Technische Universität München *Summer 2011*
Fakultät für Informatik *Kalmer Apinis*
Prof. Dr. H. Seidl *apinis@in.tum.de*

Virtual Machines

Exercise Sheet 8

Deadline: July 12th, 2011, 14:00

Exercise 1: *Algebraic Data types* *20 Points*

Previously we have extended the language with tuples, lists and trees. Now we generalize these previous attempts to support all simple algebraic data types.

To use an algebraic (or variant) types one must first write a type declaration (similar to OCaml):

```
type <type variables> <name> =
    <Constructor$_1^1$> of <type$_1^1$> * <type$_2^1$> * ... * <type$_n^1$>
  | ...
  | <Constructor$_1^k$> of <type$_1^k$> * <type$_2^k$> * ... * <type$_m^k$>
```

For example:

```
  type 'a list = Nil | Cons of 'a * ('a list)
  type ('a, 'b) pair = P of 'a * 'b
```

Then one can use these constructors to build values of the declared types. Syntax for this is similar to function application.

```
  let int_list = Cons (1, Cons (2, Cons (4, Nil))) in ...

  let something = f (P (1, 2)) in ...
```

And one can peek into algebraic values by means of pattern matching

```
  match int_list with
    | Cons (x, Cons (_, Cons (z, rest))) -> x+z
    | _                                  -> 42
```

The syntax for pattern matching is as follows:

```
  <pat> ::= _
          | <variable>
          | <constructor> (<pat$_1$>, ..., pat$_n$)
          | (<pat>)

  <exp> ::= ... |
      match <exp> with
        | <pat$_1$> -> <exp$_1$>
        | ...
        | <pat$_n$> -> <exp$_n$>
```

Define code generation for creating algebraic values and pattern matching on them. You probably need to define new instructions.