UNIVERSITY OF SOUTHAMPTON

ELECTRONICS AND COMPUTER SCIENCE

COMP2212 Programming Language Concepts

“salutLume”

User Manual

Raluca Diana Ispas

Hannah Alice Short

May 2017

# 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.

# 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

# Overview of Syntax

## Beginning and Ending a Program

Every program must start with inceput and end with sfarsit.

inceput

.

.

. sfarsit

* 1. 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 !!

* 1. 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

L1

L2

.

.

.

Ln

k

where Ln 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} |

* 1. Statements
     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 message.

* + 1. 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.

* 1. 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

* 1. Comments

For the purpose of the programmer, 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\*/

1. Features
   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. Finally, the ‘|-|’ operation can be used in order 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.

\*make this ^ less clunky – maybe bullet points or something, it’s a bit dull\*

\*I’ll edit it when we actually change it\*

* 1. 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.

* 1. Boolean Operators

The language makes use of a number of traditional (ill do this in the morning)

1. Type checking and Error messages
2. Appendix
   1. Syntax Dictionary
   2. Example Programs

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

* 1. 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