CS315: Programming Languages

PROJECT PART 1

Büşra Oğuzoğlu	**21501655**
Aylin Çakal	**21401517**
Pegah Soltani	**21500559**

Section 02
GROUP 2

March 19, 2018

Complete BNF Description of SED

```
<program> ::=   begin  { <statements> }

<statements> ::= <statement><statements> | <statement>
<statement> ::= <declaration_statement> | <function_statement> |
<comment_statement> | <input_statement> | <output_statement> |
<loop_statement> | <return_statement> | <if_statement>

<if_statement> ::= <if_then_statement> | <if_then_else_statement>
<if_then_statement> ::= if (<boolean>) <statement>
<if_then_else_statement> ::= if (<boolean>) <statement> else
<statement>

<declaration_statement> ::= <string_declaration> |
<boolean_declaration> | <char_declaration> | <set_declaration> |
<digit_declaration> | <array_declaration>

<string_declaration> ::= string <var_id> <assignment_operator>
<string>
<boolean_declaration> :: boolean <var_id> <assignment_operator>
<boolean>
<digit_declaration> ::= digit <var_id> <assignment_operator> <digit>
<char_declaration> ::= char <var_id> <assignment_operator> <char>
<set_declaration> ::= set <var_id> <assignment_operator> <set>
<array_declaration> ::= <string_array> | <int_array> | <char_arrray>
| <set_array>

<string_array> ::= string[] <var_id> <assignment_operator>
[<string_array_elements>]
<string_array_elements> ::= <string> | <string> ,
<string_array_elements>

<digit_array> ::= digit[]<var_id> <assignment_operator>
[<digit_array_elements>]
<digit_array_elements> ::= <digit> | <digit>, <digit_array_elements>

<char_array> ::= char[] <var_id> <assignment_operator>
[<char_array_elements>]
```

```
<char_array_elements> ::= <char> | <char> , <char_array_elements>

<set_array> ::= set[] <var_id> <assignment_operator>
[<set_array_elements>]
<set_array_elements> ::= <set> | <set> , <set_array_elements>


<id> ::= <var_id> | <const_id>
<var_id> ::= <string>
<const_id> ::= const <string>

<assignment_operator> ::= =

<function_statement> ::= <function_call> | <function_decleration>
<function_call> ::= <var_id> <assignment_operator> <func_name>
(<parameter_list>)
<parameter_list> ::= <set> | <set> , <parameter_list>
<func_name> ::= <string>

<function_declaration> ::= <return_type>
<func_name>(<parameter_list>) { <statements> <return_statement> }

<return_statement> ::= return <return_type>
<return_type> ::= <boolean> | <set> | <relations>
<boolean> ::= TRUE | FALSE

<comment_statement> ::= /<statements>

<input_statement> ::= in >  <var_id> | <set>
<output_statement> ::= out < <string> | <set> | <boolean> | <var_id>
<loop_statement> ::= loop ( <boolean>) {<statements>}


<uppercase_letter> ::=
A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|R|S|T|U|V|Y|Z|Q|X
<lowercase_letter> ::=
a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|r|s|t|u|v|y|z|x|q
<letter> ::= <uppercase_letter> | <lowercase_letter>
<digit> ::= 0|1|2|3|4|5|6|7|8|9
<char> ::= ?|!|*|-|_|#|$|&|%|^|£|/|\| |
<string> ::= <letter> | <letter><string> | <digit> | <digit><string>
```

```
<set> ::= {<setElements>}
<setElements> ::= <element> |  <element>, <setElements>
<element> ::= <letter>|<digit>|<char>|<string>|<set>

<operation> ::= (<operation>) | (<oprt>)
<oprt> ::= <oprt> & <set> | <oprt> | <set> | <set> & <set> | <set> |
<set> | <oprt>’ | <set> | <oprt> \ <set> | <set> \ <set>

<relations> ::= <subset> | <superset>
<subset> ::= <set> ^ <set>
<superset> ::=  <set> ^^ <set>
```

Introduction

SED is a new language designed for sets. This language is similar to imperative languages with the main difference of only working with variables and expressions of the set type. SED provides 8 basic features that are set, set elements, variable identifiers, set operators, set relations, assignment operator, components of statements for input, output, loops, function definitions and function calls and comments.

In this language, some functions returns true or false by using variables, constants or nothing. Therefore, a function also corresponds to a truth value.

Explanation of the Language Construct

Terminal Symbols:

In this project, simple symbols used as terminals to adapt conventions and increase readability and writability. Those symbols are explained in the following section.

= : This terminal is used for assigning values.

( ) : These terminals are used to determine the scope of the arguments given in a function as parameters.

{ } : These terminals are used for determining the scope of the statements given in a function.

^ : This terminal is used for indicating the subset.

* : This terminal is used for determining the superset.

< : This terminal is used to indicate input statements.

> : This terminal is used to indicate output statements.

/ : This terminal is used to indicate comment lines.

& : This terminal is used to indicate a subset operation which is union

| : This terminal is used to indicate a subset operation which is intersection.

' : This terminal is used to indicate a subset operation which is complement.

\ : This terminal is used to indicate a subset operation which is difference.

Nonterminal Symbols:

Nonterminal symbols that are used in BNF description are explained in detail in the following sections. Explanatory names are chosen by purpose to increase readability, writability as well as reliability of the code that will be written using SED programming language.

<statements> : Statement can be composed of a single statement or a collection of statements.
<statement> : Statement can be one of the assignment, function, comment, input, output, loop or return statements.
<assignment_statement> : Assignment statement is composed of a variable ID, assignment operator and a set definition.
<function_statement> : Function statement is either a function declaration or a function call.
<function_decleration> : Function declaration is composed of return type, function name, parameter list, statements and a return statement.
<function_call> : Function call is composed of a variable ID, an assignment operator, function's name and a parameter list.
<comment_statement> : Comment statement can be composed by using "/" terminal and statements.
<input_statement> : Input statement is composed of "in" reserved word and ">" followed by a variable ID or a set definition.
<output_statement> : Output statement is composed of "out" reserved word and "< " followed by a variable ID, a set definition, a string, or a boolean value.
<loop_statement> : Loop statement is composed of "loop" reserved keyword followed by a boolean value that controls the loop, and statements inside the loop.
<return_statement> : Return statement is composed of "return" reserved keyword followed by the return type of the statement.
<return_type> : Return type is either boolean, set or relations.
<boolean> : Boolean is either "TRUE" reserved word or "FALSE" reserved word.
<uppercase_letter> : Uppercase letter is one of uppercase letters in the alphabet.
<lowercase_letter> : Lowercase letter is one of lowercase letters in the alphabet.

<letter> : Letter is either uppercase letter or lowercase letter.

<digit> : Digit is one of the digits from 0-9.

<char> : Char is one of the characters that is not a letter or a digit.

<string> : String consists of a single letter, a letter followed by a string, a digit or a digit followed by a string.

<set> : Set is composed of set elements.

<setElements> : Set Elements composed of either a single element or an element that is followed by another set of elements.

<element> : An element is either a single letter, a digit, a char, a string or a set.

<operation> : An operation is either a single operation or a complex operation such as (<operation> (<oprt>)) or ((<oprt>) <operation>)

<oprt> : There are four possible operations, union, difference,complement or intersection.

<relations> : Relation is either a subset or a superset relation.

<subset> : Subset relation is showed by ^.

<superset> : Subset relation is showed by *.

Description of Nontrivial Tokens

Identifiers

The users are free to use characters, digits, strings in any order they wish. So it will be easier for the user to read and write a program in SED.

Literals

Literals are TRUE and FALSE values which are in the booleans value description.

Reserved Words

The reserved words used in this language are not many. ‘Loop’ is a reserved word to indicate that the scanner should take it as a loop. ‘Return’ is a reserved word which appears before the return value in functions.

In order to increase the readability of the program, we used the reserved word ‘input’ for getting a parameter as an input, and output for giving the output of the code.

The Example Program:

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

y={a,b}
```

```
z={d}

/ y is subset of x => return TRUE
boolean subsetFunc (x,y){
  return x^y
}

/ x is superset of y => return TRUE
boolean supersetFunc (x,y){
  return x*y
}

/ z is not a subset of x => return FALSE
boolean subsetFunc (x,z){
  return x^z
}

/Union
{2,3{34}}&{a,b,x,y}

/Intersection
{abcd,c3fg5}|{abcd,{2,3,4,5}}

/Complement
{2,3,4}'
```

```
/Difference

{2,3,4} \ {2,3}

loop (FALSE){

  z = {!}

}

/Input & Output statements

in >> {44,46,72,36}

out << x


/Function for union

set unionFunc (x,y){

  return x&y

}


/Union function call

d=unionFunc(y,z)
```

Screenshots from Dijkstra Machine:

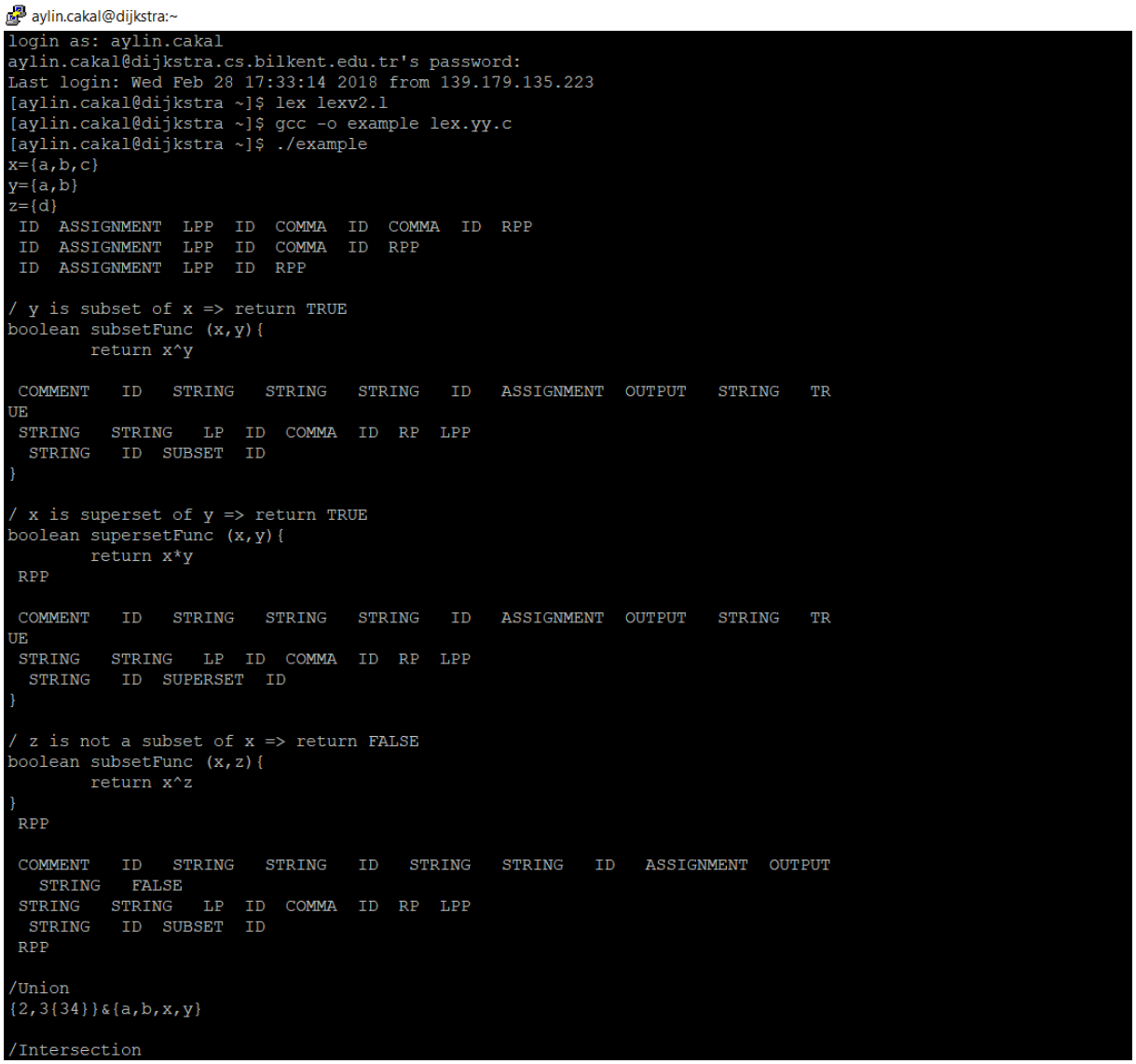

Figure 1: Screenshot 1

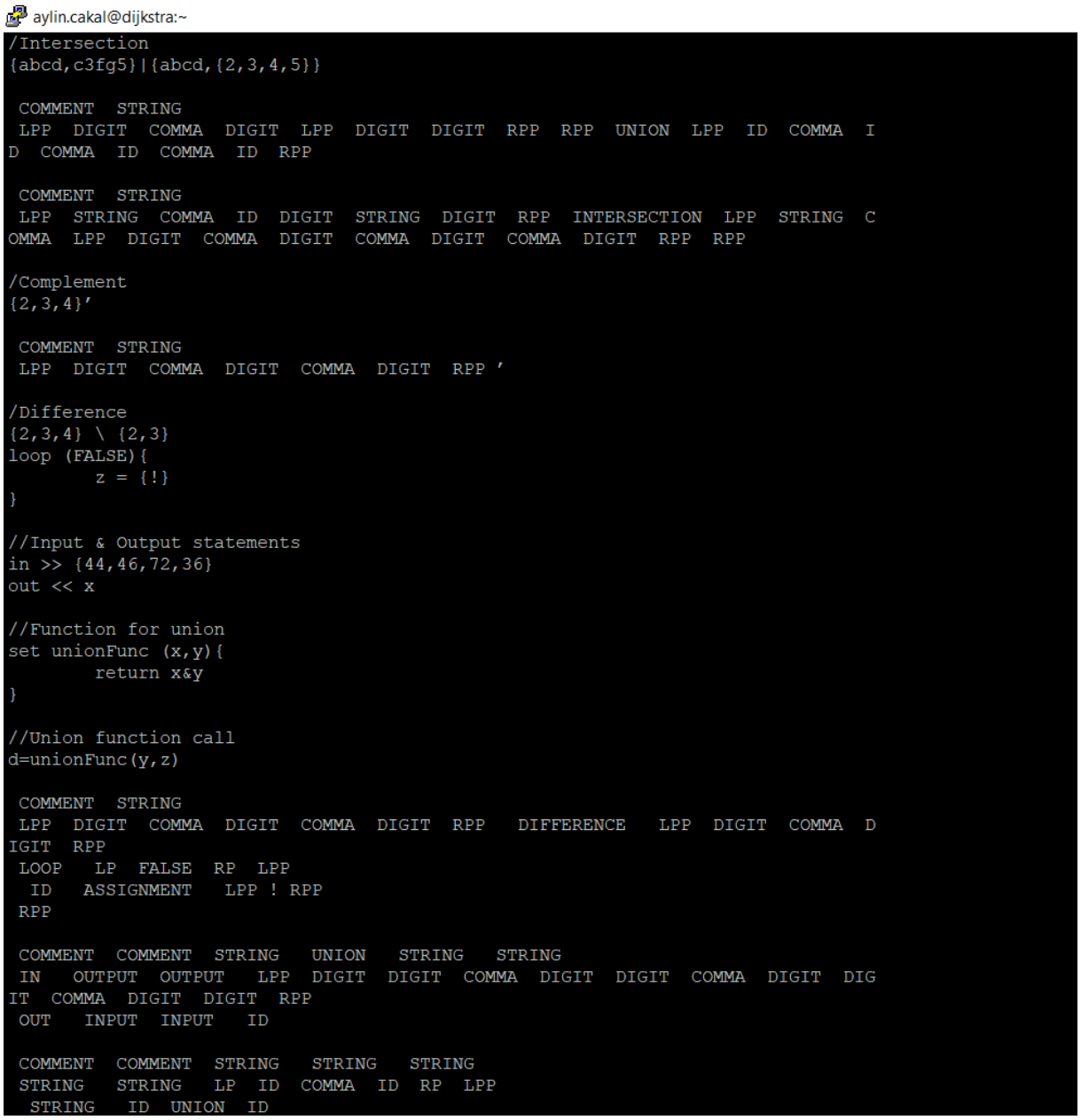

Figure 2: Screenshot 2

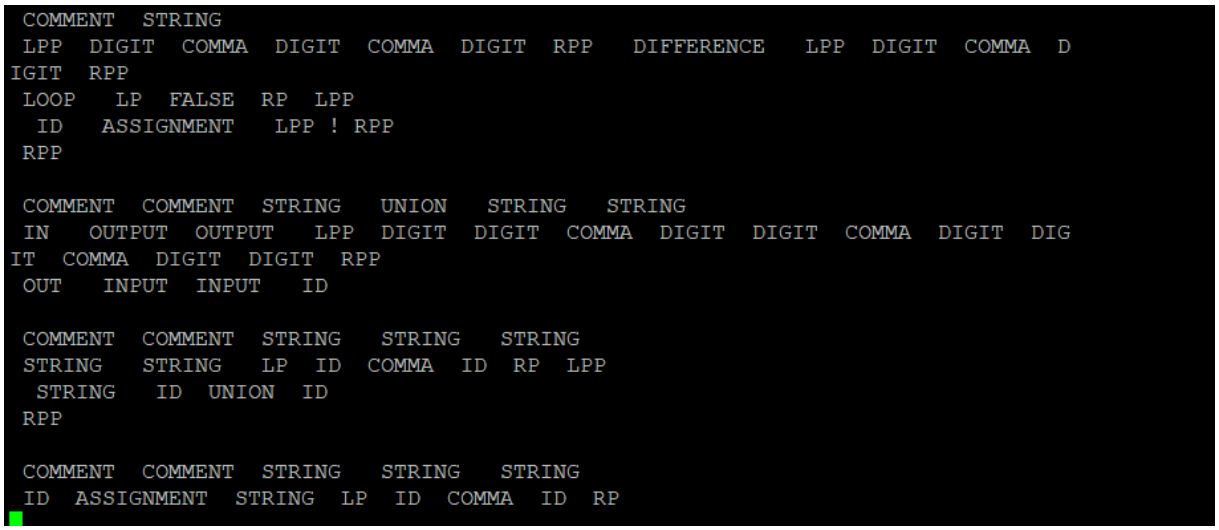

Figure 3: Screenshot 3

How We Dealt with Conflicts and Problems

In the execution of the parser, we got this message on dijkstra as follows:

- yacc: 10 rules never reduced.
- yacc: 37 shift/reduce conflicts, 20 reduce/reduce conflicts.

By tracing the “y.output” file, we tried to locate the rules that caused these conflicts. The most problematic rules that we dealt with were statements and relations. For this conflicts, we traced the parse trees of the rules and defined the ambiguous statements in the rules.

In addition, we got some errors such as undeclared identifiers but we could not fix it.