The Implementation of Functional Programming Languages, Notes

Julius Nguyen

4th May 2023

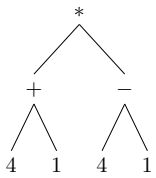
1 Preface

A functional program consists of expressions that are evaluated using a technique called **graph reduction**

$$fx = (x + 1) * (x - 1)$$

consists of **reducible expressions** that can be represented by the following graph:

Let $x = 4$ Then $f4 :=$



Which can be reduced to

Which in turn evaluates to

2 The Lambda Calculus

2.1 The syntax of Lambda Calculus

Lambda Calculus has no 'built-in' operators, but for convenience we can define operators which can be defined in terms of Lambda Calculus.

A function can take several arguments, e.g.

$$\lambda x.\lambda y.fxy$$

where f is a function.

We can rewrite functions with multiple arguments as functions that take single arguments.

$$\lambda x.(\lambda y.(fx)y)$$

The first version should be preferred, since it is the more readible one.

For convenience, we accept built-in functions such as:

```
IF TRUE E_t E_f → E_t
IF FALSE E_t E_f → E_f

HEAD (CONS a b) → a
TAIL (CONS a b) → b
```

CONS is short for CONSTRUCT.

The built-in functions like `CONS, HEAD` and `TAIL` can expressed by lambda expressions.

```
CONS = (λa.λb.λf.fab)
HEAD = (λc.c(λa.λb.a)
TAIL = (λc.c(λa.λb.b)
```

The syntax of a lambda expression in BNF looks like this:

```
<exp> :: = <constant>                    built-in constants
       | <variable>                      variable names
       | <exp> <exp>                     applications
       | λ <variable> . <exp>    lambda abstractions
```

Lower-case letters stand for variables, upper-case letters stand for lambda expressions.

2.2 The semantics of Lambda Calculus

A variable can be **bound** or **free**. A variable is bound, when it is taken as argument by λ ("λ x"). Otherwise the variable is free.

We have different operations to manipulate λ expressions.

- β-conversion: we reduce the term by replacing the variable by an argument that is provided as an input to an expression

$$(\lambda x. + x1)4$$

$$[x := 4]$$

$$+41$$

$$5$$

- β-abstraction: backwards operation of the β-reduction

$$+41 \leftarrow (\lambda x. + x1)4$$

  To show that we can convert back and forwards, we can write:

$$+41 \leftrightarrow [\beta](\lambda x. + x1)4$$

- α conversion: used to change names, when two or more expressions are equivalent

$$\lambda x. + x1 \leftrightarrow [\alpha](\lambda y. + y1)$$