Kevin Lai 040812704
Xingyi Wu 040887028

PLATYPUS LANGUAGE SPECIFICATION

3 The PLATYPUS Syntactic Specification

3.1 PLATYPUS Program

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

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

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

Fixed Left Recursion
<statements> ->
<statements> <statement> | <statement>

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

<statements'> -> <statement><statements'> | ∈
FIRST (<statements'>) = { AVID_T, SVID_T, KW_T(IF), KW_T(WHILE), KW_T(READ), KW_T(WRITE),∈ }

Kevin Lai 040812704
Xingyi Wu 040887028

3.2 Statements

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

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

3.2.1 Assignment Statement

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

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

3.2.2 Selection Statement(the if statement)

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

3.2.3 Iteration Statement (the loop statement)

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

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

Kevin Lai 040812704
Xingyi Wu 040887028

3.2.4 Input Statement

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

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

Fixed Left Recursion
<variable list> ->
<variable list>,<variable identifier> | <variable identifier>

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

<variable list'> -> ,<variable identifier> <variable list'> | ∈
FIRST (<variable list'>) = { COM_T, ∈ }

<variable identifier> -> AVID | SVID
FIRST (<variable identifier>) = { AVID_T, SVID_T }

3.2.5 Output Statement

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

Applying Left Factoring
<output statement> -> WRITE (<output_list>);
FIRST(<output statement>) = { KW_T(WRITE) }

<output_list> -> <*opt_variable list*> | STR_T;
FIRST (<output_list>) = {AVID_T, SVID_T, STR_T, ∈ }

<*opt_variable list*> -> <variable list> | ∈
FIRST (<*opt_variable list* >) = { AVID_T, SVID_T, ∈ }

Kevin Lai 040812704
Xingyi Wu 040887028

3.3 Expressions

3.3.1 Arithmetic Expression

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

<unary arithmetic expression> ->
- <primary arithmetic expression>
| + <primary arithmetic expression>
FIRST (<unary arithmetic expression>) = { -, +}

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

Fixed Left Recursion
<additive arithmetic expression> ->
<multiplicative arithmetic expression><additive arithmetic expression'>
FIRST (<additive arithmetic expression>) = { AVID_T, FPL_T, INL_T, (}

<additive arithmetic expression'> ->
+ <multiplicative arithmetic expression><additive arithmetic expression'>
| - <multiplicative arithmetic expression><additive arithmetic expression'>
| ∈
FIRST (<additive arithmetic expression'>) = { +, -, ∈}

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

Fixed Left Recursion
<multiplicative arithmetic expression> ->
<primary arithmetic expression><multiplicative arithmetic expression'>
FIRST (<multiplicative arithmetic expression>) = { AVID_T, FPL_T, INL_T, (}

<multiplicative arithmetic expression'> ->
* <primary arithmetic expression><multiplicative arithmetic expression'>
| / <primary arithmetic expression><multiplicative arithmetic expression'>
| ∈
FIRST (<multiplicative arithmetic expression'>) = { *, / , ∈}

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

3.3.2 String Expression

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

Fixed Left Recursion
<string expression> ->
<string expression> # <primary string expression> | <primary string expression>

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

<string expression'> -> # <primary string expression> <string expression'> | ∈
FIRST (<string expression'>) = { #, ∈}

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

3.3.3 Conditional Expression

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

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

Fixed Left Recursion
<logical OR expression> ->
<logical OR expression> .OR. <logical AND expression>
| <logical AND expression>

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

<logical OR expression'> -> .OR. <logical AND expression><logical OR expression'> | ∈
FIRST (<logical OR expression'>) = { .OR. , ∈ }

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

Fixed Left Recursion
<logical AND expression> ->
<logical AND expression> .AND. <relational expression>
| <relational expression>
<logical AND expression> -> <relational expression> <logical AND expression'>
FIRST (<logical AND expression>) = { AVID_T, FPL_T, INL_T, SVID_T, STR_T }

<logical AND expression'> -> .AND. <relational expression><logical AND expression'> | ∈
FIRST (<logical AND expression'>) = {.AND. , ∈ }

Kevin Lai 040812704
Xingyi Wu 040887028

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>

Applying Left Factoring
<relational expression> ->
<primary a_relational expression> <primary a_relational expression'>
| <primary s_relational expression><primary s_relational expression'>
FIRST(<relational expression>) = { AVID_T, FPL_T, INL_T, SVID_T, STR_T }

<primary a_relational expression'> ->
== <primary a_relational expression>
| <> <primary a_relational expression>
| > <primary a_relational expression>
| < <primary a_relational expression>
FIRST(<primary a_relational expression'>) = { ==, <>, >, < }

<primary s_relational expression'> ->
== <primary s_relational expression>
| <> <primary s_relational expression>
| > <primary s_relational expression>
| < <primary s_relational expression>
FIRST(<primary s_relational expression'>) = { ==, <>, >, < }

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

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