# 02140 Languages and Parsing:
# Mandatory Exercise Set 3

Michael R. Hansen

November 15, 2004

## Report Requirements

The report for this mandatory assignment must be handed in no later than **November 29, at 13.00** (hard deadline). Your report must be put into the box named 02140, which is placed in just inside the west entrance of Building 322.

The problems of this assignment are based on the examination questions in 02140 from December 11th, 2002 (Problems 2 - 5) and December 11th, 2003 (Problem 1 and 6). Notice, when considering the percentages of the individual problems, that the duration of last year's examination was 2 hours, while the duruation of the examination in 2002 was 4 hours. The duration of the written examination this year is 2 hours.

The report should comprise the following:

- Your names with signatures and student numbers. Group size should be 2 to 3.
- Solution to the problems 1 – 6.
- SML implementations, with tests, for Problem 5.2 and 5.3, and Problem 6.
- Furthermore, extend the grammar and abstract syntax in Problem 5 in order to extract the total income for a company, by adding up the income of all departments, sub-departments, sub-sub-departments, and so on. The relevant propram pieces and tests for this extra problem must also be included in the report.

## Problem 1 (30%)

Let $L$ be the language over alphabet $\Sigma = \{a, b, c\}$ defined by

$$L \stackrel{\text{def}}{=} \{a^k b^l c^n \mid k + n = l \geq 0\} \ .$$

1. Prove that $L$ is not regular.
2. Is $L$ context-free? Justify your answer.

## Problem 2 (15%)

1. Give a construction which, given arbitrary DFA $A = (Q, \Sigma, \delta, q_0, F)$, yields a DFA accepting the language
$$L^A_{\text{even}} \stackrel{\text{def}}{=} \{w \in L(A) \mid |w| \text{ is even}\} \ .$$
2. Provide a construction which, given arbitrary DFA $A = (Q, \Sigma, \delta, q_0, F)$ and $n > 0$, yields a DFA accepting the language
$$L^A_{\mathrm{nondiv}} \stackrel{\text{def}}{=} \{w \in L(A) \mid |w| \text{ is not divisible by } n\} \ .$$

## Problem 3 (20%)

Show the following claims, where all the languages are over alphabet $\Sigma = \{a, b, c\}$:

1. the set $L_1$ of all words $w \in \Sigma^*$ with the property "$w$ ends with $abba$" is regular;
2. the set $L_2$ of all words $w \in \Sigma^*$ with the property "if each occurrence of the symbol $c$ in $w$ is replaced by the string $aba$ then the resulting word is a palindrome" is context-free;
3. the set $L_3 = L_1 \cap L_2$ of all words which have the above properties is context-free.

## Problem 4 (15%)

Answer "true" or "false" to the following claims, and give a brief explanation to each answer. All strings considered are over some alphabet $\Sigma$ containing at least two symbols.

1. The language $\{w \in \Sigma^* \mid |w| \text{ is non-prime}\}$ is regular.
2. If $L$ is accepted by an NFA with $n$ states then $\mathrm{pref}(L)$ is accepted by some DFA with $2^n$ states, where $\mathrm{pref}(L) \stackrel{\text{def}}{=} \{x \mid xy \in L \text{ for some } y \in \Sigma^*\}$.
3. The intersection of a context-free language with the complement of a context-free language is regular.
4. If $L$ is the language of a 1024-state $\varepsilon$-NFA then $L$ is the language of a 1-state PDA.
5. If $L$ is a superset of a non-regular language then $L$ is non-regular.

## Problem 5 (25%)

We now consider a parsing problem concerning representations of companies.

A *company* consists of a list of departments, where a *department* is characterized by a name, which is a string, an income, which is an integer, and a (possibly empty) list of sub-departments.
The corresponding datatype declaration for the abstract syntax of companies is:

```
datatype company    = Comp of department list
and      department = Dept of string * int * department list
```

and as input specification to mosmlyac we use:

```
%{ open Absyn %}

%token <int> INCOME
%token <string> NAME
%token EOF

%start COMP
%type <Absyn.company> COMP
%type <Absyn.department> DEPT
%type <Absyn.department list> DEPTList
%%

COMP: DEPTList EOF              { Comp $1 }        ;

DEPT: NAME INCOME DEPTList      { Dept($1,$2,$3) } ;

DEPTList:
                                { []       }
  | DEPT DEPTList               { $1 :: $2 }
;
```

Unfortunately, the grammar is ambigious and shift/reduce conflicts are reported.

1. Find the ambiguity in the grammar.
2. Change/extend the specification so that the ambiguity is resolved. Give a brief explanation of your decisions. (You have to change the language defined by the grammar.)
3. Extend the specification and the abstract syntax so that a non-empty list of employees is associated with every department. An employee is described by a name and an income. Take care that your specification is free of conflict.
4. We want to enforce that an employee is associated with at most one department in a company. Give a brief explanation of how you would handle this.

## Problem 6 (20%)

A regular substitution is a substitution where the languages substituting the individual symbols are regular. The regular languages are closed under regular substitution. We here study this property in the context of regular expressions.

Consider the following representation of regular expressions in SML, which was used in the course:

```
infix 4 ++
infix 6 ##

datatype reg = Empty                (* Empty set              *)
             | Epsilon              (* Empty string           *)
             | S of string          (* Symbol in the alphabet *)
             | ++ of reg * reg      (* Union                  *)
             | ## of reg * reg      (* Concatenation          *)
             | ** of reg;           (* Kleene star            *)
```

A substitution maps symbols of the alphabet to regular expressions, i.e. a substitution is a function with the type: `string -> reg`. An example of a substitution is:

```
fun s1 "a" = ** (S "d" ++ S "e")
  | s1 "b" = Epsilon
  | s1 x   = Empty
```

which maps $a$ to the regular expression $(d+e)^*$ (represented by `**(S "d" ++ S "e")`), $b$ to $\epsilon$ (represented by `Epsilon`), and all other symbols of the alphabet are mapped to $\emptyset$ (represented by `Empty`).

1. Declare an SML function:

   ```
   subst : reg -> (string -> reg) -> reg
   ```

   where `subst` $r\ s$ is the regular expression obtained from the regular expression $r$ by substituting every occurrence of `S` $a$ in $r$ with $s(a)$. For example, let `r1` be the regular expression:

   ```
   ** (S "a" ++ S "b") ## (S "c" ++ Epsilon)
   ```

   Then `subst r1 s1` is the regular expression

   ```
   ** ( ** (S "d" ++ S "e") ++ Epsilon ) ## (Empty ++ Epsilon)
   ```