ALGONQUIN COLLEGE

21S_CST8152 Compilers

BNF TASK[1] FOR ASSIGNMENT 3.1

Team: Adam Di Cioccio - Id: **041019241**
/ **Sara Mehari** - Id: **041002736**

Note: *This task is part of the Assignment 3 from Compilers Course.*

ABRUZZO[2] LANGUAGE SPECIFICATION

NOTE 1

Change this file (starting with the **name** of your language) and check all BNF rules described here, adapting it to your language. Minimal requirements:

- *One method to use variables;*
- *Inputs and outputs (including string messages);*
- *Define mathematical expressions (using float-point variables).*

General View

This document focus on **ABRUZZO LS** (Language Specification) that is based on PLATYPUS language, originally created by Prof. Svillen Ranev for Algonquin College.

> ***Grammar, which knows how to control even kings . . .***
> —Molière, *Les Femmes Savantes* (1672), Act II, scene vi

A context-free grammar is used to define the lexical and syntactical parts of the **ABRUZZO** language and the lexical and syntactic structure of a **ABRUZZO** program.

[1] Adapted from resources developed by Prof. Svillen Ranev (Algonquin College, 2019)

[2] ABRUZZO (from Greek, "Wisdom") is also the name of the Bulgarian capital city, homeland form prof. Svillen Ranev, professor from Compilers for several years in the Algonquin College.

1. The ABR Lexical Specification

1. White Space

White space is defined as the ASCII space, horizontal and vertical tabs, and form feed characters, as well as line terminators. White space is discarded by the scanner.

<white space> → *one of* { `SPACE`, `TAB`, `NL` }

2. Comments

ABRUZZO supports single-line and multi-line comments: all the text from the ASCII characters **!!** to the end of the line is ignored by the scanner.

<comments> → # { sequence of ASCII chars } #

3. Variable Identifiers

The following variable identifier (VID) tokens are produced by the scanner: two kinds of arithmetic tokens: **IVID_T** (integer) and **DVID_T** (decimal numbers) and one kind of strings: **SVID_T**.

<variable identifier> → INT_T | DEC_T | STR_T

4. Keywords

The scanner produces a single token: **KW_T**. The type of the keyword is defined by the attribute of the token (the index of the keywordTable []). Remember that the list of keywords in ABR is given by:

main, in, print, int, decimal, string, if, then, else, while, do

5. Integer Literals

The scanner produces a single token: **INT_T** with an integer value as an attribute.

<integer_literal> → INT_T

6. Decimal Literals

DEC_T token with a real decimal value as an attribute is produced by the scanner.

<decimal_literal> → DEC_T

7. String Literals

STR_T token is produced by the scanner.

<string_literal> → STR_T

8. Separators

<separator> → *one of* { ( ) { } , ; }

Seven different tokens are produced by the scanner - **LPR_T**, **RPR_T**, **LBR_T**, **RBR_T**, **COM_T**, **EOS_T**.

9. Operators

<arithmetic operator> → *one of* { +, -, *, / }

A single token is produced by the scanner: **ART_OP_T**. The type of the operator is defined by the attribute of the token.

<string concatenation operator> → ++

A single token is produced by the scanner: **SCC_OP_T**.

<relational operator> → *one of* { >, <, ==, <> }

A single token is produced by the scanner: **REL_OP_T**. The type of the operator is defined by the attribute of the token.

<logical operator> → *one of* { && , ||, ! }

A single token is produced by the scanner: **LOG_OP_T**. The type of the operator is defined by the attribute of the token.

<assignment operator> → =

A single token is produced by the scanner: **ASS_OP_T**.

2. The ABRUZZO Syntactic Specification

1. ABRUZZO Program

1.1. Program

ABRUZZO program is composed by one special function: "MAIN" defined as follows.

```
<program> → main$ {
                <session>
        }
```

1.2. DATA

Variable Lists

The optional variable list declarations is used to define several datatype declarations:

<opt_varlist_declarations> → <varlist_declarations> | ϵ

Variable Declarations

<varlist_declarations> → <varlist_declaration>
| <varlist_declarations><varlist_declaration>

- **PROBLEM DETECTED: Left recursion – SOLVING FOR YOU:**

New Grammar

<varlist_declarations> → <varlist_declaration> <varlist_declarationsPrime>
<varlist_declarationsPrime> → <varlist_declaration> <varlist_declarationsPrime> | ϵ

Each variable declaration can be done as follows:

<varlist_declaration> → <integer_varlist_declaration>
| <decimal_varlist_declaration>
| <string_varlist_declaration>

1.3. Declaration of Lists:

The variables list declaration is defined here:

<integer_varlist_declaration> → **INT** <integer_variable_list>;

<decimal_varlist_declaration> → **DECIMAL** <decimal_variable_list>;

<string_varlist_declaration> → **STRING** <string_variable_list>;

1.4. List of Variables:

The list of variables is defined here:

Integers:

<integer_variable_list> → <integer_variable>
| <integer_variable_list>, <integer_variable>

- **PROBLEM DETECTED: Left recursion:**

New Grammar

<integer_variable_list>	**—>**	**<integer_variable><integer_variable_listPrime>**
<integer_variable_listPrime>	**—>**	**<integer_variable><integer_variable_listPrime>** | ϵ
<integer_variable>	→	**INT_T**

Decimals:

<decimal_variable_list>	→	<decimal_variable>
		| <decimal_variable_list>, <decimal_variable>

- **PROBLEM DETECTED: Left recursion:**

New Grammar

<decimal_variable_list>	**—>**	**<decimal_variable><decimal_variable_listPrime>**
<decimal_variable_listPrime>	**—>**	**<decimal_variable><decimal_variable_listPrime>** | ϵ

<float_variable>	→	DEC_T

Strings:

<string_variable_list>	→	<string_variable>
		| <string_variable_list>, <string_variable>

- **PROBLEM DETECTED: Left recursion:**

New Grammar

<string_variable_list>	**—>**	**<string_variable><string_variable_listPrime>**
<string_variable_listPrime>	**—>**	**<string_variable><string_variable_listPrime>** | ϵ

<string_variable>	→	STR_T

1.5. CODE session:

Optional Statements:

<opt_statements> → <statements> | ϵ

1.6. Statements

<statements> → <statement> | <statements> <statement>

- **PROBLEM DETECTED: Left recursion:**

New Grammar

<statements>	**—>**	**<statement><statementsPrime>**
<statementsPrime>	**—>**	**<statement><statementsPrime>** | ϵ

2. Statement

```
<statement> →  <assignment statement> | <selection statement> | <iteration statement>
               | <input statement> | <output statement>
```

2.1. Assignment Statement

```
<assignment statement>  →   <assignment expression>
```

2.2. Assignment Expression

```
<assignment expression> →  <integer_variable> = <arithmetic expression>
                           | <decimal_variable> = <arithmetic expression>
                           | <string_variable>= <string expression>
```

2.3. Selection Statement (if statement)

```
<selection statement> →  if (<conditional expression>)
                           then { <opt_statements> }
                           else { <opt_statements> } ;
```

2.4. Iteration Statement (the loop statement)

```
<iteration statement> →   while (<conditional expression>)
                            do { <statements>};
```

2.5. Input Statement

```
<input statement> → in (<variable list>);
```

Variable List:

```
<variable list> → <variable identifier> | <variable list>,<variable identifier>
```

- **PROBLEM DETECTED: Left recursion:**

New Grammar

```
<variable_list>          —>      <variable_indentifier><variable_listPrime>
<variable_listPrime>     —>      <variable_indentifier><variable_listPrime> | ϵ
```

Variable Identifier:

```
<variable identifier> → <integer_variable>
                        | <decimal_variable>
                        | <string_variable>
```

2.6. Output Statement

<output statement> → print (<opt_variable list>); | print (STR_T);

- **PROBLEM DETECTED: Left factoring (SOLVED for you here):**

New Grammar

<output statement> → print (<output statementPrime>);
<output statementPrime> → <opt_variable list> | STR_T

Optional Variable List:

<opt_variable list> → <variable list> | ϵ

- **Note:** In some cases, the grammar may be transformed to predictive grammar without applying the general rule. For example, the grammar above can be rewritten as follows.

- **Rewriting the grammar – SOLVED for you here:**

New Grammar

<output statement> → print (<output list>);
<output_list> → <opt_variable list> | STR_T

3. Expressions

3.1. Arithmetic Expression

<arithmetic expression> → <unary arithmetic expression> | <additive arithmetic expression>

Unary Arithmetic Expression:

<unary arithmetic expression> → - <primary arithmetic expression>
| + <primary arithmetic expression>

Additive Arithmetic Expression:

<additive arithmetic expression> →
<additive arithmetic expression> + <multiplicative arithmetic expression>
| <additive arithmetic expression> - <multiplicative arithmetic expression>
| <multiplicative arithmetic expression>

- **PROBLEM DETECTED: Left recursion:**

New Grammar

<additive_arithmetic_exp> —> <multiplicative_arithmetic_exp><additive_arithPrime>

<additive_arithPrime> —> + <multiplicative_arithmetic_exp><additive_arithPrime> | ϵ
<additive_arithPrime> —> - <multiplicative_arithmetic_exp><additive_arithPrime> | ϵ

Multiplicative Arithmetic Expression:

<multiplicative arithmetic expression> →
<multiplicative arithmetic expression> * <primary arithmetic expression>

| <multiplicative arithmetic expression> / <primary arithmetic expression>
| <primary arithmetic expression>

- **PROBLEM DETECTED: Left recursion:**

New Grammar
<multiplicative_arithmetic_exp> —> <primary_arithmetic_exp><multiplicative_arithPrime>

<multiplicative_arithPrime> —> + <primary_arithmetic_exp><multiplicative_arithPrime> | ϵ
<multiplicative_arithPrime> —> - <primary_arithmetic_exp><multiplicative_arithPrime> | ϵ

Primary Arithmetic Expression:

<primary arithmetic expression> → <integer_variable>
| <decimal_variable>
| DEC_T | INT_T
| (<arithmetic expression>)

3.2. String Expression

<string expression> →
<primary string expression> | <string expression> ++ <primary string expression>

- **PROBLEM DETECTED: Left recursion:**

New Grammar
<string_expression> —> <primary_string_expression><string_expressionPrime>
<string_expressionPrime> —> <primary_string_expression><string_expressionPrime> | ϵ

Primary String Expression:

<primary string expression> → <string_variable> | STR_T

3.3. Conditional Expression

<conditional expression> → <logical OR expression>

Logical OR Expression:

<logical OR expression> → <logical AND expression>
| <logical OR expression> **.OR.** <logical AND expression>

- **PROBLEM DETECTED: Left recursion:**

New Grammar
<logical_OR_expression> —> <logical_AND_expression> | <logical_OR_expressionPrime>
<logical_OR_expressionPrime> —> <logical_AND_expression> .OR.
<logical_OR_expressionPrime> | ϵ

Logical AND Expression:

<logical AND expression> → <logical NOT expression>
| <logical AND expression> **.AND.** <logical NOT expression>

- **PROBLEM DETECTED: Left recursion:**

New Grammar

<logical_AND_expression> —> <logical_NOT_expression> & <logical_AND_expressionPrime>
<logical_AND_expressionPrime> —> <logical_NOT_expression> .AND. <logical_AND_expressionPrime> | ϵ

Logical NOT Expression:

<logical NOT expression> → **.NOT.** <relational expression>
| <relational expression>

3.4. Relational Expression

<relational expression> →
<relational a_expression> | <relational s_expression>

Relational Arithmetic Expression:

<relational a_expression> →
<primary a_relational expression> == <primary a_relational expression>
| <primary a_relational expression> <> <primary a_relational expression>
| <primary a_relational expression> > <primary a_relational expression>
| <primary a_relational expression> < <primary a_relational expression>

- **PROBLEM DETECTED: Left factoring:**

New Grammar

<relational a_expression> —> <primary a_relational expression> <operator> <primary a_relational expression>
<operator> —> == | <> | > | <

Relational String Expression:

<relational s_expression> →
<primary s_relational expression> == <primary s_relational expression>
| <primary s_relational expression> <> <primary s_relational expression>
| <primary s_relational expression> > <primary s_relational expression>
| <primary s_relational expression> < <primary s_relational expression>

- **PROBLEM DETECTED: Left factoring:**

New Grammar

<relational s_expression> —> <primary s_relational expression> <operator> <primary s_relational expression>
<operator> —> == | <> | > | <

Primary Arithmetic Relational Expression:

<primary a_relational expression> → <integer_variable> | <decimal_variable> | DEC_T | INT_T

<primary s_relational expression> → <primary string expression>

Good luck with Assignment 3.1!