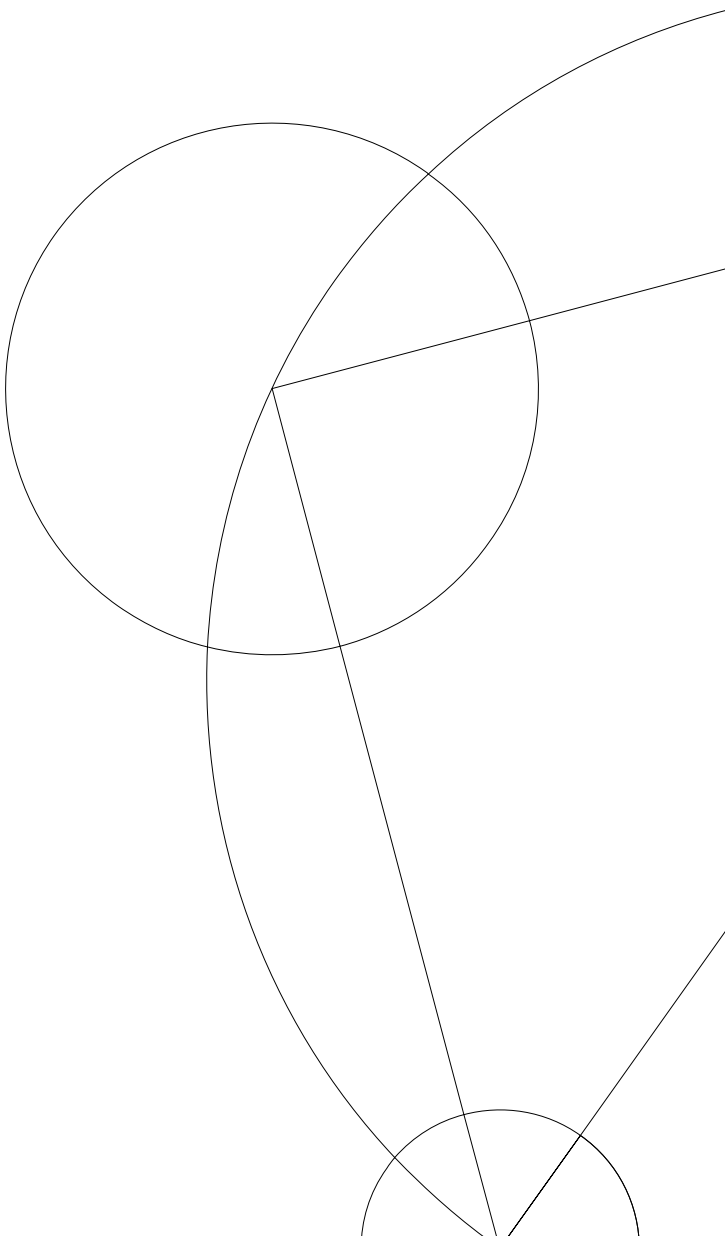
KØBENHAVNS UNIVERSITET

Noter

Adit (hjg708)

IPS - Assignment 1



10. maj 2023

Indhold

1 Task 1: A couple of lines about yourself **3**

2 Task 2: Getting Reacquainted with F#; Using AbSyn and Symbol Tables. **3**
2.1 Completeness 3
2.2 Correctness 4
2.3 Efficiency 4
2.4 Elegance 4
2.5 Other 4

3 Task 3: A First Non-Trivial Fasto Program **4**

1 Task 1: A couple of lines about yourself

My name is Adit, I am a DIKU Computer Science Bachelor student with the general profile, and I have IPS as a compulsory course. In terms of functional programming with F#, I would say that my overall proficiency is basic. I remember the basics from the "Programming and Problem Solving (PoP)"course, but I may need some brushing up and practice to become more proficient with more advanced concepts. I am familiar with the assembly programming as I used it in the Computer Systems (CompSys) course, for that course we used the RISC-V architecture. I look forward to learn about the various aspect of interpreting and translating programming languages and how we can implement a translation tool (Fasto).

2 Task 2: Getting Reacquainted with F#; Using AbSyn and Symbol Tables.

2.1 Completeness

In the new version of the implementation I decided to move my helper function inside the 'OVER' range operation, this makes it so that I no longer have to explicitly call the recursive function, as it is already contained withing the different range operations. As for the 'OPERATE' operations, I use simple pattern matching to calculate the different operations.

The follosing is a code snippet on how I implement the sum operation

```
| RSUM ->
  let rec sum (vtab : SymTab, v1 : VALUE) : VALUE =
    match v1, v2 with
      | INT(i), INT(j) ->
        if v2 < v1 then INT(0)
        else
          let v1' = eval vtab (OPERATE (BPLUS, CONSTANT(INT(1)), CONSTANT(v1))) in
          let vtab' = bind var v1' vtab in
          eval vtab (OPERATE (BPLUS, e3, CONSTANT(sum(vtab', v1'))))
  sum(vtab, v1)
```

The recursive function sum ensures that the operation calculate for all values, going from the e1 to e2. In the case of e2 being smaller than e1 it would return a zero. Using the previously defined OPERATE operation, I can ensure that v1 starts as 1. We then store the accumulated value in e3, using the OPERATE operation again.

2.2 Correctness

My implementation returned the correct results for all the tests I ran.

```
Evaluation result   : INT 1
Input an expression : 0
Evaluation result   : INT 0
Input an expression : -1
parse error
```

```
Input an expression : 5 + 5
Evaluation result   : INT 10
Input an expression : 5 - 10
Evaluation result   : INT -5
Input an expression : 5*5
Evaluation result   : INT 25
Input an expression : 5 * (0-5)
Evaluation result   : INT -25
Input an expression : (0-5) * (0-5)
Evaluation result   : INT 25
```

```
Input an expression : max x = 0 to 10 of 5 * x - x * x
Evaluation result   : INT 6
Input an expression : argmax x = 0 to 10 of 5 * x - x * x
Evaluation result   : INT 2
Input an expression : max x = 10 to 5 of 5 * x - x * x
empty range
Input an expression : argmax x = 10 to 10 of 5 * x - x * x
empty range
```

```
Input an expression : sum x = 1 to 4 of x * x
Evaluation result   : INT 30
Input an expression : sum x = 5 to 4 of x
Evaluation result   : INT 0
Input an expression : prod x = 1 to 4 of x * x
Evaluation result   : INT 576
Input an expression : prod x = 5 to 4 of x
Evaluation result   : INT 1
```

```
Input an expression : let x = 4 in x + 3
Evaluation result   : INT 7
Input an expression : let x0=2 in let x1=x0*x0 in let x2=x1*x1 in x2*x2
Evaluation result   : INT 256
```

My implementation can handle positive and negative numbers, in the case of negative number in the arithmetic function it is required that the inputted negative number is written as '(0-x)'. The implementation can also handle nested 'let x in y' statements. The implementation handles 'SUM', 'PROD', 'MAX' and 'ARGMAX' with no problem, although with large enough number the implementation will run into error, as the number will exceed the limit of the int type. As per the requirement the implementation will return 0 for an empty range in 'SUM' and return 1 in 'PROD'. And the new implementation now returns the correct result when e3 is negative.

2.3 Efficiency

The calculation of expression is performed recursively for 'sum', 'prod', 'max' and 'argmax' through different recursive functions depending on what calculation was selected. As for 'plus','minus' and 'times' I use simple pattern matching. The range operations, have time complexity proportional to the size of the range.

2.4 Elegance

The code contains instances of code duplication, in each of the range operations functions I define for v1' and vtab', I was unable to make those two variable into a global variable that I only have to define once.

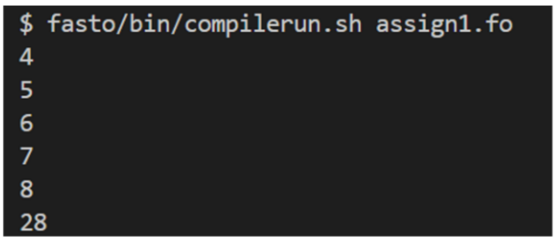
2.5 Other

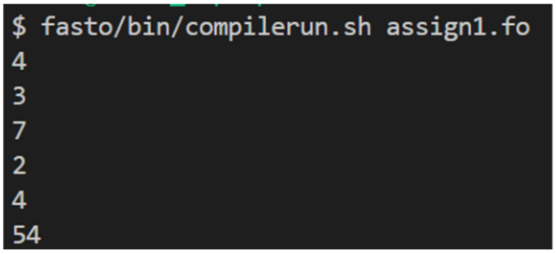
I was unsure whether or not I should make the functions for range operations outside or inside the OVER operation. If I moved it outside I would have to call those functions inside the OVER operation 'rop' match cases. In the end I decided to have it inside the OVER operation as it allowed the eval function to have a better overview, as it stays in chronological order.

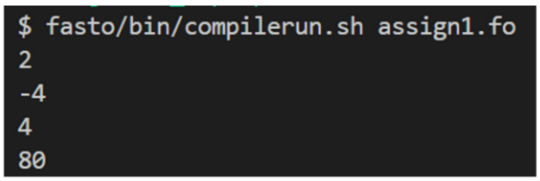
3 Task 3: A First Non-Trivial Fasto Program

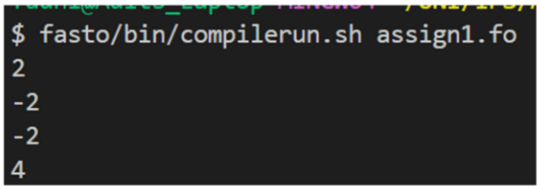
With the use of 'mul', 'make_arr', and 'difr_arr' functions, the implementation is able to correctly return the right results for the given inputs. The code contains the relevant descriptions for each functions and variables.

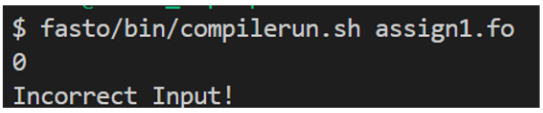
The following are the tests that I ran for the implementation of 'Assign1.fo'. I ran the tests using git bash.











It is relevant to mention that in the case of when the inputted 'n' is lower than 1 it will return an error message. The implementation can handle both positive and negative numbers with nu problem.

I also test the implementation by running the $ fasto/bin/runtests.sh' command in the terminal, this yields a 'success' in the terminal.

In the new version I have corrected the error message.