PLD Assignment 1

Ask

16. februar 2022

Indhold

1 Lambda calculus

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

The above term is what I believe we call the "omega program"or "omega combinator"Ω. This specific program does not terminate when we use beta-reduciton on it. in fact, it reduces to itself.

$$x[x \rightarrow (\lambda x\ .\ (x\ x))]$$

$$\downarrow \beta\text{-reduction}$$

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

$$\downarrow \beta\text{-reduction}$$

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

This reduction can go on and on. The term is the samllest nonterminating program and is the basis for building out loops with lambda calculus if not other metod is presented.

2 PLD LISP

2.1 a) - PLD-LISP

Reading through the file *listfunctions.le* I've reused the code for returning the length of a list. I've restructured it to be able to return the sum of a list instead.

```
1    ; return the sum of a list
2    (defin sum
3        (lambda (())    0
4            ((a . as)) ( +a (sum as))))
```

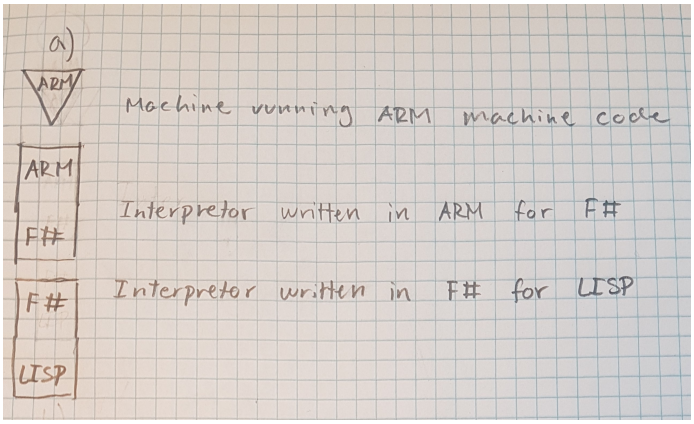
2.2 b) - Python

```
1    # return the sum of a list
2    def sum_of_list(lst):
3        sum = 0
4        for val in lst:
5            total = total + val
6        return total
```

I've chosen to implement the same type of function in python since I know python fairly well. Comparing the size of the two programs they are fairly similarly, with the python program being a couple of extra lines bigger compared to the PLD-LISP. This could possible add up if a person were to implement a bigger program.
Personally I know python better and I think it's easier to read that the PLD-LISP. PLD-LISP also has a numerous ammount of parantheses that can really mess with the programmer if you're not careful.
As for effort to program the two functions, I personally found the python program easier to make. Again, this is because I know python better than PLD-LISP, so in all fairness I don't think this is a very representable example of the question regarding "Syntaxes for size, readability and effor to program".

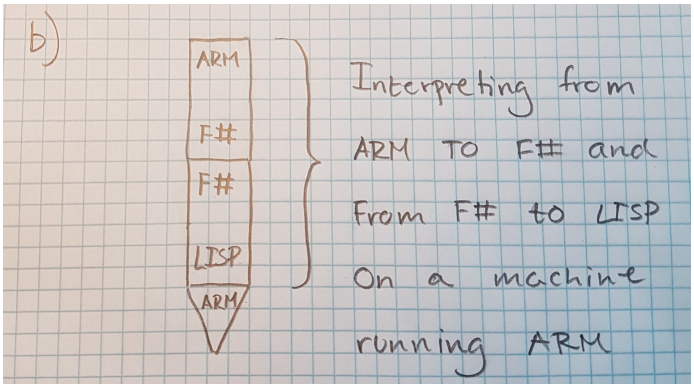
3 Bootstrapping

3.1 a)



Figur 1: Bratman diagrams for the components

3.2 b)



Figur 2: Bratman diagram for executing a LISP program

I would argue it's not possible to execute a program in LISP with the following components, unless you're allowred to rearrange the way the interpreters are setup and create a "new"interpreter that interpret from LISP → ARM.

meaning you interpret from:
ARM → F# → F# → LISP → LISP → ARM $\xrightarrow{\text{execute}}$ ARM-machine

4 Comments

4.1 a)

```
1 /* Initialize the variable a
2     a = 0;
3 /* Set the minimum size */
4     min = 100;
```

The first comment in the "program"is never closed, meaning that the variable a is never reached. In to be able to reach and use the variable a, the comment needs to be closed similarly with a " /* " at the end of line 1 similar the second comment in line 3.

4.2 b)