UNIVERSITY OF SOUTHAMPTON

ELECTRONICS AND COMPUTER SCIENCE

COMP2212 Programming Language Concepts

"salutLume"

User Manual

Raluca Diana Ispas

Hannah Alice Short

May 2017

Contents

1. Introduction

salutLume is a domain specific programming language which uses commands written entirely in Romanian and has been designed to perform set operations on finite languages. It is an interpreted language with OCaml being used to write the interpreter files. This manual introduces the user to the syntax and basic features of the *salutLume* language, giving an outline of how to interpret programs in the language, error messages that may arise and how to write an example program.

2. Using the *salutLume* Interpreter

The *salutLume* interpreter is composed of a lexer, parser and evaluator. It can be compiled on **linuxproj** using the `make` command. This produces an executable **mysplinterpreter** using the included Makefile. Users can then run their programs by passing both the program file and standard input to the interpreter as arguments. The following is an example:

```
mysplinterpreter yourProgram.spl < input
```

3. Overview of Syntax

3.1 Beginning and Ending a Program

Every program must start with `inceput` and end with `sfarsit`.

```
inceput <body> sfarsit
```

3.2 Declaration and Data Types

Variables in the language are declared using the keyword `lasa`. The four different variable types that can be declared are **integer**, **string**, **boolean** and **set**. These are all declared with their own unique symbol; '`[^]`' for integer, '~' for string, '`[?]`' for boolean, '`$`' for set. Example declarations are listed below:

```
lasa [^]intreg !!
lasa ~sir !!
lasa [?]boolean !!
lasa $set !!
```

3.3 Dealing with Input and Output

Inputs are used in the program with the set declaration symbol ('`$`') and the keyword `intrare`. They are indexed from 0; for example, the first program input is defined as `$intrare0`. Input must be given to the program in the format

L_1
L_2
…
L_n
k

where L_n is a finite language enclosed in curly braces with each element separated by a comma; and k is a positive integer representing the number of words to be output. The empty word can be included as an input, denoted as ':'.

Output can be given using the print command `afiseaza`. For example, `afiseaza $set1 !!` would print all elements in set1 to standard output. The output is ordered lexico-graphically; for instance, using a program prefixing 'a' to the start of each word as an example:

Input	Output
{a, d, b, c} 5	{aa, ab, ac, ad}

Further to this, the output does not allow repetition in the printed language. For example:

Input	Output
{a, a, b, c} 5	{aa, ab, ac}

3.4 Statements

3.4.1 Statement Syntax

'<-' is used to assign an operation value to a variable; the variable on the left of the symbol stores the result computed from the operation on the right of the symbol.
'!!' is used to mark the termination of each statement. Failure to do so will result in an error.

3.4.2 Conditional Statement Syntax

Conditional statements in the language can either be written in the format

`daca` <condition> `atunci` <statement> `opreste` or

`daca` <condition> `atunci` <statement1> `altfel` <statement2> `opreste`

with `altfel` being optional.

3.5 Loops

Only finite loops are implemented in the language in order to deal with input languages adequately. These follow the syntax:

`pentru` <element> `in` <set> `executa` <statement> `termina`

Loops are recommended to be used for set construction:

```
pentru ~sir in $intrare0 executa $set <- $set adauga ~sir !! termina
```

3.6 Comments

For the programmer's purpose, comments can be added to the source code to give annotation to programs. The interpreter is designed to ignore all words enclosed between '`/*`' and '`*/`':

```
lasa ~sir!! /*This is a comment showing how a variable is declared*/
```

4. Features

4.1 Set Operators

A variety of pre-defined functions are included in the language in order to perform operations on sets:

- The union of two sets can be obtained using the keyword `adauga` between two set variables
- One set can be subtracted from another using the keyword `elimina` between two set variables.
- The '`|-|`' operation can be used to concatenate either a string with a set or a string with a string.
- Intersection can be programmed by simultaneously looping round two sets and constructing a new intersection set based on the equivalence of elements checked during the iterations through the loops.

4.2 Primitive Mathematical Operators

salutLume supports a range of fundamental mathematical operations such as '+', '-', '*', '/' and '%'. '(' and ')' can be used as a means to give precedence to certain operations.

4.3 Boolean Operators

The language makes use of a number of traditional Boolean operators in a Romanian style: "`< ca`", "`> ca`", "`< sau =`" and "`> sau =`" act as **less than**, **greater than**, **less than or equal to** and **greater than or equal to** respectively. "`egal`" and "`diferit`" are keywords that can be used to test for equality between variables whilst "`sau`", "`si`" and "`nu`" are representing **or**, **and** and **not**. The Boolean primitives of the language are "`adevarat`" and "`fals`" – true and false.

5. Type checking and Error messages

salutLume is equipped to deal with various types of error:

- If a variable has been wrongly bound the program will output "<variable> `Need to be of type` <relevant data type> ".
- Should a variable be used which hadn't been declared, there will be the error "<set name> `not found`".
- When dealing with integers, "`Not divisible by zero`" captures the case whereby the programmer will attempt to divide an integer by a zero value.

6. Appendix

6.1 Dictionary

Romanian	English
Salut Lume	Hello World
inceput	start
sfarsit	end
lasa	let
intrare	input
afiseaza	print
daca	if
atunci	then
altfel	else
opreste	stop
pentru	for
executa	execute
termina	terminate
elimina	delete
egal	equal
diferit	different

6.2 Example Programs

Intersection: Take three languages L_1, L_2 and L_3 and output $L_1 \cap L_2$ and $L_1 \cap L_3$.

```
inceput
lasa $set1!!
lasa $set2!!
lasa $set3!!
lasa $set4!!
lasa $set5!!
pentru ~mesaj in $intrare0 executa $set1 <- $set1 adauga ~mesaj!! terminat
pentru ~mesaj in $intrare1 executa $set2 <- $set2 adauga ~mesaj!! terminat
pentru ~mesaj in $intrare2 executa $set3 <- $set3 adauga ~mesaj!! terminat
pentru ~mesaj in $set1 executa
      pentru ~mesaj1 in $set2 executa
            daca ~mesaj egal ~mesaj1 atunci $set4 <- $set4 adauga ~mesaj1!!
      oprit
      terminat
terminat
pentru ~mesaj in $set1 executa
      pentru ~mesaj1 in $set3 executa
            daca ~mesaj egal ~mesaj1 atunci $set5 <- $set5 adauga ~mesaj1!!
      oprit
      terminat
terminat
```

```
afiseaza $set4!!
afiseaza $set5!!
sfarsit
```

Union + Intersection + Concatenation:
Take languages L_1 and L_2 and output $L_1 \cup L_2$ concatenated with $L_1 \cap L_2$.

```
inceput
lasa $set1!!
lasa $set2!!
lasa $set3!!
lasa $set4!!
lasa $set5!!
lasa ~sir!!
pentru ~mesaj in $intrare0 executa $set1 <- $set1 adauga ~mesaj!! terminat
pentru ~mesaj in $intrare1 executa $set2 <- $set2 adauga ~mesaj!! terminat
$set3 <- $set2!!
pentru ~mesaj in $set1 executa $set3 <- $set3 adauga ~mesaj!! terminat
pentru ~mesaj in $set1 executa
pentru ~mesaj1 in $set2 executa
daca ~mesaj egal ~mesaj1 atunci $set4 <- $set4 adauga ~mesaj1!! oprit
terminat
terminat
pentru ~mesaj in $set3 executa
      pentru ~mesaj1 in $set4 executa ~sir <- ~mesaj|-| ~mesaj1!!
      $set5 <- $set5 adauga ~sir!!
      terminat
terminat
afiseaza $set5!!
sfarsit
```

6.3 Language BNF

baza::= INCEPUT program SFARSIT

program::= afirmatie | afirmatie program

afirmatie::= pentru_executa | daca_atunci_altfel | proces SEMNUL_EXCLAMARII

pentru_executa::= PENTRU VARSIR IN set EXECUTA program TERMINA |
PENTRU proces_boolean EXECUTA program TERMINA

daca_atunci_altfel::= DACA proces_boolean ATUNCI program OPRESTE |
DACA proces_boolean ATUNCI program ALTFEL program OPRESTE

proces::= operatie | declarare | afisare | mutare

operatie::= seteaza_proces| proces_intreg | proces_sir | proces_boolean

proces_intreg::= PARANTEZA_STANGA proces_intreg PARANTEZA_DREAPTA|
intreg_sau_variabilaIntreg | proces_intreg PLUS proces_intreg |
proces_intreg MINUS proces_intreg | proces_intreg INMULTIT proces_intreg |
proces_intreg IMPARTIT proces_intreg | proces_intreg REST proces_intreg |

MINUS proces_intreg %prec UMINUS intreg_sau_variabilaIntreg |
INT | VARINT

proces_sir::= sir_sau_variabilaSir| proces_sir CONCATENARE sir_sau_variabilaSir

sir_sau_variabilaSir::= SIR | VARSIR

proces_boolean::= PARANTEZA_STANGA proces_boolean PARANTEZA_DREAPTA |
boolean_sau_variabilaBoolean | proces_intreg MAI_MIC proces_intreg |
proces_intreg MAI_MARE proces_intreg |
proces_intreg MAI_MIC_EGAL proces_intreg |
proces_intreg MAI_MARE_EGAL proces_intreg |
proces_intreg EGAL proces_intreg | proces_intreg DIFERIT proces_intreg |
proces_sir EGAL proces_sir | proces_sir DIFERIT proces_sir |
proces_boolean EGAL proces_boolean |
proces_boolean DIFERIT proces_boolean |
proces_boolean SI proces_boolean | proces_boolean SAU proces_boolean |
NU proces_boolean

boolean_sau_variabilaBoolean::= ADEVARAT | FALS | VARBOOL

seteaza_proces::= set | set ADAUGA sir_sau_variabilaSir | set ELIMINA sir_sau_variabilaSir

set::= INPUT | VARSET

declarare::= LASA VARSET | LASA VARINT | LASA VARINT ATRIBUIRE proces_intreg |
LASA VARSIR | LASA VARSIR ATRIBUIRE proces_sir | LASA VARBOOL |
LASA VARBOOL ATRIBUIRE proces_boolean

mutare::= VARSET ATRIBUIRE seteaza_proces | VARINT ATRIBUIRE proces_intreg |
VARSIR ATRIBUIRE proces_sir | VARBOOL ATRIBUIRE proces_boolean

afisare::= AFISEAZA operatie