University of Southampton

COMP2212

Programming Language Concepts

The *Pepperoni* Language User Manual

A Domain Specific Language to solve set theory related problems

Author:
Shakib-Bin Hamid
Thomas Aley

Lecturer:
Pawel Sobocinski

March 19, 2015

Contents

1 Problem Domain

This is primarily a Domain Specific Programming Language made to solve set theory related problems e.g. union, concatenation, intersection, difference, kleene star etc. For example: $L_1 = \{a, b\}, L_2 = \{b, c\}$ then $L_1 \cup L_2 = \{a, b, c\}$ can be done in this language.

However, additional functionality also solves many problems in the Integer domain for example: generating fibonacci numbers etc.

2 Data Types and Manipulation

The languages are treated as mathematical sets which consist of finite strings built from any English alphabet and/or digits separated by comma. So, `{a, b, 123, "hello"}` is a valid input to the programs.

The supported primitive types in the language are *String*, *Integer* and *Boolean*.

2.1 Primitive Variable Declaration

Every non-control structure such as variable declaration *must* end with a semi-colon. So, a variable can be declared as let < *variable_name* >; Variable name can be any combination of letters, digits and underscores. Below is a code snippet demonstrating variable declaration.

```
/* Comments can be placed in this form */
let #x; /* # indicates Integer type.  Here #x is auto initialised to 0 */
let #y = 5; /* y is initialised to 5 */
let @my_string; /* @ indicates String type.  Here @my_string is initialised to "" */
let @hello = "hello world"; /* Strings must be surrounded by double quotes */
let ?ToF; /* ?  indicates Boolean type.  Here ToF is initialised to false */
let ?ocamlFun = true ; /* Here ocamlFun is initialised to true */
```

You must initialise the variable (if chosen to do so) correctly. For example: let `#x = "";` will show Fatal error: exception Parsing.Parse_error.

2.2 Primitive Operations

Every operation must be in the form < *primitive* >< *operation* >< *primitive* >; The original primitive is not mutated, so the value *must* be assigned to a variable for reference or the result of the operations can be directly printed.

2.2.1 Integer Operations

The language supports Integer Addition (+), Subtraction (-), Multiplication (*), Division (/), Modulo (%). They are all left associative with precedence in the order of $\% > */ > +-$ unless parenthesis are used to dictate order. Below are some code snippets.

```
/* Example Integer operations */
let #x = 3; let #y; let #z = 4;
#y = #y + 1; /* #y is incremented by 1 */
#x = #x + #y * #z; /* #x is now 7 because * has more precedence than + */
#y = (#x + #y) * #z; /* #y is now 32 because parenthesis overrode the precedence */
```

2.2.2 String Operations

The language also supports String concatenation (^). Below are some code snippets.

```
/* Example String operations */
let @s = "hello"; let @t = "world";
@s = @s ^ " " ^ @t; /* @s is now "hello world" */
```

2.2.3 Boolean Operations

The language supports Boolean operations: Less Than (<), Greater Than (>), Less Than or Equal (<=), Greater Than or Equal (>=), Equal (==), Not Equal (!=), AND (&&), OR (||), NOT (!). && , || are right associative whereas the rest are left associative except ! which is non-associative. The precedence is in the order of ! > (< , > , <= , >=) > (== , !=) > && > || .

Below are some code snippets.

```
/* Example Boolean operations */
let @s = "hello"; let ?b = true ; let #x = 5;
@s == "hello" || ?b && #x == 4; /* String & Integer equality.  Returns true */
(@s == "hello" || ?b) && #x == 4; /* Parenthesis overrides.  Returns false */
```

The language has the facility to print to the standard out for all primitive variables and actions as well as for the Set data structure that will be introduced shortly. The print statement always prints on a new line. Below are some code snippets.

```
/* Example print operations */
let @str = "Hello"; let #x = 10; let ?lightsOn = true ;
print @str ^ " world"; /* Prints Hello world */
print 5 + #x - 3; /* Prints 12 */
print ?lightsOn == true ; /* Prints true */
```

Performing a primitive operation on an undeclared variable will result in an error similar to Fatal error: exception Failure ("Variable #i Not Declared as an int type. Hint: Maybe try - let #i = 0;").

2.3 Set Data Structure

The language supports set data structure of type String with operations to add and remove elements. Input to the programs are read as sets from a file and stored as "input" followed by the line number (zero-based). For example: $input0 refers to the first input set. When printed, a set will appear like a mathematical set, with elements sorted lexicographically. Below is a code snippet.

```
/* Set operations */
let @str = "b"; let $aSet; /* $ Indicates a set.  It must not be initialised */
$aSet = $aSet add "a1"; /* Set operation work for string variables and primitives */
$aSet = $aSet add @str; print $aSet; /* Prints { a1, b } */
$aSet = $aSet remove "a1"; print $aSet; /* Prints { b } */
```

If a set operation is performed before the set has been declared, for example $aSet add ""; it will show Fatal error: exception Failure ("Set $aSet Not Found. Hint: Maybe try - let $aSet;") or when trying to print a set that is not declared it will show Fatal error: exception Failure ("Set Not Found. Hint: Maybe didn't declare it?")

3 Program Structure

Every program must start with begin and finish with end . Every program must have an input file directed to it. Below is the canonical example of a program in this language, in a file called `pr1.spl`

```
/* Your first program */
begin
      print "Hello World";
end
```

To run the program type `./mysplinterpreter pr1.spl < input`

4 Control Structures

The language supports conditional and loop statements to control program flow. These in turn can be nested.

4.1 Conditionals

Both short and complete if statements are supported. The structure is if < *condition* > then < *statement* > fi or if < *condition* > then < *statement* > else < *statement* > fi . Below are some code snippets.

```
/* Demonstration of conditionals */
let ?doMe = true ; let #x = 12;
if ?doMe then print "I am getting done"; fi

if #x < 0 then
      print "It will never happen";
else
      if !?doMe == false then
            print "Nested if triggered";
      fi
fi
```

4.2 Loops

Both while loops and enhanced for loops are supported. The structure is for < *condition* > do < *statement* > done for while loops and for < *string_variable* > in < *set* > do < *statement* > done for enhanced loops. The enhanced loops are able to iterate over input sets and programmer declared sets alike. Below are some code snippets.

```
/* Demonstration of conditionals */
let #i = 5; let $mySet; let #count;
$mySet = $mySet add "a";

for #i > 0 do /* #i must have been declared before */
      print #i; /* Prints the numbers 5 to 1 */
      #i = #i - 1; /* Remember to decrement #i to prevent infinite loop */
done

for @element in $mySet do /* @element must not have been declared previously */
      #count = #count + 1;
done
print "Number of elements:  "; print #count;
```

In the event of using a declared variable in an enhanced loop condition it will show an error similar to Fatal error: exception Failure ("Variable @element used already! Hint: Try a variable that is not already declared by a let declaration.").
Also, if the referred input set does not exist it will show an error similar to Fatal error: exception Failure ("Input set $input99 Not Found. Hint: $input0 refers to the 1st language in input file and so on.").

5 Example Programs

Contains 'n'

```
/* First n Fibonacci Numbers */
begin
      let #a = 0;
      let #b = 1;
      let #i; let #tmp;

      for #i < #OUTPUT_COUNT do
            print #a;
            #tmp = #a;
            #a = #b;
            #b = #tmp + #b;
            #i = #i + 1;
      done
end
```

Prints 0 1 1 2 ... 34 with each number on a new line.

Input file has 6 on line one

```
/* Kleene Star */
begin
      let $set;
      let @arg;
      let #count;

      for #count < #OUTPUT_COUNT do
            $set = $set add @arg;
            @arg = @arg ^ "a";
            #count = #count + 1;
      done
      print $set;
end
```

Prints { :, a, aa, aaa, aaaa, aaaaa }

Input file has {a, b} on line one, {b, c} on line two and 6 on line three

```
/* Difference of Two Sets */
begin
      for @inp in $input1 do
            $input0 = $input0 remove @inp;
      done
      print $input0 ;
end
```

Prints { a, c }

Input file contains n. Prints 3 5 7 ... on new line

```
/* Prime Numbers from 3 to n*/
begin
      let ?divisible; let #i = 3; let #j = 2;
      for #i < #OUTPUT_COUNT do
            ?divisible = false ;
            for #j < #i do
                  if #i % #j == 0 then
                        ?divisible = true ;
                  fi
                  #j = #j + 1;
            done
      #j = 2;
      if !?divisible then
            print #i;
      fi
      #i = #i + 1;
      done
end
```