Technische Universität München *Dr. K. N. Verma*
Fakultät für Informatik *verma@in.tum.de*
Prof. Dr. H. Seidl *Room: MI 02.07.041*

Virtual Machines

Summer Semester 2007

Exercise sheet 8 *Deadline: 19 June 2007 12:00*

Exercise 1: *20 Points*

References in functional languages correspond to variables (and pointers) in imperative languages. Consider the following example

```
letrec
   x = 1;
   f = fn y => if y <= 0 then x
               else let x = x * 2 in f (y - 1)
in f 8
```

`x = x * 2` creates a *new* `x` which is visible only inside the function `f`. The result is always 1.

Now consider the following example with references:

```
letrec
   x = ref 1;
   f = fn y => if y <= 0 then !x
               else let z = (x := !x * 2) in f (y - 1)
in f 8
```

`x = ref 1` assigns to the variable `x` a reference to the value 1. The assignment `x := !x * 2` will modify the value of the reference-variable `x`. No new variable will be created. The result is 256 ($= 2^8$).

For implementation, we introduce reference-objects as new heap objects. Reference objects consist of the tag R and a pointer (to a value).

Give code generation functions ($Code_V$) for the following expressions. Define new instructions as needed (e.g. `mkref` or `getref`).

a) `ref` e
creates a new reference object for the expression e and puts a pointer to it on the stack.

b) `!`e
gives the value of the reference defined by the expression e.

c) e_1 `:=` e_2
The reference defined by e_1 is assigned the value of the expression e_2, and this value is put on the stack.

d) Translate the second example above, with $\rho = \emptyset$ and $sd = 0$.