Technische Universität München
Fakultät für Informatik
Prof. Dr. H. Seidl

Summer 2011
Kalmer Apinis
apinis@in.tum.de

Virtual Machines

Exercise Sheet 6

Deadline: June 28th, 2011, 14:00

Exercise 1: *Reverse Engineering* *8 Points*

Consider this pice of code and answer the questions below.

```
    alloc 1
    pushloc 0
    mkvec 1
    mkfunval _0
    jump _1
_0: targ 2
    pushloc 0
    getbasic
    pushloc 2
    getbasic
    gr
    jumpz _2
    mark _4
    pushloc 3
    getbasic
    pushloc 5
    getbasic
    sub
    mkbasic
    pushloc 5
    pushglob 0
    apply
_4: jump _3
_2: pushloc 0
    getbasic
    pushloc 2
    getbasic
    le
    jumpz _5
    mark _7
    pushloc 4
    getbasic
    pushloc 4
    getbasic
    sub
    mkbasic
    pushloc 4
    pushglob 0
    apply
_7: jump _6
_5: pushloc 0
_6:
_3: return 2
_1: rewrite 1
    mark _8
    loadc 6
    mkbasic
    loadc 4
    mkbasic
    pushloc 5
    apply
_8: slide 1
    halt
```

1. Is this code CBN or CBV?
2. Determine the stack distance sd for every program point (initially $sd = 0$).
3. What does this program compute ?

Exercise 2: *Extend MaMa with Registers* *12 Points*

We assume that we have an infinite number of *local* registers as well a single *global return* register. The former registers will be denoted as R_i and the latter one as GR. Again, local registers are *invariant* under function applications or the evaluation of closures.

Since we are not dealing with *references* like in, for instance, the expression **let** `x` = **ref** `0`, everything can be safely kept in registers at all times.

The basic code generation scheme for any one of $code_B, code_V$ and $code_C$ will be of the form

$$code_{(B|V|C)}\ e\ \rho\ sd\ i,$$

where e denotes the *expression* to be translated, ρ constitutes the current *address environment*, sd the *stack distance* and i the number of the local register that will hold the value of e after its evaluation.

During code generation, all local registers R_j with $j \geq i$ may be modified without any further constraints.

1. Develop the following code translation schemata:

$$\begin{aligned} code_B\, b\, \rho\, i \\ code_B\, x\, \rho\, i \\ code_B\, (\square_1\ e)\, \rho\, i \\ code_B\, (e_1\ \square_2\ e_2)\, \rho\, i \\ code_B\, (\textbf{if}\ e_0\ \textbf{then}\ e_1\ \textbf{else}\ e_2)\, \rho\, i \\ code_B\, e\, \rho\, i \end{aligned}$$

   Hint: You may have to give new implementations of `getvar`, `getbasic` and `jumpz`. Call them `rgetvar`, `rgetbasic` and `rjumpz`, respectively.

2. Then develop the following code translation schemata:

$$\begin{aligned} code_V\, b\, \rho\, i \\ code_V\, x\, \rho\, i \\ code_V\, (\square_1\ e)\, \rho\, i \\ code_V\, (e_1\ \square_2\ e_2)\, \rho\, i \\ code_V\, (\textbf{if}\ e_0\ \textbf{then}\ e_1\ \textbf{else}\ e_2)\, \rho\, i \end{aligned}$$

   Hint: You may have to give a new implementation of `mkbasic`, say `rmkbasic`.

3. Now that you have done the above, develop *CBN* translation schemata for both of

   $$code_V\, (\textbf{let}\ x_1\ =\ e_1\ \textbf{in}\ e_0)\, \rho\, i$$

   and

   $$code_V\, (\textbf{let rec}\ x_1 = e_1\ \textbf{and}\ \ldots\ \textbf{and}\ x_n = e_n\ \textbf{in}\ e_0)\, \rho\, i$$

   Don't bother with $code_C$ as this is a topic for future discussion.