Rudresh Mishra
Raymond Klutse
ELU 610 Langage et Logique Project
23 Juin 2019

Exercise 1(*expl*)

What is a stack? What are the operations that you usually execute on a stack?

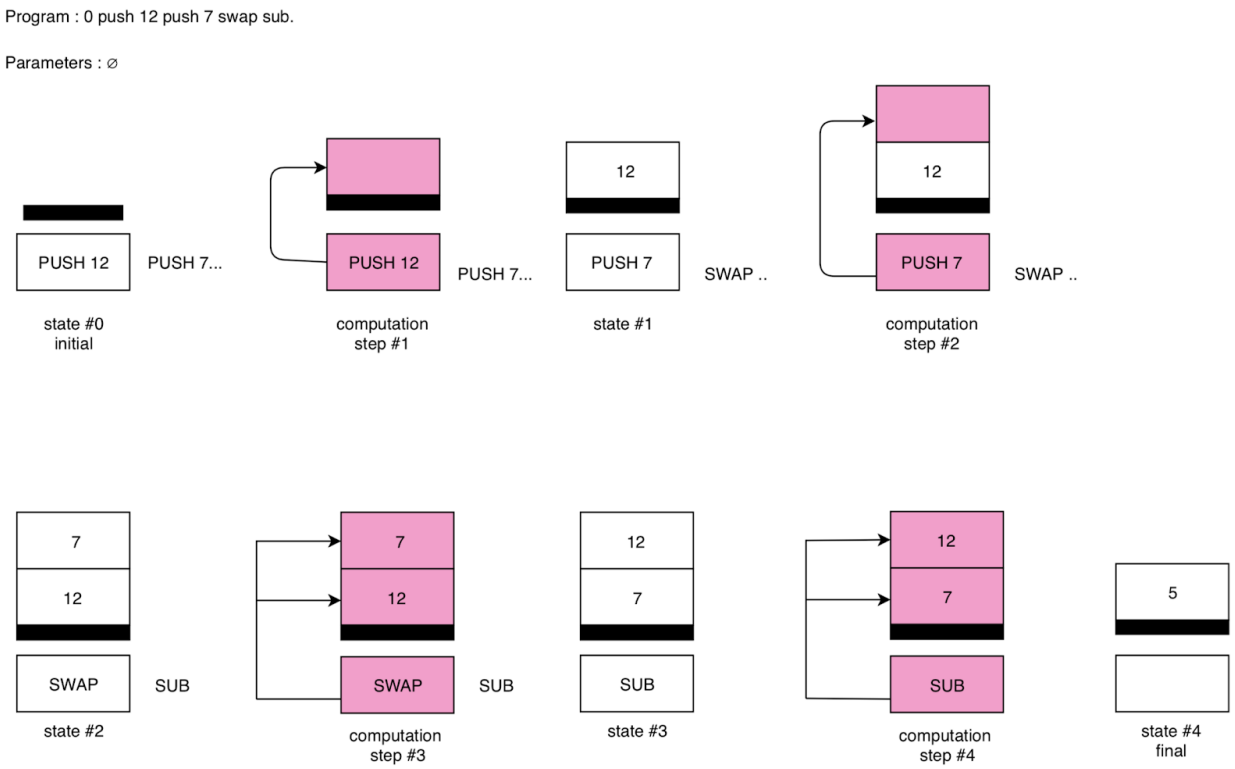
A stack is an Abstract Data Type(ADT) which is used as a collection of elements. It follows a LIFO(Last in First Out) format of accessing its data elements. That is to say, you can only access the last element inserted in the stack at a particular time.

Functions available in this data structure

Init : initializes the stack with no elements
isEmpty : return a boolean indicating if the stack is empty
isFull : return a boolean indicating if the stack is full
Pop : return the top element of the stack.
Push : used to insert an element in the stack.

Exercise 2(*expl*)

Detail in the same way the execution of 0 push 12 push 7 swap sub.



Rudresh Mishra
Raymond Klutse
ELU 610 Langage et Logique Project
23 Juin 2019

Exercise 3

Question 3.1 (expl):
Explain using plain words the semantics of programs.

Semantic 1

$$(1)\ \frac{i \neq n}{v_1, \ldots, v_n \vdash i, Q \Rightarrow \text{ERR}}$$

contexte subject conclusion

According to the premise, i is not equal to n. In the conclusion, we see that the elements v_1,...,v_n (n elements) are passed to the program while i elements are expected by the sequence of instruction Q according to $\vdash$ i,Q . Hence this program leads to an error.

Semantic 2

$$(2)\ \frac{Q, v_1 :: \ldots :: v_n :: \varnothing \rightarrow^* \text{ERR}}{v_1, \ldots, v_n \vdash n, Q \Rightarrow \text{ERR}}$$

Stack

According to the premise,it states that given that in one step, the execution of a sequence of instructions Q with Stack $v_1:: \ldots ::v_n :: \varnothing$ leads to an error, then the conclusion v_1,...,$v_n \vdash$ n,Q will also return an error.

Semantic 3

$$(3)\ \frac{Q, v_1 :: \ldots :: v_n :: \varnothing \rightarrow^* \varnothing, v :: S}{v_1, \ldots, v_n \vdash n, Q \Rightarrow v}$$

According to the premise, it states that, given that in one step the execution of a sequence of instructions Q with Stack $v_1:: \ldots ::v_n ::\varnothing$ leads to ∅(empty instruction sequence) and a stack with v element at the top of the stack,then the conclusion $v_1,\ldots,v_n \vdash$ n,Q returns element v.

Question 3.2 (math):
A case is still missing, spot it out and give the corresponding rule.

$$\frac{Q,v1 :: \ldots :: vn :: \varnothing \rightarrow * \varnothing,\varnothing}{v1, \ldots ,vn \vdash n,Q \Rightarrow Err}$$

According to the premise,it states that, given that in one step the execution of a sequence of instructions Q with Stack $v_1:: \ldots ::v_n ::\varnothing$ leads to ∅ (an empty instruction sequence) and ∅ (an empty stack) then conclusion $v_1,\ldots,v_n \vdash$ n,Q returns an Err.

Question 3.3 (math):
Give the rules describing the small step semantics for instruction sequences. Beware to cover all cases of runtime errors.

Considering a sequence of instructions **Q** , a stack **S**, a numeric element **V** and a set of arithmetic operations : **ADD**(“+”), **SUB**(“-”),**MUL**(“*”),**DIV**(“/”) , **REM**, the following are the rules describing the small step semantics for instruction sequences taking into consideration their runtime errors.

PUSH
Add element to the stack

PUSH.V.Q, S → Q ,V :: S

PUSH ERROR

Error occurs when the stack is full

N = number of expected elements

I = position of new element

I > N

PUSH.V.Q, S → Err

POP

Remove element from the top of the stack

POP.Q,V_{top}::S → Q,S

POP ERROR

Error occurs when the stack is empty

POP.Q,∅ → Err

SWAP

Change the position of two elements in the stack

SWAP.Q, V_1:: V_2:: ….. ::V_N::S → Q,V_2:: V_1:: .... ::V_N :: S

SWAP ERROR

Error occurs when the stack contains only one element

SWAP.Q, V :: ∅ → Err

SWAP.Q, ∅ → Err

ARITHMETIC OPERATIONS

Add,Subtract,Divide,Multiply and get the Remainder of an arithmetic operation

OPERATION ={ADD,SUB,DIV,MUL,REM}

OPERATION .Q, V_1:: V_2:: ….. ::S → Q,V_{result}:: .... ::V_N :: S

Where V_{result} is the result of the operation

ARITHMETIC OPERATIONS ERROR

Error occurs when elements in the operation are not numeric, and stack is empty

OPERATION .Q, V_1:: ∅ → Err

OPERATION .Q, ∅ → Err

Execption case

DIV .Q, V_1::0:: ∅ → Err

Exercise 4

Question 4.1 (code):

Propose the OCaml code for a type command describing the Pfx instructions. It should be in the file pfxAst.ml.

Code available in the "/Klutse_Raymond::Mishra_Rudresh/pfx/pfxAst.ml file"

Question 4.2 (code):

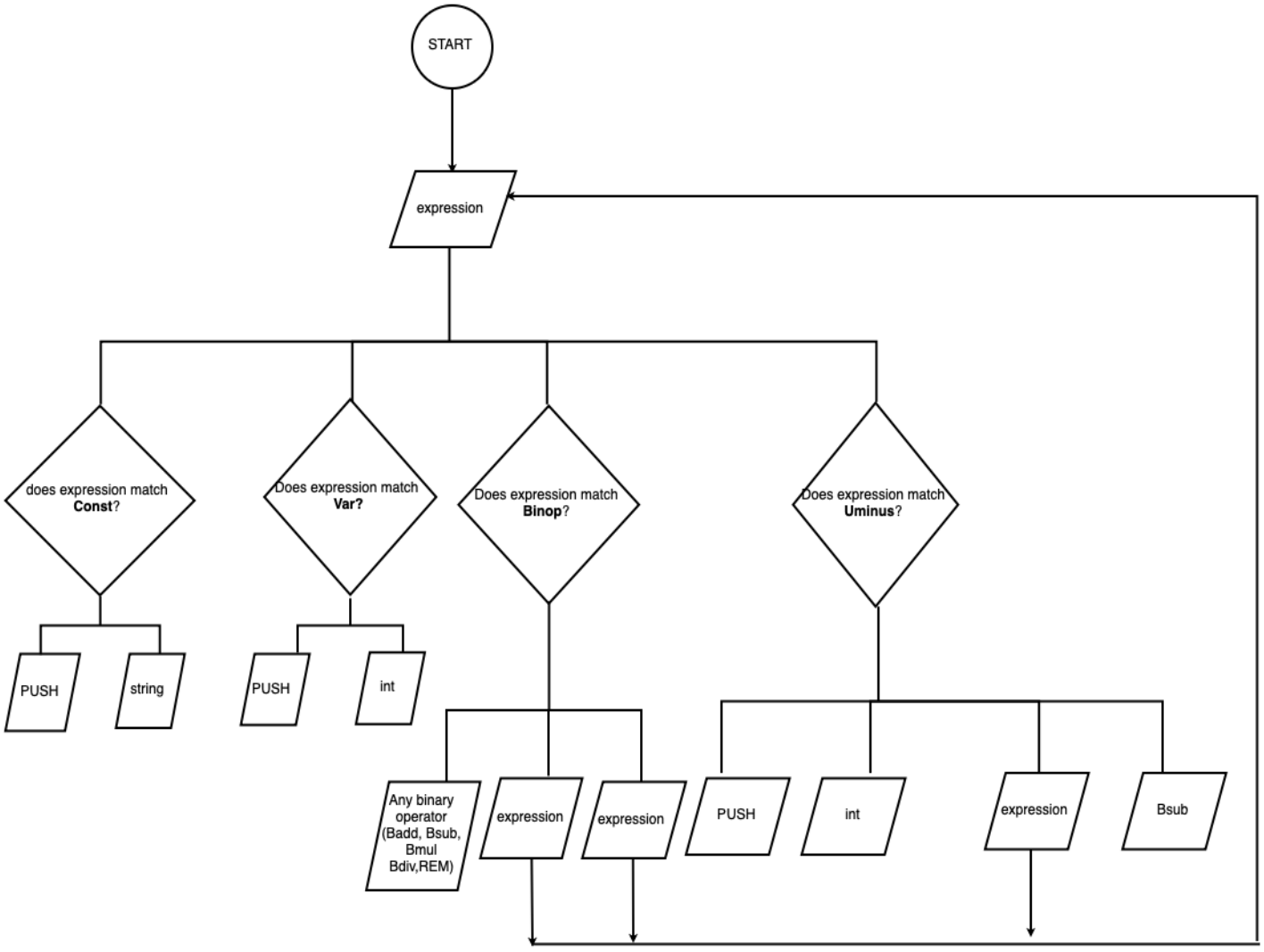
Write an OCaml function step that implements the small step reduction you defined above the Pfx instructions. It should be in the file pfxEval.ml.

Code available in the "/Klutse_Raymond::Mishra_Rudresh/pfx/pfxEval.ml file"

Rudresh Mishra
Raymond Klutse
ELU 610 Langage et Logique Project
23 Juin 2019

Exercise 5

Question 5.1 (math):
Propose a compilation schema of Expr in Pfx. Give its formal description. Notice that with the current definition of Pfx, we cannot implement variables. We defer their implementation to a later exercise.



With N is an integer numeral, Q is an instruction sequence and V is a variable:

➢ Q => PUSH . N . Q	(Const N)
➢ Q => PUSH . V . Q	(Var V)
➢ Q => expression . expression . (ADD | SUB | MUL | DIV | REM) . Q	(Binop (expression,expression))
➢ Q => PUSH . 0 . expression . SUB . Q	(Uminus expression)

Question 5.2 (code):
Define a function generate implementing the semantics you defined in previous question. It should be in the file exprToPfx.ml.

Code available in the “Klutse_Raymond::Mishra_Rudresh/expr/exprToPfx.ml” file

Exercise 6 (*A first* Pfx *lexer*)

Question 6.1 (code):
Write a lexer for the Pfx stack machine language. Complete the provided pfxLexer.mll. To test it without the parser, have a look at the file exprLexer_standalone.mll on Moo- dle.

Code available in the “Klutse_Raymond::Mishra_Rudresh/expr/pfxLexer.mll” file

Question 6.2 (code):
Reuse this code to be able to parse a file containing a Pfx program and prints all the tokens encountered in the process.

Code available in the “/Klutse_Raymond::Mishra_Rudresh/test_6_2.ml” file.

Exercise 7 (*Locating errors,* code)

Question 7.1 (code):

Modify your code from the previous exercise to be able to return the location of errors.

Modified the lexer to find the location:-
raise (Location.Error(Printf.sprintf "Illegal character '%c': " c ^"and the location is at "^Location.string_of (Location.curr lexbuf),Location.curr lexbuf));

Code available in the “Klutse_Raymond::Mishra_Rudresh/expr/pfxLexer.mll” file

Exercise 8 (*A first* Pfx *parser*)

Question 8.1 (code):
Write a parser for the Pfx stack machine language.

Code available in the “Klutse_Raymond::Mishra_Rudresh/pfx/pfxParser.mly” file.

Question 8.2 (code):
Test it in combination with your Lexer. To do it, you will have to write a function that prints the AST of Pfx. You should now use the provided file pfxVM.ml as the main file for the Pfx virtual machine. It is the file that should be given to ocamlbuildas a target. Notice that it requires that you modify slightly your lexer to remove the main func- tions and replace the token type definition by an open of the parser module.

Code available in the “Klutse_Raymond::Mishra_Rudresh/test_8_2.ml” file.

Exercise 9

Question 9.1 (expl):
Do we need to change the rules for the already defined constructs?

Yes, the code needs to modify in order to handle the executable sequence, exec and get defined as per the instructions.

Question 9.2 (math):
Give the formal semantics of these new constructions.

1-> (Q1).Q, V_1::....::V_N::S → Q, Q1:: .... ::V_N :: S

2-> exec.Q, Q1::V1::S → Q1.Q , V1::S

3-> get.Q, i::S → Q, V_i ::S

Question 9.3 (code):
If needed, extend the lexer and parser of Pfx to include these changes.

Lexer Code available in the “Klutse_Raymond::Mishra_Rudresh/pfx_fun/pfxLexer.mll”
Parser Code available in the “Klutse_Raymond::Mishra_Rudresh/pfx_fun/pfxParser.mly”

Exercise 10

Question 10.3 (code):
Provide a new version of generate.

Code available in the “Klutse_Raymond::Mishra_Rudresh/expr_fun/exprToPfx.ml” file.

Exercise 11

Question 11.2 (code):
What part of the code must be modified to get support for let?

The modified the lexer to suuport the let statement
(*Support let*)

Lexer Code available in the “Klutse_Raymond::Mishra_Rudresh/pfx_fun/pfxLexer.mll”