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

# Virtual Machines

*Exercise Sheet 4*

*Deadline: 7. June 2011, 14:00*

Exercise 1: *Structs and Arrays* *8 Points*

The so-called *Horner scheme* can be used for the quick evaluation of a given polynomial. For instance, one would evaluate the polynomial

$$f : x \mapsto -12x^4 + 8x^3 - 4x^2 + 3x + 1$$

as follows:

$$f_{Horner} : x \mapsto (((-12x + 8)x - 4)x + 3)x + 1$$

The following piece of code implements the Horner scheme while making use of a struct- and array-based representation of polynomials. Note that the use of *integers* instead of *floats* is a simplification due to the design of the CMa.

```
struct polynomial {
    int *coeffs;
    int degree;
};

int fCoeffs[] = { 1, 3, -4, 8, -12 };
struct polynomial f;
f.coeffs = fCoeffs;
f.degree = 4;

int x, acc, i;

x = 3;
acc = 0;
for (i = f.degree; i >= 0; --i)
    acc = acc * x + f.coeffs[i];
```

1. Give the mapping $\rho$ for the variables and field names in this code.
2. Write a VAM executable .cma file for the above code segment.

Exercise 2: *Von Neumann architecture* *5 Points*

In the lecture we usually separate the instruction array ($C$) and other memory ($S$ and $H$). This is called the fixed-program model — the program is fixed at run time and cannot change. Machines (like CMa) that are designed according to these models are called fixed-program machines. But most real hardware stores its programs in the same memory "array" as other data — these are stored-program machines.

For this exercise define a instruction encoding scheme, that encodes a list of CMa instructions (and their arguments) into an array of 32bit integers. The encoding scheme may not add any restrictions (other than that integers are at most 32bit) — all valid programs must be encodabel.

CMa instructions: ADD JUMPI ALLOC JUMPZ NEG AND LE NEQ CALL LEQ NEW DIV LOAD NOT DUP LOADA OR ENTER LOADC POP EQ LOADR RETURN GE LOADRC STORE GEQ MARK STOREA GR MOD STORER HALT MOVE SUB JUMP MUL XOR

Exercise 3: *Ackermann function* *2 Points*

Implement the Ackermann function (defined below) in the Pure Functions language. For natural numbers $m$ and $n$:

$$A(m,n) = \begin{cases} n+1, & \text{if } m = 0 \\ A(m-1,1), & \text{if } m > 0 \text{ and } n = 0 \\ A(m-1, A(m,n-1)) & \text{if } m > 0 \text{ and } n > 0. \end{cases}$$

Exercise 4: *Global Variables* *5 Points*

1. Give a formal definition of the function *glob* such that $glob(e)$ is the set of global variables in $e$.
2. Determine the set of global variables for each of the following expressions.
   - `(fun x -> x y) (fun y -> y)`
   - `fun x,y -> z (fun z -> z (fun x -> y))`
   - `(fun x,y -> x z (y z)) (fun x -> y (fun y -> y))`
   - 
```
((fun x -> x) z) + let a = x
                   in let x = f y
                      in let y = z
                         in x + y + z
```