we can ultimately delete u and v separately by deleting innermost (), and thereby delete $uv = w$ altogether. In the second case, $w = (u)$ with $u \in L_b$ by induction, thus we can delete w by first deleting u (by deleting all innermost ()) and then deleting the outermost () of w.

Exercise 2. Consider the language $L = \{w \in \{a, b\}^* \mid w \text{ is not of the form } vv\}$.

a. (20 points) Show that L is not regular.
b. (30 points) Show that L is a context-free language.
c.

Solution. a. First use the pumping lemma to show that $L^c = \{w \in \{a, b\}^* \mid w \text{ is of the form } vv\}$ is not regular. That's a routine argument: let n be the PL constant. Assume L^c is regular. Consider the word $a^n b^n$. It can be written as xyz with $|xy| \leq n$, $y \neq \varepsilon$. Because $|xy| \leq n$, $y = a^k$ for some $k > 0$. By PL, $a^{n-k} b^n \in L^c$. Contradiction. Thus L^c is not regular. Because the regular languages are closed under complement, L is not regular either.

First observe that

$$
\begin{aligned}
L = L_1 \cup L_2 \cup L_3 :=\ & \\
& \{uavxby \in \{a, b\}^* \mid |u| = |x| \geq 0 \text{ and } |v| = |y| \geq 0\ \} \cup \\
& \{ubvxay \in \{a, b\}^* \mid |u| = |x| \geq 0 \text{ and } |v| = |y| \geq 0\ \} \cup \\
& \{v \in \{a, b\}^* \mid v \text{ has uneven length}\}.
\end{aligned}
$$

L_3 is clearly regular (for complete completeness, you may easily construct a DFA for this language) and thereby context-free. L_2 can be re-written as $\{uavxby \in \{a, b\}^* \mid |u| = |v| \geq 0$ and $|x| = |y| \geq 0\}$. Observing this, it is easy to find a CFG for this language, for instance, $S \to AB$, $A \to a \mid aAa \mid bAa \mid aAb \mid bAb$, $B \to a \mid aBa \mid bBa \mid aBb \mid bBb$ does it. By a similar argument, L_3 is context-free, too. So we have three context-free languages. Without loss of generality ("w.l.o.g."), you may assume that we have grammars for these languages with disjoint variable sets and start variables S_1, S_2, S_3 and production sets P_1, P_2, P_3. Then you get a CFG for L by joining P_1, P_2, P_3 and adding a new start symbol S and the rule $S \to S_1 \mid S_2 \mid S_3$. Thus, L is context-free.

Exercise 3 (30 points). You might want to boost the power of grammars by allowing regular expressions (over the joined alphabet of terminals and variables) for the body. For instance, it would then be allowed to write the rule $A \to ((a + b)^*B)(C + a)^*$. You would use such productions in two steps: first, replace the regular expression by any word described by this expression (in this example, for instance, you might get $A \to aaBCa$ in this first step), second, use the ordinary production that you now obtained in the ordinary way in derivations. Because a regular expression can define an infinite number of words, this amounts to a shorthand notation for an infinite set of ordinary production rules. Show that the languages defined by such "boosted" grammars G are again the context-free languages. Note: The HMU book gives a direct proof of this claim on page 202/203, using structural induction over the form of regex's. You should give a different proof that uses the fact that regular languages are context-free.

Solution: Let $G = (V, T, S, P)$ be a "boosted" grammar, where P contains rules that have regex's as bodies. Let $A \to e$ be such a rule (e is a regex over $\Sigma = V \cup T$). We show how to replace it by a finite set of ordinary production rules. First introduce another alphabet $\Sigma' = V' \cup T'$ which is a variant of Σ obtained by replacing each $A \in V$ by a new symbol A' and each $a \in T$ by a'. Denote by e' the regex obtained from e by replacing variables from Σ with their counterparts from Σ'. Let $G'(e')$ be a CFG for the language of e' [here you need the fact that regular languages are context-free], where without loss of generality the variables of G' are disjoint from the variables of G. Note that the set of terminal symbols of G' is Σ'. Let S' be the start symbol of G'. Now replace $A \to e$ by the following set of rules: (i) a rule $A \to S'$, (ii) the rules from $G'(e')$, (iii) for each $x' \in \Sigma'$, a rule $x' \to x$. Then G with the new "ordinary" rules (i) – (iii) instead of the boosted rule $A \to e$ describe the same language. Repeat for all boosted rules.

Exercise 4. (Writing context-free grammars in two formats for an XML document). Consider the XML document from Example 6.4 of the script.

a. (20 points) Write a CFG in Backus-Naur format (the format used in Fig. 6.1 in the script) that describes XML documents of the kind given in Example 6.4 (that is, that XML document should lie in the language of the grammar you write). Consult http://dublincore.org/documents/2002/07/31/dcmes-xml/ for guidance – actually, all you have to do is to pick a suitable subset from the grammar given in the URL.
b. (20 points) Write a DTD for RDF documents as the one from the example, that is, the particular RDF document given in Example 6.4 should fall into the class of XML documents described by your DTD. Ignore the `xmlns` declarations of the example document and stick to the toy example 6.4 from the tutorial. (See http://dublincore.org/documents/2002/07/31/dcmes-xml/dcmes-xml-dtd.dtd for a DTD that would cover the second block in the document from above – but because this DTD is very detailed, it is also confusing).

Solution. First I must apologize for a serious typo in the original exercise text. The "example 6.4" mentioned in the text should correctly read as "example 6.8" in the first to occurences [this happened because I re-arranged the script after writing the exercise sheet – sorry.]. Only with this adjustment made, the exercise makes much sense. The example 6.8 was the following RDF document:

```
<rdf:RDF xmlns:rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns#'
 xmlns:iX='http://ns.adobe.com/iX/1.0/'>

 <rdf:Description about=''
  xmlns='http://ns.adobe.com/pdf/1.3/'
  xmlns:pdf='http://ns.adobe.com/pdf/1.3/'>
  <pdf:CreationDate>2003-09-17T10:50:16Z</pdf:CreationDate>
  <pdf:Author>Herbert Jaeger</pdf:Author>
  <pdf:Creator>Acrobat PDFMaker 5.0 for Word</pdf:Creator>
  <pdf:Title>Exercises for ACS 1, Fall 2003</pdf:Title>
 </rdf:Description>

 <rdf:Description about=''
  xmlns='http://purl.org/dc/elements/1.1/'
  xmlns:dc='http://purl.org/dc/elements/1.1/'>
  <dc:creator>Herbert Jaeger</dc:creator>
  <dc:title>Exercises for ACS 1, Fall 2003</dc:title>
 </rdf:Description>

</rdf:RDF>
```

The Backus-Naur format for grammars employs special characters "<" and ">". This same character as used in the target RBF documents, too. This means that we have to use another special escape character within the Backus-Naur specification to distinguish the target "<" from the Backus-Naur "<". Using \> and \> for the target symbols, a Backus-Naur grammar for documents of this kind might look as follows.

```
<RDFdocument> ::= \<rdf:RDF xmlns:rdf= '<string>' xmlns:iX='<string>'\>
      <descriptions> \</rdf:RDF\>
<string> ::= <char><string> | <char>
<char> ::= ε | a | b | ... |1 | ... | 9 | 0 | , | .. (same for all ASCII symbols)
<descriptions> ::= <adobeDescription> <purlDescription> (plus other types of descr.)
<adobeDescription> ::= \<rdf:Description about=''
      xmlns='http://ns.adobe.com/pdf/1.3/'
      xmlns:pdf='http://ns.adobe.com/pdf/1.3/'\>
      <adobeFields>
      \</rdf:Description\>
<adobeFields> ::=
      <adobeCreationDateField><fieldsAfterAdobeCreationDateField> |
      <fieldsAfterAdobeCreationDateField>
<fieldsAfterAdobeCreationDateField> ::=
      <adobeAuthorField><fieldsAfterAdobeAuthorField> |
      <adobeAuthorField>
<fieldsAfterAdobeAuthorField> ::=
      <adobeCreatorField><fieldsAfterAdobeCreatorField> |
      <adobeCreatorField>
<fieldsAfterAdobeCreatorField> ::=
      <adobeTitleField><fieldsAfterAdobeTitleField> |
      <adobeTitleField> | ε
<adobeCreationDateField> ::= \<pdf:CreationDate\> <string>
      \</pdf:CreationDate\>
...
<adobeTitleField> ::= \<pdf:Title\> <string> \</pdf:Title\>
... (same for purlDescription)
```

This grammar supposes that a RDF document must have descriptions for adobe, purl, and possible others in that order, possibly empty. It further supposes that adobe descriptions have entries for creation data, ..., title, at most one them, in that order, possibly empty. I don't know about the true standards for RDF documents; other assumptions would be reflected in variants of such grammars.

b. A DTD for RDF documents, ignoring the xmlns parts, might look as follows (note that ? means "zero or one occurrence of"):

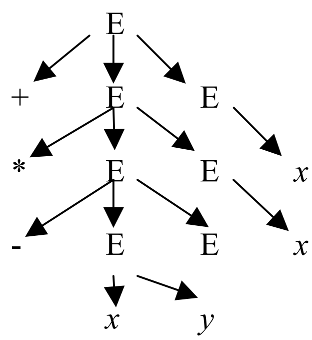
```
<!DOCTYPE RDFdocument [
      <!ELEMENT rdf:RDF (rdf:Description)*>
      <!ELEMENT rdf:Description
            (pdf:CreationDate? pdf:Author? ... pdf:Title? |
            dc:creator? ... dc:title?
            )>
      <!ELEMENT pdf:CreationDate (\#PCDATA)>
...

      <!ELEMENT dc:title (\#PCDATA)>
]>
```

Exercises for ACS 1, Fall 2003, sheet 4: Solutions

Exercise 1. The following grammar generates "prefix" expressions of the kind $+*-xyxx$: $E \to +EE \mid -EE \mid *EE \mid x \mid y$.

a) (10 points) Find a leftmost derivation and a parse tree for $+*-xyxx$.
b) (30 points) Prove that this grammar is unambiguous.

Solution. **a.** Leftmost derivation: $E \Rightarrow +EE \Rightarrow +*EEE \Rightarrow +*\text{-}EEEE \Rightarrow +*\text{-}xEEE \Rightarrow +*\text{-}xyEE \Rightarrow +*\text{-}xyxE \Rightarrow +*\text{-}xyxx$. Parse tree:



b. First we show that each word in $L(G)$ has a unique leftmost derivation.

General comment: If one has to show a statement of the general form "for every A there exists exactly one B such that blabla", then the proof typically has two parts: (a) show that for every A there exists a B such that blabla, (b) if for A there exist B such that blabla and B' such that blabla, then B ≠ B'. This argument is often made by contradiction, that is, assuming B = B' is led into a contradiction.

So we here too first have to show that each word $w \in L(G)$ has a leftmost derivation. But we already know that from a theorem early in the lecture, which said among other things that if a word can be derived in a grammar, then it also has a leftmost derivation.

So now assume that for $w \in L(G)$ we have two different leftmost derivations, say $E = \alpha_0 \Rightarrow \alpha_1 \Rightarrow \alpha_2 \Rightarrow ... \Rightarrow \alpha_l = w$ and $E = \beta_0 \Rightarrow \beta_1 \Rightarrow \beta_2 \Rightarrow ... \Rightarrow \beta_k = w$. For some $0 < n \leq \min(k, l)$ it must hold that $\alpha_n \neq \beta_n$ but $\alpha_i = \beta_i$ for all $i < n$. $\alpha_{n-1} = \beta_{n-1}$ must contain at least one variable because otherwise the derivation couldn't go further to β_n. Let α_{n-1} ($= \beta_{n-1}$) be of form $\gamma E \gamma'$, where E is the first E in α_{n-1} from the left. Now, in both derivations, this E must be replaced by applying some production, because both derivations are leftmost. But all productions that replace E are of the form $E \to t$ or $E \to tEE$, where t is a terminal. Importantly, different productions introduce different terminals t. Thus, $\alpha_n \neq \beta_n$ implies that different rules are used to replace the E in $\gamma E \gamma'$, which means that α_n starts with γt and β_n starts with $\gamma t'$, where $t \neq t'$. This implies that the two derivations cannot yield the same word, a contradiction.

So every word in the language of our grammar has a unique leftmost derivation. Now if the grammar would be ambiguous, some word would possess two different parse trees. But if two parse trees are different, then also the leftmost derivations obtained by traversing the parse tree in a left-first, depth-first way would differ, a contradiction to our previous finding.

it must hold that

Basis. $|w| = 1$: then $w = x$ or $w = y$, and in each case there is only one possible derivation, which is a leftmost derivation.
Induction. Assume statement holds for all $w \in L(G)$, $|w| \leq n$.

Exercise 2. (30 points) Design a PDA for the language L of words that contain twice as many 0's as 1's (including ε). Specify your PDA by its transition function, and describe the principles behind your design in intuitive terms. You may choose acceptance by empty stack or accepting states, whatever you find more convenient.

Solution: Call a word "equilibrated" if it has twice as many 0's as 1's. Use two stack symbols X, Y. At any time, the stack either is empty, (then you may accept by empty stack), or contains only Z_0, or contains only X's on top of Z_0, or contains only Y's on top of Z_0. Semantics of X's: "if there are n X's, the subword that has been read in so far has an excess of $n/2$ 1's to become equilibrated". Semantics of Y's: "if there are n Y's on the stack, the subword that has been read so far has an excess of n 0's to be equilibrated". [You might also say, each Y is "worth" one 0 and each X is "worth" 1/2 1]. If a 1 is read while the top of the stack is X or Z_0, push XX on the stack. If a 0 is read while the top of the stack is X, pop one X. If a 0 is read while the top of the stack is Y or Z_0, push Y on the stack. If a 1 is read while the top of the stack is Y, pop two Y's. This latter action must be coded as a sequence of two transitions that pop two Y's consecutively. Finally, there is one transition that simply pops Z_0 on ε input, leading to empty (=accepting) stack.

The transition function would look as follows (let q_0 be starting state, and $Q = \{ q_0, q_1 \}$.

$\delta(q_0, 1, Z_0) = \{(q_0, XXZ_0)\}$
$\delta(q_0, 1, X) = \{ (q_0, XXX) \}$
$\delta(q_0, 0, Z_0) = \{ (q_0, YZ_0) \}$
$\delta(q_0, 0, Y) = \{ (q_0, YY) \}$
next line contains the "pop-two-Y" cycle. Note that the case where the cycle hits the stack bottom must be caught and one X be pushed worth half the 1 that started the cycle.
$\delta(q_0, 1, Y) = \{ (q_1, \varepsilon)\}$ $\quad$ $\delta(q_1, \varepsilon, Y) = \{ (q_0, \varepsilon)\}$ $\quad$ $\delta(q_1, \varepsilon, Z_0) = \{ (q_0, XZ_0) \}$
$\delta(q_0, 0, X) = \{ (q_0, \varepsilon)\}$
$\delta(q_0, \varepsilon, Z_0) = \{ (q_0, \varepsilon)\}$

Exercise 3 (30 points). Show that if P is a PDA accepting by empty stack, then there is a PDA P' with only two stack symbols (plus Z_0) that accepts the same language, also by empty stack. *Hint*: this is taken from the HMU book. In the book the hint is given to binary-code the original stack symbols from P. That is one option, but not the only one.

Solutions (sketch): As an alternative to the suggestion of HMU, code original stack symbols by sequences of X's of different length and use the other stack symbol Y as a delimiter. In each case, one must install appropriate state sequences in the "control unit" of P' that have to be traversed deterministically to detect what coded stack symbol is currently being read or

written. More precisely, if the original stack alphabet Γ had n symbols (excluding Z_0) X_1, ..., X_n, code them by stack words YX^i. (i = 1, ..., n). For an original stack contents α, denote by code(α) the stack word obtained by replacing all X_i in a by YX^i. An original transition rule of the form $\delta(q, a, X_i) = (q', \alpha)$ is replaced a sequence of rules
$\delta(q, \varepsilon, X) = (p_1, \varepsilon)$, $\delta(p_1, \varepsilon, X) = (p_2, \varepsilon)$, ..., $\delta(p_{i-1}, \varepsilon, X) = (p_i, \varepsilon)$, $\delta(p_i, a, Y) = (q', \text{code}(\alpha))$, where for every original state q and every original stack symbol X_i, one such sequence is installed with new states. An original transition where Z_0 is read and some nonempty α that is different from Z_0 is pushed must be replaced by another transition which is like the original one but pushes code(α) instead of α.

If binary coding is used (as suggested by HMU book), all binary code-stack-symbol-strings for original symbols must have equal length, because there is now no delimiter symbol available.

Exercise 4 (20 points). Convert the grammar $S \to aAA$, $A \to aS \mid bS \mid a$ into a PDA that accepts the same language by empty stack.

Solution: Using the construction from theorem 7.2, we get PDA rules for an equivalent PDA by using S as the stack start symbol and only a single state q:

$\delta(q, \varepsilon, S) = \{(q, aAA)\}$
$\delta(q, \varepsilon, A) = \{(q, aS), (q, bS), (q, a)\}$
$\delta(q, a, a) = \{(q, \varepsilon)\}$
$\delta(q, b, b) = \{(q, \varepsilon)\}$

Exercises for ACS 1, Fall 2003, sheet 5

Return solutions in paper form on Thursday Nov. 27, in the lecture

Note: a maximum of 100 points is accredited for this sheet.
Remark: the first 5 exercises are simple.

Exercise 1 (5 points) Is the PDA $P = \{\{p, q\}, \{0,1\}, \{X, Z_0\}, \delta, q, Z_0\}$ given by the transitions below deterministic? Either show that it meets the definition of a DPDA or find a rule / some rules that violate determinism.

1. $\delta(q, 1, Z_0) = \{(q, XZ_0)\}$
2. $\delta(q, 1, X) = \{(q, XX)\}$
3. $\delta(q, 0, X) = \{(p, X)\}$
4. $\delta(q, \varepsilon, X) = \{(p, \varepsilon)\}$
5. $\delta(p, 1, X) = \{(p, \varepsilon)\}$
6. $\delta(p, 0, Z_0) = \{(q, Z_0)\}$

Solution: It is not deterministic because rules 2. and 4. violate the second condition in the definition of DPDAs.

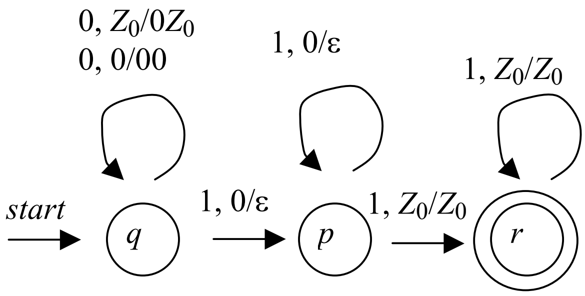
Exercise 2 (10 points) Give a deterministic PDA to accept $\{0^n1^m \mid 1 \le n \le m\}$. Describe your PDA in words and specify its transition function and give a transition diagram.

Solution: One possible PDA: Accept by final state. Let r be the accepting state. First read in 0's in starting state q, counting them by putting 0's on the stack. Go to another state p when the first 1 is encountered and pop 0's when reading 1's. End dead when the bottom stack symbol is seen while there is still a 1. Go to accepting state r when the bottom is seen. In r, read in more 1's if there are any, staying in r.

$P = \{\{p, q, r\}, \{0,1\}, \{0, Z_0\}, \delta, q, Z_0\}$

Transition function and diagram:

1. $\delta(q, 0, Z_0) = \{(q, 0Z_0)\}$
2. $\delta(q, 0, 0) = \{(q, 00)\}$
3. $\delta(q, 1, 0) = \{(p, \varepsilon)\}$
4. $\delta(p, 1, 0) = \{(p, \varepsilon)\}$
5. $\delta(p, 1, Z_0) = \{(r, Z_0)\}$
6. $\delta(r, 1, Z_0) = \{(r, Z_0)\}$



Exercise 3 (40 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.

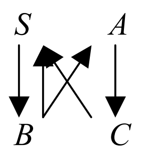
$$S \to 0A0 \mid 1B1 \mid BB$$
$$A \to C$$
$$B \to S \mid A$$

$C \to S \mid \varepsilon$

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 $S \to BB$ add $\{S \to BB, S \to B\}$ 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'

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

(ii) **a.** Finding unit pairs: PAIRS(1) = $\{(A, A), (B, B), (C, C), (S, S)\}$, PAIRS(2) = $\{(A, A), (B, B), (C, C), (S, S), (S, B), (A, C), (B, S), (B, A), (C, S)\}$, PAIRS(3) = $\{(A, A), (B, B), (C, C), (S, S), (S, B), (A, C), (B, S), (B, A), (C, S), (S, A), (A, S), (B, C), (C, B)\}$, 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:



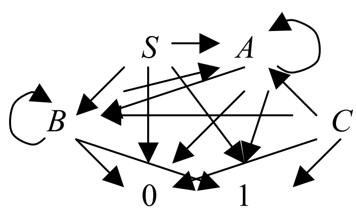
b. Adding to P'' all productions of the form $A \to \alpha$, where $B \to \alpha$ is a non-unit production in P' and (A, B) is a unit pair, yields P'' =

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

(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 $P''' =$

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

(iv) In the last step, we obtain a CNF grammar by carrying out the two steps given in the proof of theorem 8.5 in the lecture notes.

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

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

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 $P_{\text{CNF}} =$

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

Exercise 4 (10 points) In the construction of CNFs, we eliminated ε-production, unit pairs, and useless symbols. One step in the eliminiation of ε-productions was to find all nullable variables. This can be done inductively by constructing sets NULL(1), NULL(2) etc. Similarly, the eliminiation of unit pairs included a step where all unit pairs of variables had to be found. Again, this was done inductively by constructing PAIRS(n). One step in the elimination of useless symbols was to find all reachable symbols. In the script, a graph-theoretical method is sketched. Your task here: describe an inductive procedure for finding all reachable symbols of a grammar G, by constructing sets REACH(1), REACH(2), ... and prove correctness of your construction.

Solution. Inductive procedure to compute reachable symbols: Let $G = (V, T, P, S)$ be the grammar for which we want to find the reachable symbols. Put REACH(1) = $\{S\}$. Construct REACH(n+1) from REACH(n) as follows. For each variable symbol A from REACH(n) and each production $A \to \alpha$ from P and each symbol X in α, add X to REACH(n), to make REACH(n+1). Terminate when REACH(n+1) = REACH(n).

Proof of correctness: we have to show that this procedure (i) finds all reachable symbols, and (ii) that all found symbols are reachable.

(i): Let X be reachable, that is, there exists a sequence of derivations $S \Rightarrow^* \alpha X\beta$ for some $\alpha, \beta \in (V \cup T)^*$, consisting of n-1 single derivation steps, that is, $S = \alpha_1 \Rightarrow \alpha_2 \Rightarrow \ldots \Rightarrow \alpha_{n-1} \Rightarrow \alpha_n = \alpha X\beta$. We show by induction on this sequence that all symbols occurring in this sequence are found by the algorithm, and more strongly, that all symbols in α_1 are contained in REACH(i). Basis: all symbols in α_1 are found and in REACH(1): clear because these symbols are $\{S\}$ and that's REACH(1). Induction: assume all symbols in α_i are in REACH(i). The symbols from α_{i+1} are either kept unchanged from α_i, or are generated from some variable symbol A in α_i by a production from P. In both cases, they are in REACH(i+1). – Thus, the algorithm finds all reachable symbols.

(ii) By induction we show that all symbols in REACH(n) are reachable, for all n. Basis $n = 1$: clear. Induction: Assume REACH(n) contains only reachable symbols. Let X be a symbol that first appears in REACH(n+1). Then there must be some variable symbol A in REACH(n) and a production of the form $A \to \gamma X\delta$. But each variable symbol A in REACH(n) can be derived from S by some $S \Rightarrow^* \alpha A\beta$. Therefore, $S \Rightarrow^* \alpha\ \gamma X\delta\ \beta$, that is, X is reachable. Thus REACH(n+1) contains only reachable symbols.

Exercise 5 (20 points) Show that $L = \{a^m b^m c^k \mid k \leq m\}$ is not context-free (use the pumping lemma).

Solution: Assume L is context-free. Let n be the PL constant. Consider the word $w = a^n b^n c^n$. Then $a^n b^n c^n$ can be written as $x\underline{uyv}z$ such that

1. $|uyv| \leq n$,
2. $|uv| \geq 1$,
3. for all $i \geq 0$, $xu^i yv^i z \in L$.

Case 1: uyv fits in the first block a^n of w. Then by PL, $a^{n-|uv|}b^n c^n$ is in L, contradiction.
Case 2: uyv strikes both the a^n block and the b^n block of w. It cannot touch the c^n block. u must contain some a or v must contain some b (non-exclusive or). Then xu^0yv^0z has less than n a's or less than n b's and hence is not in L, contradiction.
Case 3: uyv fits in the second block b^n of w. Then by PL, $a^n b^{n-|uv|} c^n$ is in L, contradiction.
Case 4: uyv strikes both the b^n block and the c^n block of w. It cannot touch the a^n block. u must contain some b or v must contain some c (non-exclusive or). Then $xu^{n+1}yv^{n+1}z$ contains more b's than a's or more c's than a's (non-exclusive or) and hence is not in L, contradiction.
Case 5: uyv fits in the last block c^n of w. Then $xu^{n+1}yv^{n+1}z$ contains more c's than a's (non-exclusive or) and hence is not in L, contradiction.

Exercise 6 (50 points) Let L be a language. Define $half(L) = \{w \mid$ for some x such that $|x| = |w|$, $wx \in L\}$. Notice that odd-length words in L do not contribute to $half(L)$. Show that the context-free languages are not closed under *half*. (This is marked a very difficult problem in the HMU book, and I must agree; I did not find a solution within two hours. Some fraction of the 50 points will also be awarded for attempted but failed solutions, if the attempts are clearly explained).

Solution: a solution can be found on the Web, e.g. at
http://www.cis.ksu.edu/~howell/770s03/protected/sol8.pdf

Exercises for ACS 1, Fall 2003, sheet 6

Return solutions in paper form on Thursday Dec. 4, in the lecture

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

Exercise 1 (very simple, 10 points) Is *baaab* in the language of the grammar $S \to AB \mid BC, A \to BA \mid a, B \to CC \mid b, C \to AB \mid a$? Provide the CYK table and the answer.

Solution: the CYK table is

$\{S,C\}$				
$\{S, A, C\}$	$\{S,C\}$			
$\{\}$	$\{S,C,A\}$	$\{B\}$		
$\{S,A\}$	$\{B\}$	$\{B\}$	$\{S,C\}$	
$\{B\}$	$\{A,C\}$	$\{A,C\}$	$\{A,C\}$	$\{B\}$
b	a	a	a	b

and the answer is yes.

Exercise 2 (20 points, easy) Give an algorithm to decide whether $|L(G)| \geq 100$, for some CFG G.

Solution. First decide whether $L(G)$ is finite, using one of the known algorithms from the lecture. If not, then $|L(G)| \geq 100$ and the decision is done. If yes, transform G to CNF and determine the pumping lemma constant n. All words of $L(G)$ must have length shorter than n, because any word of length at least n could be pumped into infinitely many different other words from L. Construct all parse trees with at most $2(n - 1) - 1$ interior nodes. Any word of length less than n must have such a parse tree. Check whether among these parse trees there are at least 100 that yield different words. If yes, $|L(G)| \geq 100$, if no, no.

Exercise 3 (20 points, easy) 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).

a. John is the boyfriend of Mary.
b. John loves Mary.
c. If John is the boyfriend of Mary, then John loves Mary.
d. The boyfriend of any person is a man, and the person is a woman.
e. Everybody loves somebody.
f. If a man loves a woman, then the woman loves the man, or she doesn't.

Which of your propositions are tautologies, which are contradictions?

Solution. One possibility is to choose S containing constant symbols John, Mary; the unary predicates person, man, woman; the binary relation loves; the unary function boyfriend-of. We then put

a. boyfriend-of Mary = John
b. loves John Mary
c. boyfriend-of Mary = John $\rightarrow$ loves John Mary
d. $\forall x \forall y$ (person $x \rightarrow$ (($y =$ boyfriend-of $x \rightarrow$ man y) $\wedge$ (woman x)))
e. $\forall x$ (person $x \rightarrow \exists y$ loves xy)
f. $\forall x \forall y$ ((man $x \wedge$ woman $y \wedge$ loves xy) $\rightarrow$ (loves $yx \vee \neg$ loves yx))

f. is a tautology, none is a contradiction.

Exercise 4 (10 points, easy) For your symbol set S of the previous exercise, describe an S-structure in which all the statements of Exercise 3 hold.

Put A = {`John`, `Mary`}; this set contains two particular persons. Put person^A = {`John`, `Mary`}, man^A = {`John`}, woman^A = {`Mary`}, loves^A = {(`Mary, John`), (`John, Mary`)}, boyfriend-of^A = {(`Mary, John`)}.

Exercise 5 (10 points, easy). Give a very short S-expression φ that is equivalent to your S-expression ψ for Exercise 3f. Explain in words why. (Equivalent means: for every S-structure $\mathcal{A}$, it holds that $\mathcal{A} \models \varphi$ iff $\mathcal{A} \models \psi$. "Very short" means: containing at most 3 symbols from S and/or FOL generic symbols $=, \wedge, \vee, \neg, \rightarrow, \leftrightarrow, \exists$ or $\forall$.)

Solution. Because 3f. is a tautology, it is equivalent to any other tautology, the shortest of which is $x = x$.

Exercise 6 (20 points, easy to medium). **a.** (10 points) Show that for any φ, $\exists x \forall y \varphi \models \forall y \exists x \varphi$.
b. (10 points) Show that for a binary relation symbol R, $\forall y \exists x Rxy \models \exists x \forall y Rxy$ does not hold.

Solution. **a.** Let $(\mathcal{A}, \beta) \models \exists x \forall y \varphi$ for some $\mathcal{A}$ with domain A. We have to show that $(\mathcal{A}, \beta) \models \forall y \exists x \varphi$. There exists some $a \in A$, such that $(\mathcal{A}, \beta\frac{a}{x}) \models \forall y \varphi$. That is, for all $b \in A$, we have $(\mathcal{A}, (\beta\frac{a}{x})\frac{b}{y}) \models \varphi$. This is equivalent to: $(\mathcal{A}, (\beta\frac{b}{y})\frac{a}{x}) \models \varphi$ for all b. This is equivalent to: $(\mathcal{A}, (\beta\frac{b}{y})) \models \exists x \varphi$ for all b. This is equivalent to $(\mathcal{A}, \beta) \models \forall y \exists x \varphi$.

b. We give a counterexample, that is, an $\{R\}$-structure (A, R^A) where $(A, R^A) \models \forall y \exists x Rxy$ but not $(A, R^A) \models \exists x \forall y Rxy$. There are many such counterexample structures. One is to take $A = \mathbb{N}$ and choose R^A to be the >-relation on $\mathbb{N}$. Then clearly $(\mathbb{N}, >^{\mathbb{N}}) \models \forall y \exists x\, x>y$ but not $(\mathbb{N}, >^{\mathbb{N}}) \models \exists x \forall y\, x>y$.

Exercise 7 (20 points, easy to medium). Consider the empty symbol set $S = \emptyset$. An S-structure $\mathcal{A}$ is then just a set $\mathcal{A} = (A)$, and any set qualifies as an $\emptyset$-structure.

a. (15 points) Find a (possibly infinite) set $\Phi^{=i}$ of $\emptyset$-expressions such that $(A) \models \Phi$ iff A has i elements, where i is a positive natural number.

b. (5 points) Find a (possibly infinite) set $\Phi^{=\infty}$ of $\varnothing$-expressions such that $(A) \vDash \Phi$ iff A is infinite.

Solution. a. Let $\varphi^{=i}$ be the proposition

$$
\begin{aligned}
\exists x_1 \exists x_2 \ldots \exists x_i ((\quad & \neg x_1{=}x_2 \wedge \neg x_1{=}x_3 \wedge \ldots \wedge \neg x_1{=}x_i \wedge \\
& \wedge \neg x_2{=}x_3 \wedge \neg x_2{=}x_4 \wedge \ldots \wedge \neg x_2{=}x_i \wedge \\
& \ldots \\
& \wedge \neg x_{i-1}{=}x_i\ ) \\
\wedge \quad & \forall x_{i+1}(x_1{=}x_{i+1} \vee x_2{=}x_{i+1} \vee \ldots \vee x_i{=}x_{i+1})\ )
\end{aligned}
$$

The first long conjunction states that pairwise different $x_1, x_2,\ldots, x_i$ exist, the second states that any x_{i+1} must be one of the $x_1, x_2,\ldots, x_i$, so there cannot exist more than i elements in A. Put $\Phi = \{ \varphi^{=i} \}$.

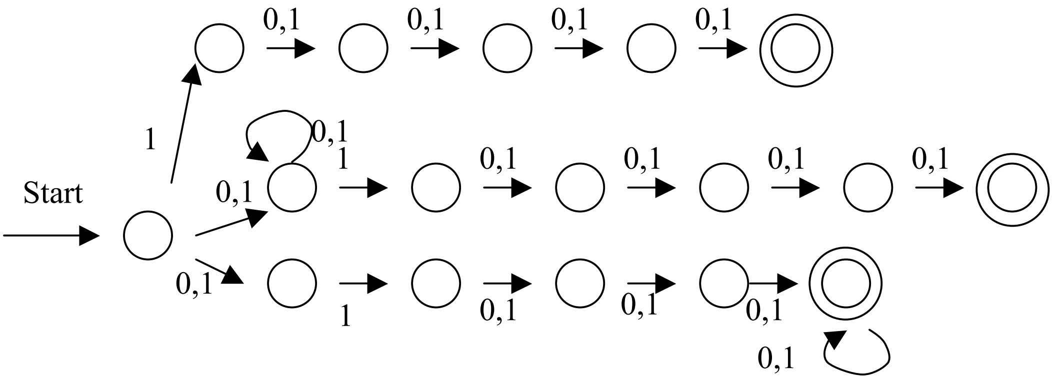
b. Put $\Phi^{=\infty} = \bigcup_{i=1}^{\infty} \{\neg\varphi^{=i}\}$

Solutions to the A group assignments for the ACS1 midterm, Fall 2003

Note: only solutions for group A are provided; the assignments for group B are minor variants.

1. Design a NFA that accepts all words from {0,1}* that have length at least 5 and whose fifth symbol from the right end is a 1 or whose second symbol is a 1 (non-exclusive "or"). Present your NFA by a transition diagram.

Solution. One solution is to provide three branches. The first branch is for all words of the form $1(0+1)^4$, the second for words of the form $(0+1)^+1(0+1)^4$, the last for all words of the form $(0+1)1(0+1)^3(0+1)^*$:



2. Show that the language $L = \{w1w \in \{0,1\}^* \mid |w| > 1000\}$ is not regular.

Solution: By pumping lemma, for a regular language *L,* there exists some n such that for all $w \in L$, where $|w| \geq n$, we can put $w = xyz$, where $|xy| \leq n$ and $|y| > 0$, and all xy^kz, $k \geq 0$, are also in L. Put $m = \min(1000, n)$. Consider $w = 0^m10^m$. Then $w \in L$. Assume L is regular. By pumping lemma, we can find a nonempty substring y of 0's in the first zero string of w, such that deleting this substring yields another word $0^{m-|y|}10^m$ in L. Contradiction.

3. Let A be a DFA with states $q_0, q_1, ..., q_n$. A has a single accepting state q_k. Let a be a symbol from its input alphabet. Assume that the a-transitions form a cycle over the states, that is, $\delta(q_i, a) = q_{i+1}$ for $i = 0, ..., n-1$, and $\delta(q_n, a) = q_0$. Let $L(A)$ be the language accepted by A.

   a. Give a formal description of $L_a = L(A) \cap \{a\}^*$.
   b. Prove that the language you described in a. is actually the language accepted by A.

Solution. a. $L_a = \{a^m \mid m = k + in, i \geq 0\}$.
b. (i) We have to show that every a^{k+in} is accepted by the DFA, for every i. This is true for $i = 0$ because obviously $\hat{\delta}(q_0, a^k) = q_k$. Induction proof: Assume $\hat{\delta}(q_0, a^{k+in}) = q_k$. Then also $\hat{\delta}(q_0, a^{k+(i+1)n}) = q_k$ because $\hat{\delta}(q_k, a^n) = q_k$. (ii) Conversely, let $w \in L(A) \cap \{a\}^*$. Then w is

a string of all a's and is accepted by A. By induction on length of words a^m, show that $\hat{\delta}(q_0, a^m) = q_{\mathrm{mod}(m,n)}$. (For a perfect solution, this induction would have to be done explicitly). That is, $\hat{\delta}(q_0, a^m) = q_k$ iff $m \equiv k \pmod{n}$, that is, $m = k + in$.

4. Design a PDA for the language L of words that contain at most as many 0's as 1's (including ε). Specify your PDA by its transition function, and describe the principles behind your design in intuitive terms. You may choose acceptance by empty stack or accepting states, whatever you find more convenient.

Sketch of a solution: Accept by final state. Use two stack symbols + and –. Idea: at any time, there are either only +'s or –'s in the stack (plus the start stack symbol Z_0). +'s indicate excess of 1's over 0's in the input read so far, –'s indicate excess of 0's. Whenever Z_0 or + is on top of stack, the PDA may go to the accepting state. When a 1 is read and a + or Z_0 is on top of stack, push a further +. When a 1 is read and a – is on top, pop the –. When a 0 is read and a – or Z_0 is on top of stack, push a further –. When a 0 is read and a + is on top, pop the +.

5. Consider the CFG $S \to aS \mid aSbS \mid \varepsilon$.

a. Give two different parse trees for *aaba*.
b. Prove that this grammar produces all words w over $\{a, b\}$ such that every prefix of w has at least as many a's as b's.

Solution to b. (i) Let w be generated by the grammar. Induction over number # of derivations used to derive w. Basis: # = 1 entails $w = \varepsilon$, prefix property clear. # = 2: entails $w = a$, prefix property also clear. Induction: Case 1: Let the first step in a derivation of w be $S \to aS$, that is, $w = av$, and v can be derived in fewer steps than w. By induction, v has prefix property. Then w has prefix property because a prefix ax of av has one more a than x, which is a prefix of v, which already has at least as many a's as b's. Case 2: Let the first step in a derivation of w be $S \to aSbS$, that is, $w = avbu$, and v and u have prefix property. Then every prefix of av has at least one more a's as b's, as in case 1; this implies that every prefix of avb has at least as many a's as b's; which finally implies that every prefix of $avbu$ has as many a's as b's because u has this property.

(ii) Conversely, let w be a word with the prefix property. Induction over length l of w. Basis $l = 0$: then $w = \varepsilon$, can be derived in grammar. $l = 1$: then $w = a$, can also be derived. Induction: Let w have length greater 1. It must start with an a. Case 1: w consists only of a's. Then it can obviously be derived by subgrammar $S \to aS \mid \varepsilon$. Case 2: w has at least one b. Then $w = avbu$, where $u \in \{a\}^*$, and v has prefix property (because if it hadn't, avb would have more b's than a's.) Thus, v and u can be derived in grammar; which entails that $avbu$ can be derived, too, by starting with $S \to aSbS$ and inserting the derivations for v and u for the two S's.

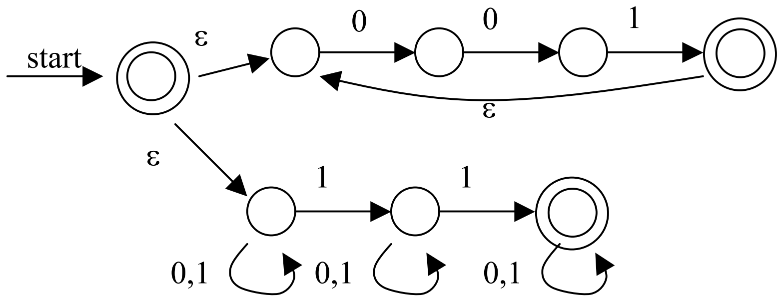
Advanced Computer Science 1
Final Exam, December 16, 2003

Group A

Note: a maximum of 100 points is accredited for this exam.
I wish you a happy landing!!

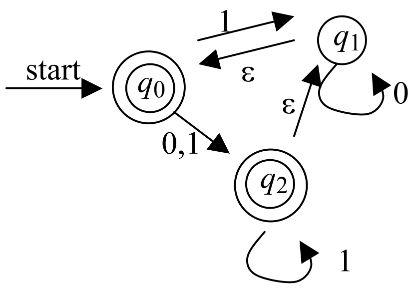
Problem 1 (15 points in total) **a.** (10 points) Give an ε-NFA for the language $L = \{w \in \{0,1\}^* \mid w \text{ is a concatenation of } i \text{ copies of } 001, \text{ where } i \geq 0\} \cup \{w \in \{0,1\}^* \mid w \text{ contains at least two } 1\text{'s}\}$.
b. (5 points) Give a regular expression for this language. You may use bracket-saving conventions.

Solution: a.



b. 001* + (0+1)* 1 (0+1)* 1 (0+1)*

Problem 2 (15 points) **a.** (01 points) Convert the ε-NFA shown below into a DFA, using the subset construction. Specify the state set of the DFA obtained, the starting state, the accepting states, and give a transition table. **b.** (5 points) Give the equivalent minimal DFA.



Solution. Step 1: convert to DFA $A = (Q', \{0,1\}, \delta', q_0', F')$. We perform a direct subset construction and get

$Q' = \text{Pot}(\{q_0, q_1, q_2\})$
$q_0' = \text{ECLOSE}(q_0) = \{q_0\}$
$F' = \{\{q_0\}, \{q_1\}, \{q_0, q_1\}, \{q_0, q_2\}, \{q_1, q_2\}, \{q_0, q_1, q_2\}\}$
the transition table for δ' is quite simple, because we find that from the starting point we reach state $\{q_0, q_1, q_2\}$ with 0 and 1:

	0	1
$\{q_0\}$	$\{q_0, q_1, q_2\}$	$\{q_0, q_1, q_2\}$
$\{q_0, q_1, q_2\}$	$\{q_0, q_1, q_2\}$	$\{q_0, q_1, q_2\}$

Step 2 would be to convert this into the minimal DFA. But we see that because all states in A are accepting states, the language L is actually $\{0,1\}^*$, and thus we can directly say that the minimal DFA is the single-state DFA with state q (which is both the starting and the accepting state) and transitions $\delta(q,0) = \delta(q,1) = q$.

Problem 3 (10 points). Prove that a regular language L over Σ can be specified by a regular expression that does not contain * if and only if L is finite.

Solution: "$\Rightarrow$": Induction on the structure of regular expressions E without *. Basis: $E = \varepsilon$ or $E = \emptyset$ or $E = \boldsymbol{a}$, where $a \in \Sigma$: then $L(E) = \{\varepsilon\}$ or $L(E) = \emptyset$ or $L(E) = \{a\}$, all of which are finite. Induction: let E, F be regex's without * and their corresponding languages $L(E)$ and $L(F)$ be finite. Then $L((E + F)) = L(E) \cup L(F)$ and $L((E\ F)) = L(E)\ L(F)$, both of which are finite.
" $\Leftarrow$": Let L be a finite language with words $w^i = a^i{}_1...a^i{}_{Ni}$, where $i = 1, ..., N$. Put $F^i = \boldsymbol{a}^i{}_1...\boldsymbol{a}^i{}_{Ni}$ and $E = F^1... F^N$. Then E is a regular expression for L without *.

Problem 4 (20 points) Show that $L = \{a^m b^m c^k \mid k = 2m+1\}$ is not context-free.

Solution: Assume L is context-free. Let n be the PL constant. Consider $w = a^n b^n c^{2n+1} \in L$. By PL, $w = xuyvz$, such that $|uyv| \leq n$, $|uv| \geq 1$, for all $i \geq 0$, $xu^iyv^iz \in L$. Case distinction:

1. uyv falls within $a^n b^n$. Then putting $i = 0$ changes $a^n b^n$ into something different from $a^n b^n$, say r, and the resulting $xu^iyv^iz = rc^{2n+1} \notin L$.
2. uyv touches both b^n and c^{2n+1} (therefore u touches b^n and v touches c^{2n+1}), and $u \notin \varepsilon$. Then with $i = 0$, $xu^0yv^0z = a^n b^{n'} c^{k'}$, with $n < n'$, and $a^n b^{n'} c^{k'} \notin L$.
3. uyv touches both b^n and c^{2n+1}, and $u = \varepsilon$. Then $v \notin \varepsilon$, and with $i = 0$, $xu^0yv^0z = a^n b^{n'} c^{k'}$, with $k' \neq n$, therefore $a^n b^{n'} c^{k'} \notin L$.
4. uyv falls within c^{2n+1}. Then $xu^0yv^0z = a^n b^n c^k$, with $k \neq 2n+1$, therefore $a^n b^n c^k \notin L$.

Thus, however we may shift uyv within w, by pumping with $i = 0$ we get a word not in L, therefore our assumption that L is context-free must be wrong.

Problem 5 (15 points). For the alphabet of terminals $T = \{x, {}_1, {}_0, f, c, d\}$ give a context-free grammar $G = (V, T, P, S_0)$ in which you can derive exactly those words of terminals that correspond to FOL terms over the FOL symbol set $S = \{c, d, f\}$, where c and d are constant symbols and f is a binary function symbol. Assume that the indices i of the FOL variable set are written in binary, that is, the FOL variables are the strings $x_0, x_1, x_{10}, x_{11}, ...$.

Solution: Put $V = \{t, \text{var}, \text{index}\}$, $S_0 = t$, and let P be made from the following productions:

$t \rightarrow \text{var} \mid c \mid d \mid ftt$
$\text{var} \rightarrow x_0 \mid x_1\ \text{index}$
$\text{index} \rightarrow {}_0\ \text{index} \mid {}_1\ \text{index} \mid \varepsilon$

Problem 6 (20 points). Design a type-0 grammar (that is, an unrestricted grammar where in productions you may replace arbitrary substrings by arbitrary substrings, see Definition 9.1 in script) for $L = \{a^m b^m c^m \mid m > 0\}$. Explain the idea behind your grammar in words and list your productions.

Solution. There are many ways to do this. One possibility is to generate in a first stage a string 11...1*abc* of *m* 1's followed by *abc*, with *m*-1 1's (which are variables) and then in a second stage use up the 1's from right to left, where for each 1 used the final string of $a^i b^i c^i$ is turned into $a^{i+1} b^{i+1} c^{i+1}$. At the beginning of stage 2, $i = 1$. The information that some 1 is currently processed is transported across the *a*, *b*, and *c*'s with special marker variables. Concretely, this might look as follows:

Start symbol: S
Variables: $\{S, T, 1\}$

Stage 1: generate 11...1*abc*, with zero or more 1's. Productions for this stage:

$S \to Tabc$
$T \to 1T \mid \varepsilon$

Stage 2: any rules of stage 2 are only applicable after T has been replaced by ε, that is, after termination of stage 1. Then, the leading 1's are "sweeped" right across the $a^i b^i c^i$, such that each sweeped 1 leaves behind an extra *a*, an extra *b*, and an extra *c*. The productions are desgned such that the 1's, while being sweeped, cannot interfere with each other. That is, if some 1 is sweeped while an earlier 1 is still "in the line", and the two meet in a 11 subsequence, there is no production to continue this.

$1aa \to a1a$	// start moving the rightmost 1 right, in case there are already at least 2 *a*'s
$1ab \to a1b$	// start moving the rightmost 1 right, in case there was only one *a* so far
$a1a \to aa1$	// continue moving right if *a*'s are still there
$a1b \to aab1$	// add one *a* and start moving the 1 across the *b*'s
$b1b \to bb1$	// continue moving right if *b*'s are still there
$b1c \to bbcc$	// if the *c*'s are met, add one *b* and one *c*

Problem 7 (15 points) Recall from elementary algebra that a group is a set A with a special neutral element e and a binary operation $\circ$ such that the following axioms hold:

1. $\circ$ is associative, that is, for all elements x, y, z of A it holds that $(x \circ y) \circ z = x \circ (y \circ z)$,
2. the neutral element satisfies $x \circ e = x$ for all elements x of A,
3. every element has a right inverse, that is, for all x there exists an y such that $x \circ y = e$.

Using the FOL symbol set $S = \{e, \circ\}$, formulate these three axioms as FOL propositions in the correct FOL syntax (that is, use only variables of the form x_i, use prefix notation for the function symbol $\circ$, don't use bracket saving conventions).

Solution.

$\forall x_1 \forall x_2 \forall x_3 \circ \circ x_1\ x_2\ x_3 =\circ\ x_1 \circ x_2\ x_3$
$\forall x_1 \circ x_1\ e = x_1$
$\forall x_1 \exists x_2 \circ x_1\ x_2 = e$

1. Consider the CFG $S \to aS \mid aSbS \mid \varepsilon$.

c. (5 points) Give two different parse trees for *aaba*.
d. (30 points) Prove that this grammar produces all words w over $\{a, b\}$ such that every prefix of w has at least as many a's as b's.