Nicholas Sturgeon
040911218

CST8152 – Compilers
Lab Section 013

Assignment 3

Presented to
Professor Sv. Ranev

Due Date
2019/12/05

Submission Date
2019/12/05

Contents

PLATYPUS Modified Syntactic Grammar and FIRST Sets

PLATYPUS LANGUAGE SPECIFICATION

1.2 Grammar Notation

<program> ->
PLATYPUS { <opt_statements> }

is merely a convenient abbreviation for:

<program> ->
PLATYPUS { <statements> }
| PLATYPUS { }

In this case the implied production for the optional element is:

<opt_statements> ->
<statements> | ∈

2. The PLATYPUS Lexical Specification

2.4 Variable Identifiers

<variable identifier> -> AVID_T | SVID_T

3 The PLATYPUS Syntactic Specification

3.1 PLATYPUS Program

<program> ->
PLATYPUS {<opt_statements>}

<statements> ->
<statement> | <statements> <statement>

(rearrange for convenient transform)
<statements> ->
<statements> <statement> | <statement>

(eliminate left recursion)
<statements> ->
<statement> <statements'>
<statements'> ->
<statement> <statements'> | ε

3.2 Statements

<statement> ->
<assignment statement>
| <selection statement>
| <iteration statement>
| <input statement>
| <output statement>

3.2.1 Assignment Statement

<assignment statement> ->
<assignment expression>;

< assignment expression> ->
AVID = <arithmetic expression>
| SVID = <string expression>

3.2.2 Selection Statement(the if statement)

<selection statement> ->
IF <pre-condition> (<conditional expression>) THEN { <opt_statements> }
ELSE { <opt_statements> } ;

3.2.3 Iteration Statement (the loop statement)

<iteration statement> ->
WHILE **<**pre-condition> **(<**conditional expression>**)**
REPEAT **{** <statements>**};**

<pre-condition> ->
TRUE | FALSE

3.2.4 Input Statement

<input statement> ->
READ (<variable list>);

<variable list> ->
<variable identifier> | <variable list>,<variable identifier>

(rearrange for convenient transform)
<variable list> ->
<variable list>,<variable identifier> | <variable identifier>

(eliminate left recursion)
<variable list> ->
<variable identifier> <variable list'>
<variable list'> ->
,<variable identifier> <variable list'> | ε

3.2.5 Output Statement

<output statement> ->
WRITE (<*opt_variable list*>);
| WRITE (STR_L);

(eliminate left factoring)
<output statement> ->
WRITE (<output list>);
<output list> ->
<opt variable list> | STR_L

3.3 Expressions

3.3.1 Arithmetic Expression

<arithmetic expression> - >
<unary arithmetic expression>
| <additive arithmetic expression>

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

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

<multiplicative arithmetic expression> ->
<multiplicative arithmetic expression> * <primary arithmetic expression>
| <multiplicative arithmetic expression> / <primary arithmetic expression>
| <primary arithmetic expression>
<primary arithmetic expression> ->
AVID_T
| FPL_T
| INL_T
| (<arithmetic expression>)

(eliminate left factoring)
<additive arithmetic expression> ->
<additive arithmetic expression> <additive arithmetic operator>
| <multiplicative arithmetic expression>
<additive arithmetic operator> ->
+ <multiplicative arithmetic expression>
| - <multiplicative arithmetic expression>

(eliminate left recursion)
<additive arithmetic expression> ->
<multiplicative arithmetic expression> <additive arithmetic expression'>
<additive arithmetic expression'> ->
<additive arithmetic operator> <additive arithmetic expression'> | ε

(eliminate left factoring)
<multiplicative arithmetic expression> ->
<multiplicative arithmetic expression> <multiplicative arithmetic operator>
| <primary arithmetic expression>
<multiplicative arithmetic operator> ->
*** <primary arithmetic expression>**
| / <primary arithmetic expression>

(eliminate left recursion)
<multiplicative arithmetic expression> ->
<primary arithmetic expression> <multiplicative arithmetic expression'>
<multiplicative arithmetic expression'> ->
<multiplicative arithmetic operator> <multiplicative arithmetic expression'>
| ε

3.3.2 String Expression

<string expression> ->
<primary string expression>
| <string expression> << <primary string expression>

<primary string expression> ->
SVID_T
| STR_T

(rearrange for convenient transform)
<string expression> ->
<string expression> << <primary string expression>
| <primary string expression>

(eliminate left recursion)
<string expression> ->
<primary string expression> <string expression'>
<string expression'> ->
<< <primary string expression> <string expression'> | ε

3.3.3 Conditional Expression

<conditional expression> ->
<logical OR expression>

<logical OR expression> ->
<logical AND expression>
| <logical OR expression> .OR. <logical AND expression>

<logical AND expression> ->
<relational expression>
| <logical AND expression> .AND. <relational expression>

(rearrange for convenient transform)
<logical OR expression> ->
<logical OR expression> .OR. <logical AND expression>
| <logical AND expression>

(eliminate left recursion)
<logical OR expression> ->
<logical AND expression> <logical OR expression'>

<logical OR expression'> ->
.OR. <logical AND expression> <logical OR expression'> | ε

(rearrange for convenient transform)
<logical AND expression> ->
<logical AND expression> .AND. <relational expression>
| <relational expression>

(eliminate left recursion)
<logical AND expression> ->
<relational expression> <logical AND expression'>
<logical AND expression'> ->
.AND. <relational expression> <logical AND expression'> | ε

3.3.4 Relational Expression

<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> < <primary a_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> < <primary s_relational expression>

<primary a_relational expression> ->
AVID_T
| FPL_T
| INL_T

<primary s_relational expression> ->
<primary string expression>

(eliminate left factoring)
<relational expression> ->
<primary a_relational expression> <relational operator>
<primary a_relational expression>
| <primary s_relational expression> <relational operator>
<primary s_relational expression>
<relational operator> ->
== | <> | > | <

3.4 FIRST sets

FIRST(<program>) = { KW_T(PLATYPUS) }

FIRST(<opt_statements>) = { AVID_T, SVID_T, KW_T(IF), KW_T(WHILE), KW_T(READ), KW_T(WRITE), ε }

FIRST(<statements>) = { AVID_T, SVID_T, KW_T(IF), KW_T(WHILE), KW_T(READ), KW_T(WRITE) }

FIRST(<statements'>) = { AVID_T, SVID_T, KW_T(IF), KW_T(WHILE), KW_T(READ), KW_T(WRITE), ε }

FIRST(<statement>) = { AVID_T, SVID_T, KW_T(IF), KW_T(WHILE), KW_T(READ), KW_T(WRITE) }

FIRST(<assignment statement>) = { AVID_T, SVID_T }

FIRST(<assignment expression>) = { AVID_T, SVID_T }

FIRST(<selection statement>) = { KW_T(IF) }

FIRST(<iteration statement>) = { KW_T(WHILE) }

FIRST(<pre-condition>) = { KW_T(TRUE), KW_T(FALSE) }

FIRST(<input statement>) = { KW_T(READ) }

FIRST(<variable list>) = { AVID_T, SVID_T }

FIRST(<variable list'>) = { COM_T, ε }

FIRST(<output statement>) = { KW_T(WRITE) }

FIRST(<output list>) = { STR_T, AVID_T, SVID_T, ε }

FIRST(<arithmetic expression>) = { ART_OP_T(PLUS), ART_OP_T(MINUS), AVID_T, FPL_T, INL_T, LPR_T }

FIRST(<unary arithmetic expression>) = { ART_OP_T(PLUS), ART_OP_T(MINUS) }

FIRST(<additive arithmetic expression>) = { AVID_T, FPL_T, INL_T, LPR_T }

FIRST(<additive arithmetic expression'>) = { ART_OP_T(PLUS), ART_OP_T(MINUS), ε }

FIRST(<additive arithmetic operator>) = { ART_OP_T(PLUS), ART_OP_T(MINUS) }

FIRST(<multiplicative arithmetic expression>) = { AVID_T, FPL_T, INL_T, LPR_T }

FIRST(<multiplicative arithmetic expression'>) = { ART_OP_T(MULT), ART_OP_T(DIV), ε }

FIRST(<primary arithmetic expression>) = { AVID_T, FPL_T, INL_T, LPR_T }

FIRST(<multiplicative arithmetic operator>) = { ART_OP_T(MULT), ART_OP_T(DIV) }

FIRST(<primary string expression>) = { SVID_T, STR_T }

FIRST(<string expression> = { SVID_T, STR_T }

FIRST(<string expression'>) = { SCC_OP_T, ε }

FIRST(<conditional expression>) = { AVID_T, FPL_T, INL_T, SVID_T, STR_T }

FIRST(<logical OR expression>) = { AVID_T, FPL_T, INL_T, SVID_T, STR_T }

FIRST(<logical OR expression'>) = { LOG_OP_T(OR), ε }

FIRST(<logical AND expression>) = { AVID_T, FPL_T, INL_T, SVID_T, STR_T }

FIRST(<logical AND expression'>) = { LOG_OP_T(AND), ε }

FIRST(<relational expression>) = { AVID_T, FPL_T, INL_T, SVID_T, STR_T }

FIRST(<primary a_relational expression>) = { AVID_T, FPL_T, INL_T }

FIRST(<primary s_relational_expression>) = { SVID_T, STR_T }

FIRST(<relational_operator>) = { REL_OP_T(EQ), REL_OP_T(NE), REL_OP_T(GT),
REL_OP_T(LT) }