SPL User Manual

Vedant Chokshi - Riddhi Dhall

01 May, 2017

Contents

1 Introduction

Languages are sets of words over some alphabet. SPL is designed to perform simple operations on these languages. This user manual intends to explain the use of our SPL language, covering its syntax and usage as well as some additional features. SPL is a literate imperative programming language.

2 Data Types

2.1 Word

A word literal in SPL can be represented by writing a sequence of lowercase alphabets. For example, `word` is a world literal.

2.2 Language

A language/set in SPL can be represented by using curly braces with word literals in between that are delimited by a comma. For example, `{a, b, c}` generates a language consisting of the words `a`, `b` and `c`.

3 Syntax

3.1 Code Structure

SPL is built upon a sequence of statements where each each statement is delimited by a new line. **Each statement should either be a print statement or an assignment.**

3.2 Input

Input arguments can be referred to as variables which are called lang (followed by the line number of the language/set in the input file) e.g. `lang1` refers to the first language/set in the input file, `lang2` refers to the second language/set in the in the input file and etc. The input file format is shown in Appendix Section 5.2.

3.3 Output

The print operation can be used to output a language to the console in lexicographical order. It can be used as such:

```
print lang1
```

3.4 Operators

3.4.1 Assignment

Languages/sets and the results of operations (which are also languages/sets) can be bound to a variable by using the equals symbol. It can be used such:

```
var = {a, b, c}
```

3.4.2 Union

The union operation takes in two languages/sets as an input and returns their union. It can be used as such:

```
union lang1 lang
```

3.4.3 Intersection

The intersection operation takes in two languages/sets as an input and returns their intersection. It can be used as such:

```
inter lang1 lang2
```

3.4.4 Concatenation

The concatenation operation takes in two languages/sets as an input and returns a new language/set which consists of the concatenation of every element in the first language/set with every element in the second language/set. It can be used as such:

```
concat lang1 lang2
```

3.4.5 Kleene Star of a Word

This operation can used to generate a language/set which consists of the kleene star of a word. This can be done by putting a star next to the word. For example, to generate the language/set {:, a, aa, aaa, aaaa, ...}, we can do this:

```
a*
```

3.4.6 Generating all Words of a Fixed Length

This operation can be used to generate a language/set of all words of a fix length. This can be done by using square brackets (with length in between) after a language. For example, to generate language/set that consists of all words of length 2 over the words in a language/set, we can do this:

```
{a, b, c}[2]
```

3.5 Examples

Now that you have learnt the SPL syntax, here are a few example programmes to help you get started.

3.5.1 Intersection + Union

Take three languages lang1, lang2 and lang3 and produce languages $(lang1 \cup lang2) \cap lang3$

```
result = (union lang1 lang2)
print (inter result lang3)
```

3.5.2 Concatenation + Kleene Star + Union + Generate all Words of a Fixed Length

Take a language lang1 and produce the language which concatenates $a*$ with union of lang1 and the language that consists of all words of length 2 over the language {a, b, c, d}

```
print (concat a* (union lang1 {a, b, c, d}[2]))
```

4 Additional Features

4.1 Programmer Convenience

4.1.1 Literate Programming Style

Being a literate programming language, this means that it is extremely easy to read and write in this language. Furthermore, this style of programming allows programming in the order demanded by logic and flow of thoughts.

4.1.2 Parentheses

SPL supports the use of nested statements. Although all operators in SPL have the same precedence, it a good idea to make use of the parentheses in the language when using nested statements to make your code mode readable. For example, this statement:

```
print union lang1 concat {a} inter lang2 lang3
```

can easily be made more readable by using parentheses to group statements together:

```
print (union lang1 (concat {a} (inter lang2 lang3)))
```

4.1.3 Variables

Programming in this language is kept simple through the use of variables. This deems it unnecessary for a whole programme to be on one line and instead allows a smaller and simpler expression to be assigned and saved for later use in the program.

4.1.4 White Space Support

In order to improve readability of code we have supported the feature of white space which helps to stop code from looking cluttered.

4.2 Type Checking

SPL only consists of languages/sets and words which means that it is type safe as languages/sets can't be words and vice versa. Therefore, type checking was not required.

4.3 Error Messages

SPL consists of a variety of error messages to display an unexpected conditions that may occur. These error messages will help you debug your code and allow you to successfully write a working programme.

4.3.1 Parse Error

This error occurs when the programme executed has incorrect syntax that can not be recognised by the parser. An example of this involves using functions/operators that do not exist in the language.

```
Error: Programme cannot be parsed. Incorrect programme input.
```

4.3.2 Syntax Error

This error occurs when a statement in the programme does not print or assign values. For example just doing union lang1 lang2 would result in this error as it is not a printing or assigning statement.

```
Error: Statements should either print or assign values.
```

This error occurs when a variable name is expected in the syntax. For example just doing union lang1 would result in this error as union expects two inputs but only one is provided.

```
Error: Variable name expected
```

This error occurs when an operation expects two languages/sets but receives something unexpected. For example union lang1 (print lang2) would result in this error as (print lang2) does not return a language.

```
Error: Input cannot be interpreted as a set
```

4.3.3 Unbound Variable Error

This error occurs when a programme makes use of a variable that does not exist. This can easily be fixed by checking the programme syntax and ensuring that the unbound exists.

```
Error: Variable (variable-name) unbound
```

4.3.4 Invalid Input

This error occurs when there is an incorrect input file into the programme, meaning the interpreter could not recognise the input file languages/sets and hence this error occurs. This can easily be fixed by checking the input file and making sure all languages/sets are entered in the correct format.

```
Error: All languages should consist of words constructed from the English
lowercase alphabet and in the format - {abc, bca, ...}
```

This error occurs when the output size is less than 0. Like the error above, this can easily be fixed by checking the input file and making sure that the output size at the end is greater than or equal to 0.

```
Error: The number of words to be printed cannot be negative
```

5 Appendix

5.1 Running a programme

A programme can be executed on command line with the following command on bash:

```
./mysplinterpreter pr.spl < input
```

where pr.spl is the file containing the programme and input is the file containing the programme input (format shown in Appendix 5.2).

5.2 Programme Input

The inputs into the programme will take the following form:

```
L1
L2
.
.
.
Ln
k
```

where every line is a finite language and k is a positive integer which represents the output how many words are to be printed from each of the expected output languages/sets. An example of an input file can be:

```
{a, b, c}
{d, e, f}
5
```

5.3 SPL BNF

To help understand the SPL language more easily, here is the SPL BNF:

⟨*language*⟩	::= print (<expression>) | <word> = <expression>
⟨*expression*⟩	::= <function><set><set> | <word>* | <function> (<expression>) <set> | <function> (<expression>) (<expression>) | <set>[<number>]
⟨*function*⟩	::= union | inter | concat
⟨*set*⟩	::= {<part>}
⟨*part*⟩	::= <word> | <word> , <part>
⟨*word*⟩	::= <letters> | <letters><word>
⟨*letters*⟩	::= a | b | c | d | ... | z
⟨*number*⟩	::= <integer> | <integer><number>
⟨*integer*⟩	::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9