P. Thiemann, G. Radanne Summersemester 2017/18

Essentials of Programming Languages

https://proglang.informatik.uni-freiburg.de/teaching/konzepte/2018ss/

Language 1 – Lambda calculus

2018-04-18

Lambda calculus

The lambda calculus needs no introduction! We will consider the lambda calculus with *Weak Head normal forms* with evaluation contexts and arithmetic operators.

$$\begin{array}{rcll} e & ::= & x & \text{Variables} \\ & | & (e\ e\ \ldots) & \text{Application} \\ & | & \lambda x.e & \text{Abstraction} \end{array}$$

Exercise 1 (Small step – Call by value)
Implement a small step call-by-value semantics for the lambda calculus that evaluates to Weak Head Normal Forms (WHNF) using evaluation contexts, as defined in the lecture. Write and test a few encoding such as booleans, church numerals, ... Try some big numerals.

Warning Be mindful about the definition of *substitutions*! Substitutions are often the source of bugs in the implementation of semantics. A file `subst.rkt` defining a simple substitution function can be found on the course website.

To use it, add the following at the top of your file:

```
(require "subst.rkt")
```

You can then use the metafunction `subst` like below. Notes how it avoids substitution for bound variables.

```
> (term (subst (x 1) (+ x)))
'(+ 1)
> (term (subst (x 1) (λ y ((+ x) y))))
'(λ y ((+ 1) y))
> (term (subst (x 1) (λ x ((+ x) y))))
'(λ x ((+ x) y))
```

We will see later how to extend this file to other types of variable declarations.

Exercise 2 (Call by Name)
Define a second reduction relation which uses different evaluation contexts to implements call-by-*name*. Show off some examples where call-by-value and call-by-name differs.

Bonus Write a non-deterministic semantics that can simulate both call by name and call by value. Does it always end up with the same result? If it doesn't, when?

Exercise 3 (Constants – Arithmetic operations)
Extend the language with arithmetic operations using the `define-extended-language` function provided by plt-redex. Reuse the initial reduction relation as much as possible. You might want to define a metafunction `delta` (or δ) to implement the application of constants.

$$\begin{array}{rcll} e & ::= & \ldots & \\ & | & \mathit{number} \mid + \mid \ldots & \text{Arithmetic operations} \end{array}$$

Exercise 4 (More constants)
Define some more constants of your choosing (booleans, lists, ...).