**UPPSALA UNIVERSITET**
Matematiska institutionen
Erik Palmgren

LABORATION 1
Tillämpad logik DV1, ht-01
2001-09-11

**Laboration 1**

The following exercises concern mainly Chapter 1 – 4 of "Konstruktiv logik". The purpose of this laboration is to make you familiar with the proof editor Alfa. The proof editor is based on a logical framework closely related to Martin-Löf type theory. The goal is not that you should understand Martin-Löf type theory already at this first laboration, but rather to be able relate to your knowledge of functional programming, e.g. ML or Haskell. The logical framework of Alfa (called Agda) is essentially a typed lambda-calculus which has a lot in common with ordinary functional programming languages. Among the exercises are also some problems on the BHK-interpretation.

The exercises may be solved in groups of up to three persons. The solved Alfa-files (`.alfa`) should be sent by e-mail to `palmgren@math.uu.se` at the *latest October 17, 2001.* Also hand in an annotated and commented version on paper (or insert the comments directly in the Alfa-file, if you know how to do this without breaking the file). The files necessary for the laboration can be downloaded from the home page of the course. Put them in an empty directory and make this your working directory. Start Alfa using the command `alfa`.

1. Open the file `lab1_1.alfa`. In this file are some simple functions that you might recognize from a course in functional programming. Declare and define constants that represent the following functions by using the `case`-construction of Alfa. Do not forget to test your definitions. Use the termination check button to see that your function does not call itself without decreasing some argument. The compute/evaluate button makes it possible to run the examples.

    (a) The identity function *id* on natural numbers.

    (b) The function *max* which returns the greatest of two natural numbers.

    (c) The function *fact* which computes the factorial of a natural number. You will probably need to define some auxiliary function.

2. In the previous exercise the `case`-construction was used to build terms. In the compendium "Konstruktiv logik" a different method is described. By this method a primitive recursion principle is first given for the data type which is then used to define other recursive functions. For example, the recursion principle for $\mathbb{N}$ is formalized using the recursion operator `rec`.

    (a) Open the file `lab1_2.alfa`, declare and define constants representing the function listed in Exercise 1 by using the recursion operator `rec`. (Notice that in Alfa `rec` has an extra argument $A$ indicating the type. This is sometimes filled in automatically, but you may have to do it by hand.)

Remark 1: Observe that the recursion operator rec is defined via Alfa's case-construction. This is typical of strict formalizations in Alfa. I.e. the case-construction is used to *define* a theory. Derivations *within* this theory use recursion operators.

As you will see, it is neither easy nor lucid to use recursion operators. The case-operation is far easier to read. In this course it is recommended to use the case-construction wherever this is suitable.

Remark 2: In Laboration 2 we will see how to to specify functions like $max$ by refining its type. When constructing an element of this type we construct not only the program, but simultaneously also its correctness proof. Consequently we do not need to test the program.

3. Open the file `lab1_3.alfa`. Prove the following propositions by filling in the question marks in Alfa's main window.
   (a) $A \to A$
   (b) $A \wedge B \to B \wedge A$
   (c) $\neg A \to (A \to B)$
   (d) $A \to (\neg A \to B)$
   (e) $A \to \neg\neg A$
   (f) $\neg\neg\neg A \to \neg A$
4. Binary trees might be viewed as a sort of generalisation of natural numbers. A natural number is either 0 or $S(n)$ where $n$ is a natural number. Similarly a binary tree is a leaf $L$ or a successor (branching) $B(b_0, b_1)$ where $b_0, b_1$ are binary trees. (By letting $0 = L$ and $S(n) = B(n, L)$ we can easily represent the natural numbers.)
   (a) Open the file `lab1_4.alfa`. Define a type $Bin$ of binary trees complete with introduction, elimination and equality rules.
   (b) The depth of a tree is defined by

$$\begin{aligned} depth(L) &= 0 \\ depth(B(b_0, b_1)) &= 1 + max(depth(b_0), depth(b_1)). \end{aligned}$$

   Construct an element $depth$ of the type $Bin \to \mathbb{N}$ that computes the depth of a tree. Use the recursion operator.

   (c) Test the function $depth$ on some input.

This laboratory exercise was developed by Lars Lindqvist.