KØBENHAVNS UNIVERSITET

Assignment 1 - PLD

Mikkel Willén

22. februar 2023

Indhold

1. Lambda Calculus

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

$(x(\lambda x\ .\ x))[x \to y]$ using the first rule

$(y(\lambda x\ .\ x))$ replace x with y

2. PLD LISP

a

```
(define last (lambda (as) (car (reverse as))))
```

b

```
(define helperLeft
    (lambda ((a . b)) (if (<= (helperLeft a) (helperRight b)) (helperRight b))
            ()         ()
            (a)        a
            (a . ())   ()
            (() . a)   ()))

(define helperRight
    (lambda ((a . b)) (if (<= (helperLeft a) (helperRight b)) (helperLeft a))
            ()         ()
            (a)        a
            (a . ())   ()
            (() . a)   ()))

(define getLeft
    (lambda ((a . b)) (getLeft a)
            (a)        a))

(define minOrder?
    (lambda ((a . b)) (if (<= (helperLeft a) (helperRight b)) (getLeft a))
            ()         ()
            (a)        a
            (a . ())   ()
            (() . a)   ()))
```

c

```
type tree =
    | Node of tree * tree
    | Leaf of int

type Sexp =
    | Num of int
    | Error of string

let rec helperLeft(t : tree) : Sexp =
    match t with
        | Leaf n -> Num n
        | Node (lt, rt) ->
            let minL = helperLeft(lt)
            let minR = helperRight(rt)
            match (minL, minR) with
                | (Num l, Num r) ->
```

```
                    if l <= r then
                        minR
                    else Error "()"
                | _ -> Error "()"

and helperRight(t : tree) : Sexp =
    match t with
        | Leaf n -> Num n
        | Node (lt, rt) ->
            let minL = helperLeft(lt)
            let minR = helperRight(rt)
            match (minL, minR) with
                | (Num l, Num r) ->
                    if l <= r then
                        minL
                    else Error "()"
                | _ -> Error "()"

let rec getLeft(t : tree) : Sexp =
    match t with
        | Leaf n -> Num n
        | Node (lt, _) ->
            getLeft(lt)

let rec minOrder(t : tree) : Sexp =
    match t with
        | Leaf n -> Num n
        | Node (lt, rt) ->
            let minL = helperLeft(lt)
            let minR = helperRight(rt)
            match (minL, minR) with
                | (Num l, Num r) ->
                    if l <= r then
                        getLeft(t)
                    else Error "()"
                | _ -> Error "()"
```

I have done pretty much the same thing in both versions of the program. Main difference is that Lisp works on lists, while F works on a tree type I have defined.

d

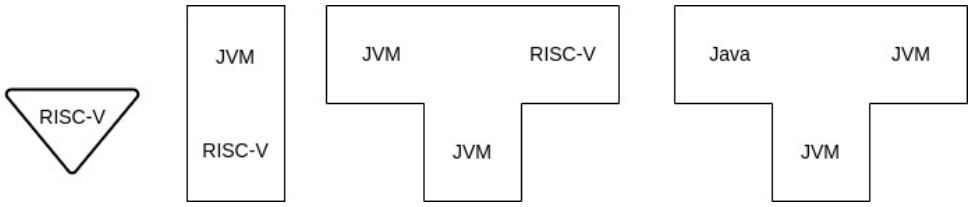
```
let unops = ["number?"; "symbol?"; "abs"]
```

```
and applyUnop x v =
  match (x, v) with
  | ("number?", Num _) -> v
  | ("symbol?", Symbol _) -> v
  | ("abs", Num s)-> Num (Math.Abs s)
  | _ -> Nil
```

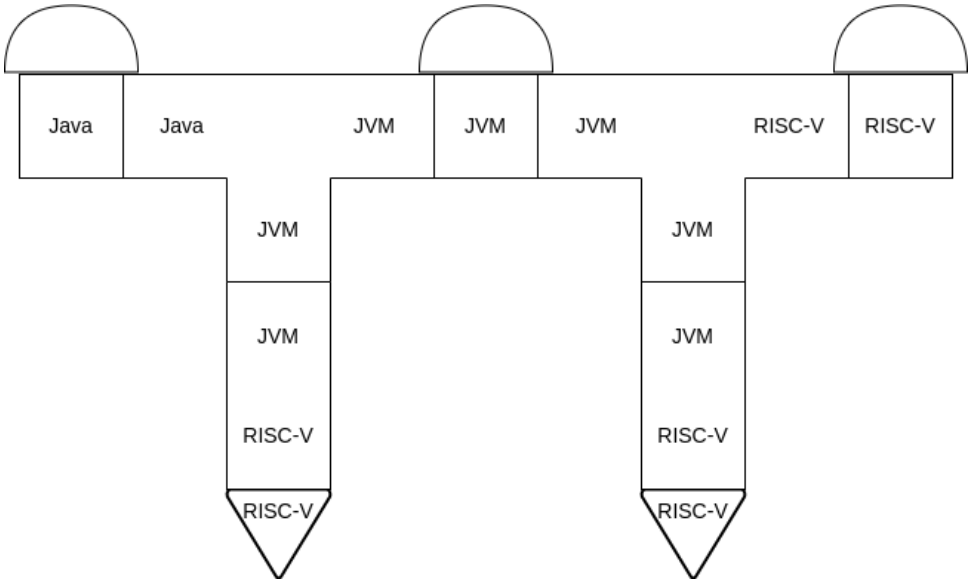
3. Boostraping

a

b



Figur 1: Bratman components



Figur 2: Compile Java to RISC-V

4. Syntax

Since it is possible to write your own syntax in Lisp, it has the potential to change the notation and make it difficult for somebody else to read. The pre-defined syntax of python makes it easier for another user to read the program, and understand what is does.
Fixing syntax errors in python is pretty easy, since gives an explanation and points to where it starts to not understand the code. In Lisp it is more difficult, since Lisp does not give a clear indication where the problem is. It says which line has a problem, but if this line calls another function and the problem is in the other function it might still give the error in the caller function, which makes it really difficult to find the error.
Lisp has a functions with "random"names as car, cdr and so on, which makes sense to people, who were around, when this was implemented, but not to new programmers. There are also a lot of parentheses in lisp, which makes it harder to get an overview of the the program. Python on the other hand, has functions, where the name for the most part pretty clearly indicates what the function is doing, and if you have been coding in python for some time, it is really easy to guess functions when writing in a IDE that completes the words. Since python uses indentation to indicate where functions start and stop, it is much easier to get an overview of the program. With all the parentheses in Lisp it can be pretty difficult to determine the scope of declarations. In python it is much easier, since it uses indentation to seperate code blocks.

5. Syntax

a

An advantages of making identifiers lexically distinct is, that it is much easier to recognize identifiers, when looking at the code. It can also be easier to compile or interpret, since these specific cases, immediately can be reconigze by the program. A disadvantages is, that it limits what the user can call their own functions and variables, and also, that some of these options, like underlining and boldface, is not normal in programming, and can be tedious to write, since you have to switch in and out of underlining or boldface mode.

b

One way of making keywords lexically distinct, could be to have all keywords be in snake_case and or start with a specific character like _. Again this limits the users posibilities of calling variables and functions what they want, though it makes it easier to regonize as keywords for a user.