Brady McIntosh – 040706980 Phillip Clarke – 040832994

PLATYPUS LANGUAGE SPECIFICATION

3 The PLATYPUS Syntactic Specification

3.1 PLATYPUS Program

FIRST = { PLATYPUS }
<program> ->
    PLATYPUS {<opt_statements>}

FIRST = { }
<statements> ->
    <statement> | <statements> <statement>

FIRST = { AVID_T, SVID_T, IF, WHILE, READ, WRITE, ε }
<opt_statements> ->
    <statements> | ε

FIRST = { AVID_T, SVID_T, IF, WHILE, READ, WRITE }
<statements> ->
    <statement> <statements'>
(changed according to assignment document)

FIRST = { AVID_T, SVID_T, IF, WHILE, READ, WRITE, ε }
<statements'> ->
    <statement> <statements'> | ε
(changed according to assignment document)

3.2 Statements

FIRST = { AVID_T, SVID_T, IF, WHILE, READ, WRITE }
<statement> ->
    <assignment statement>
    | <selection statement>
    | <iteration statement>
    | <input statement>
    | <output statement>

3.2.1 Assignment Statement

FIRST = { AVID_T, SVID_T }
<assignment statement> ->
    <assignment expression>;

FIRST = { AVID_T, SVID_T }
< assignment expression> ->
AVID = <arithmetic expression>
| SVID = <string expression>

3.2.2 Selection Statement(the if statement)

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

3.2.3 Iteration Statement (the loop statement)

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

FIRST = { TRUE, FALSE }
<pre-condition> ->
TRUE | FALSE

3.2.4 Input Statement

FIRST = { READ }
<input statement> ->
READ (<variable list>);

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

FIRST = { AVID_T, SVID_T, ε }
<variable list> ->
<variable identifier> <variable list^{l}>
(eliminate left recursion)

FIRST = { ,, ε }
<variable list^{l}> ->
, <variable list> | ε
(new predictive production)

3.2.5 Output Statement

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

FIRST = { WRITE }
<output statement> ->
WRITE (<output_list>);
(changed according to assignment document)

FIRST = { AVID_T, SVID_T, STR_T, ε }
<output_list> ->
<opt_variable_list> | STR_T
(changed according to assignment document)

3.3 Expressions

3.3.1 Arithmetic Expression

FIRST = { +, -, AVID_T, FPL_T, INL_T, (}
<arithmetic expression> - >
<unary arithmetic expression>
| <additive arithmetic expression>

FIRST = { +, - }
<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>

FIRST = { AVID_T, FPL_T, INL_T, (}
<additive arithmetic expression> ->
<multiplicative arithmetic expression> <additive arithmetic expression$^{\prime}$>
(eliminate left recursion)

FIRST = { +, -, ε }
<additive arithmetic expression$^{\prime}$> ->
+ <additive arithmetic expression> (print here)
| - <additive arithmetic expression> (print here)
| ε
(new predictive production & recursion)

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

FIRST = { AVID_T, FPL_T, INL_T, (}
<multiplicative arithmetic expression> ->
<primary arithmetic expression> <multiplicative arithmetic expression$^{|}$>
(eliminate left recursion)

FIRST = { *, /, ε }
<multiplicative arithmetic expression$^{|}$> ->
* <multiplicative arithmetic expression> (print here)
| / <multiplicative arithmetic expression> (print here)
| ε
(new predictive production & recursion)

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

3.3.2 String Expression

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

FIRST = { SVID_T, STR_T }
<string expression> ->
<primary string expression> <string expression$^{|}$>
(eliminate left recursion)

FIRST = { <<, ε }
<string expression$^{|}$> ->
<< <string expression> | ε
(new predictive production)

FIRST = { SVID_T, STR_T }
<primary string expression> ->
SVID_T
| STR_T

3.3.3 Conditional Expression

FIRST = { AVID_T, FPL_T, INL_T, SVID_T, STR_T }
<conditional expression> ->
    <logical OR expression>

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

FIRST = { AVID_T, FPL_T, INL_T, SVID_T, STR_T }
<logical OR expression> ->
    <logical AND expression> <logical OR expression^{I}>
(eliminate left recursion)

FIRST = { .OR., ε }
<logical OR expression^{I}> ->
    .OR. <logical OR expression>
    | ε
(new predictive production)

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

FIRST = { AVID_T, FPL_T, INL_T, SVID_T, STR_T }
<logical AND expression> ->
    <relational expression> <logical AND expression^{I}>
(eliminate left recursion)

FIRST = { .AND., ε }
<logical AND expression^{I}> ->
    .AND. <logical AND expression>
    | ε
(new predictive production)

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>

FIRST = { AVID_T, FPL_T, INL_T, SVID_T, STR_T }
<relational expression> ->
        <primary a_relational expression> <a_relational operation>
        | <primary s_relational expression> <s_relational operation>
(eliminate unpredictability)

FIRST = { ==, <>, >, < }
<a_relational operation> ->
        == <primary a_relational expression>
        | <> <primary a_relational expression>
        | > <primary a_relational expression>
        | < <primary a_relational expression>
(new predictive production)

FIRST = { ==, <>, >, < }
<s_relational operation> ->
        == <primary s_relational expression>
        | <> <primary s_relational expression>
        | > <primary s_relational expression>
        | < <primary s_relational expression>
(new predictive production)

FIRST = { AVID_T, FPL_T, INL_T }
<primary a_relational expression> ->
          AVID_T
        | FPL_T
        | INL_T

FIRST = { SVID_T, STR_T }
<primary s_relational expression> ->
        <primary string expression>
```