Revised Platypus Grammar

James Holmes and Adam White

3.1 Platypus Program

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

<program> -> PLATYPUS {<opt\_statements>}

First(<program>) = {PLATYPUS}

<opt\_statements> -> <statements> | ɛ

First(<opt\_statements>) = {AVID\_T, SVID\_T, IF, WHILE, READ, WRITE, ɛ}

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

MODIFICATION: removed left recursion

<statements> -> <statement><statements'>

First(<statements>) = {AVID\_T, SVID\_T, IF, WHILE, READ, WRITE}

<statements'> -> <statement><statements'> | ɛ

First(<statements'>) = {AVID\_T, SVID\_T, IF, WHILE, READ, WRITE, ɛ}

3.2 Statements

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

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

First(<statement>) = {AVID\_T, SVID\_T, IF, WHILE, READ, WRITE}

3.21 Assignment Statement

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

<assignment statement> -> <assignment expression>;

First(<assignment statement>) = {AVID\_T, SVID\_T}

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

First(<assignment expression>) = {AVID\_T, SVID\_T}

3.22 Selection Statement

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

<selection statement> -> IF TRUE (<conditional expression>) THEN {<opt\_statements>} ELSE {opt\_statements>};

First(<selection statement>) = {IF}

3.23 Iteration Statement

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

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

First(<iteration statement>) = {WHILE}

<pre-condition> -> TRUE | FALSE

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

3.24 Input Statement

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

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

First(<input statement>) = {READ}

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

MODIFICATION: removed left recursion

<variable list> -> <variable identifier><variable list'>

First(<variable list>) = {SVID\_T, AVID\_T}

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

First(<variable list'>) = {SVID\_T, AVID\_T, ɛ}

<variable identifier> -> AVID\_T | SVID\_T

First(<variable identifier>) = {SVID\_T, AVID\_T}

3.25 Output Statement

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

<output statement> ->WRITE (<opt\_variable list>); | WRITE (STR\_T);

MODIFICATION: left factored

<output statement> -> WRITE(<output list>);

First(<output statement>) = {WRITE}

<output list> -> <opt\_variable list> | STR\_T

First(<output list>) = {SVID\_T, AVID\_T, STR\_T, ɛ}

<opt\_variable list> -> <variable list> | ɛ

First(<opt\_variable list>) = {SVID\_T, AVID\_T, ɛ}

3.31 Arithmetic Expressions

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

<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>

MODIFICATION: left factored

<unary arithmetic expression> -> <unary operator><primary arithmetic expression>

First(<unary arithmetic expression>) = {+,-}

<unary operator> -> + | -

First(<unary operator>) = {+,-}

<additive arithmetic expression> ->

<additive arithmetic expression> + <multiplicative arithmetic expression>

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

| <multiplicative arithmetic expression>

MODIFICATION: removed left recursion, applied left factoring via <additive arithmetic expression'>

<additive arithmetic expression> -> <multiplicative arithmetic expression><additive arithmetic expression'>

First(<additive arithmetic expression>) = {AVID\_T, FPL\_T, INL\_T, (}

<additive arithmetic expression'> -> <additive arithmetic expression operator><multiplicative arithmetic expression><additive arithmetic expression'> | ɛ

First(<additive arithmetic expression'>) = {+, -, ɛ}

<additive arithmetic expression operator> -> + | -

First(<additive arithmetic expression operator>) = {+, -}

<multiplicative arithmetic expression> ->

<multiplicative arithmetic expression> \* <primary arithmetic expression>

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

| <primary arithmetic expression>

MODIFICATION: removed left recursion, applied left factoring via <multiplicative arithmetic expression'>

<multiplicative arithmetic expression> -> <primary arithmetic expression><multiplicative arithmetic expression'>

First(<multiplicative arithmetic expression>) = {AVID\_T, FPL\_T, INL\_T, (}

<multiplicative arithmetic expression'> -> <multiplicative arithmetic expression operator><primary arithmetic expression><multiplicative arithmetic expression'> | ɛ

First(<multiplicative arithmetic expression'>) = {\*, /, ɛ}

<multiplicative arithmetic expression operator> -> \* | /

First(<multiplicative arithmetic expression operator>) = {\*, /}

<primary arithmetic expression> -> AVID\_T | FPL\_T | INL\_T | (<arithmetic expression>)

First(<primary arithmetic expression>) = {AVID\_T, FPL\_T, INL\_T, (}

3.32 String Expressions

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

<string expression> ->

<primary string expression>

| <string expression> # <primary string expression>

MODIFICATION: removed left recursion

<string expression> -> <primary string expression><string expression'>

First(<string expression>) = {SVID\_T, STR\_T}

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

First(<string expression'>) = {#, ɛ}

<primary string expression> -> SVID\_T | STR\_T

First(<primary string expression>) = {SVID\_T, STR\_T}

3.33 Conditional Expressions

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

<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>

MODIFICATION: removed left recursion

<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'> | ɛ

First(<logical OR expression'>) = {.OR., ɛ}

<logical AND expression> ->

<relational expression>

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

MODIFICATION: removed left recursion

<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'> | ɛ

First(<logical AND expression'>) = {.AND., ɛ}

3.34 Relational Expressions

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

<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>

Modification: “Middle factored”, so to speak. Definitely not left factored!

<relational expression> -> <primary a\_relational expression><relational operator><primary a\_relational expression>

| <primary s\_relational expression><relational operator><primary s\_relational expression>

First(<relational expression>) = {AVID\_T, FPL\_T, INL\_T, SVID\_T, STR\_T}

<relational operator> -> == | <> | < | >

First(<relational operator>) = {==, <>, <, >}

<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}