Written Examination, December 11th, 2003 Course no. 02140

The duration of the examination is 2 hours.

Course Name: Languages and parsing

Allowed aids: All written material

The problem set consists of 5 problems which are weighted as follows:
Problem 1: 15%, problem 2: 30%, problem 3: 15%, problem 4: 20%, problem 5: 20%

Marking: 13-scale.

Problem 1 (15%)

Provide regular expressions *or* DFAs for the following languages:

1. $\{w \in \{a, b\}^* \mid \#_a(w) \text{ is even}\}$,
   where $\#_a(w)$ denotes the number of occurrences of a in w;

2. $\{w \in \{a, b\}^* \mid \#_a(w) \geq 1 \wedge \#_b(w) \geq 2\}$,
   where $\#_a(w)$ is as before and $\#_b(w)$ is the number of occurrences of b in w.

Problem 2 (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 3 (15%)

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

1. If L is accepted by a DFA with 2^n states, where $n \geq 1$, then L is accepted by some NFA with n states.

2. If $G = (V, T, P, S)$ is a grammar with exactly one production (i.e., $|P| = 1$) then $L(G)$ is regular.

3. Each superset of a co-finite language is regular. (A co-finite language contains all but finitely many strings from Σ^*.)

 ... Continued on next page

Problem 4 (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 has 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 ϵ (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)
   ```

 ... Continued on next page

Problem 5 (20%)

We now consider a parsing problem for *piecewise linear curves* in the plane.

A *point* in the plane is given by a pair of integers (x, y) and a *line segment* is given by the *start point* and the *end point* of the segment. For example, $\mathrm{Line}((1,2),(7,-3))$ describes the line segment, where $(1,2)$ is the start point and $(7,-3)$ is the end point. We shall consider the following operations on curves:

- $\mathrm{Join}(c_1, c_2)$ is the piecewise linear curve obtained by connecting the end point of the curve c_1 with the start point of the curve c_2.
- $\mathrm{MirrorX}\, c$ is the mirror of the curve c with respect to the X-axis.
- $\mathrm{Scale}\, n\, c$ scales the curve c with the factor n, which is a positive integer.

The corresponding datatype declarations for the abstract syntax of points and curves are:

```
datatype point = P of int * int
and      curve = Line of point * point
               | Join of curve * curve
               | MirrorX of curve
               | Scale of int * curve
```

In the concrete syntax Line and Join are infix operators, and as input specification to mosmlyac we use:

```
%{ open Absyn %}

%token <int> INT
%token LPAR RPAR COMMA LINE JOIN MIRRORX SCALE
%token EOF

%start START
%type <Absyn.curve> CURVE START
%type <Absyn.point> POINT
%%

POINT: LPAR INT COMMA INT RPAR     { P($2, $4) } ;

START:  CURVE EOF                  { $1 } ;

CURVE:
     LPAR CURVE RPAR               { $2          }
   | POINT LINE POINT              { Line($1,$3) }
   | CURVE JOIN CURVE              { Join($1,$3) }
   | MIRRORX CURVE                 { MirrorX $2  }
   | SCALE INT CURVE               { SCALE($2,$3)};
```

1. Find the ambiguities in this grammar and show how the problems with them can be solved.