Informatics 2A 2018–19
Tutorial Sheet 4 (Week 6)

Mary Cryan

This week's exercises concern material from Lectures 13 and 14 on formal languages.

1. Recall the grammar for mathematical regular expressions from last week's sheet:

$$\begin{array}{rcl} \mathsf{RegExp} & \to & \mathsf{Atom} \mid \mathsf{RegExp} + \mathsf{RegExp} \mid \mathsf{RegExp}\ \mathsf{RegExp} \\ & & \mid \mathsf{RegExp}* \mid (\mathsf{RegExp}) \\ \mathsf{Atom} & \to & \mathsf{sym} \mid \boldsymbol{\emptyset} \mid \boldsymbol{\epsilon} \end{array}$$

   Using methods from Lecture 13, construct an LL(1) grammar for the language of regular expressions that is equivalent to the grammar given earlier. Your grammar should embody the usual precedence conventions for regular expressions: $*$ takes precedence over concatenation, which takes precedence over $+$. (If you are in doubt as to whether your grammar is LL(1), see whether you can construct a parse table for it.)

   Hint: Use some of the ideas we exploited in building an unambiguous (and also LL(1)) grammar for arithmetic expressions (Lecture 13).

2. The *concrete syntax* for Micro-Haskell (MH) types (see Lecture 14 and Assignment 1) is specified by the LL(1)-grammar:

$$\begin{array}{rcl} \mathsf{Type} & \to & \mathsf{Type0}\ \mathsf{TypeRest} \\ \mathsf{TypeRest} & \to & \epsilon \mid \texttt{->}\ \mathsf{Type} \\ \mathsf{Type0} & \to & \texttt{Integer} \mid \texttt{Bool} \mid (\ \mathsf{Type}\ ) \end{array}$$

   with start symbol Type. The *abstract syntax* is specified by the grammar

$$\mathsf{Type} \to \texttt{Integer} \mid \texttt{Bool} \mid \mathsf{Type}\ \texttt{->}\ \mathsf{Type}$$

   (a) Write out the concrete parse tree for the type expression:

```
Integer -> Bool -> Integer
```

   (b) Write out the abstract syntax tree that the above parse tree will be converted to.

   Consider the following grammar for an abstract syntax of arithmetic expressions (involving only the binary operations $-$ and $+$).

$$\mathsf{Exp} \to \mathsf{n} \mid \mathsf{Exp} + \mathsf{Exp} \mid \mathsf{Exp} - \mathsf{Exp}$$

   where (as in the last tutorial) n stands for some lexical class of *numeric literals*.

(c) Give an LL(1)-grammar for a corresponding concrete syntax of bracketed arithmetic expressions.

(d) Write out the concrete parse tree for the arithmetic expression:

$$10 - 5 - 4$$

(e) Write out the abstract syntax tree that the above parse tree should be converted to.

(f) What difference arises in the process of translating concrete parse trees to abstract syntax trees between the case of MH types and that of arithmetic expressions? What is the origin of this difference?

3. Consider any context-free grammar $\mathcal{G} = (N, \Sigma, P, S)$ (where $S \in N$ and where P is some finite set of production rules $X \to \alpha$ for $X \in N$ and $\alpha \in (\Sigma \cup N)^*$).

   We write $X \Rightarrow^* \beta$ if the right-hand side $\beta \in (\Sigma \cup N)^*$ can be generated from the non-terminal $X \in N$ over any number (including 0) of applications of the production rules. We write $X \Rightarrow^+ \beta$ if β can be generated by a strictly positive number of applications of the production rules (not 0).

   The grammar $\mathcal{G}$ is said to be *cycle-free* if there is no non-terminal X such that $X \Rightarrow^+ X$.

   Give an algorithm to convert a given context-free grammar into an equivalent grammar which is cycle-free.