ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

D. Basin and P. Müller

Formal Methods and Functional Programming

Exercise Sheet 6: Lazy Evaluation and Repetition

Submission deadline: April 1st, 2012

Assignment 1:

Recall that the *lazy evaluation strategy* for an application `t1 t2` works as follows.

1. Evaluate `t1`
2. If beta-reduction is possible, then substitute `t1` without prior evaluation
3. No evaluation under abstraction

The *eager evaluation strategy* for an application `t1 t2` works as follows.

1. Evaluate `t1` and then evaluate `t2`
2. Perform beta-reduction, if possible
3. Evaluation carried out under an abstraction

Consider the two λ-terms $F = \lambda x.\, x\,(\lambda y.\, x\, y)$ and $G = \lambda x.\, y\, x$. Use β-reduction to evaluate FG using (i) the lazy and (ii) the eager evaluation strategy.

Assignment 2:

Consider the following functions.

```
foldr f z []     = z                    -- foldr.1
foldr f z (x:xs) = f x (foldr f z xs)   -- foldr.2

foldl f z []     = z                    -- foldl.1
foldl f z (x:xs) = foldl f (f z x) xs   -- foldl.2
```

Prove the following lemma.

Lemma: $\forall$ `f::(b -> a -> b) xs::[a] z::b.`
`foldl f z xs = foldr (\x r l -> r (f l x)) id xs z`

Hint: Take care of handling the quantifiers correctly. Use explicit $\forall$-elimination to reason about universal quantification.

Assignment 3: Symbolic Expressions

We represent multivariate polynomials over variables of type `a` with `Integer` coefficients using the following Haskell type.

```
type Poly a = [(Integer, [a])]
```

For example, the Haskell value `[(3, ["x","y"]), (1, ["y","y"])]` of type `Poly String` is a representation of the multivariate polynomial $3xy + y^2$.

(a) Write a Haskell function `evalPoly :: (a -> Integer) -> Poly a -> Integer` such that `evalPoly f p` computes the value of the polynomial `p` where every variable `x` is replaced by the value `f x`.

(b) We represent symbolic arithmetic expressions using the following Haskell type.

```
data AExpr a = Var a
             | Lit Integer
             | Add (AExpr a) (AExpr a)
             | Mul (AExpr a) (AExpr a)
```

Define the fold function

```
foldAExpr :: (a -> b) -> (Integer -> b) -> (b -> b -> b) -> (b -> b -> b)
          -> AExpr a -> b
```

for symbolic arithmetic expressions represented using `AExpr a`.

(c) Give the induction scheme for structural induction over `AExpr a` as an inference rule.

(d) Use the `foldAExpr` function from (b) to write a Haskell function

```
toPoly :: AExpr a -> Poly a
```

that converts an arithmetic expression to an equivalent multivariate polynomial. For example, the arithmetic expression $(3x+y)*y$ can be converted to the multivariate polynomial $3xy+y^2$.

Note that you do not have to prove the correctness of your solution.

Assignment 4:

(a) State the most general type (you do not have to formally prove your claims). Take also into account the type classes like `Eq`, `Ord`, `Num`, ...

(1) `\x y z -> head (map x y) y`

(2) `\x y z w -> z (x + 0) + w (y + 1)`

(3) `\x y z -> x y (y z) (fst z)`

(4) `\x y z -> (x y < z x) < (3 < 4)`

(b) Formally prove the following typing judgement:

$$\vdash (\lambda c.\, (\lambda x.\, \textbf{iszero}\ (c\ x)))\, (\lambda x.\, \textbf{snd}\ x) :: (a, Int) \to Bool$$