**3 The PLATYPUS Syntactic Specification**

**3.1 PLATYPUS Program**

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

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

***Grammar was transformed due to left recursion***

|  |
| --- |
| **<program> ->**PLATYPUS {<opt\_statements>}  FIRST(program) = {KW\_T(PLATYPUS) } |
| **<opt\_statements>->** <statements> | E  FIRST(opt\_statements) = { FIRST(statements), E } = { AVID\_T, SVID\_T, KW\_T(IF), KW\_T(WHILE), KW\_T(READ), KW\_T(WRITE), E} |
| **<statements>->** <statement><statements’>  FIRST(statements) = { FIRST(statement) } = { AVID\_T, SVID\_T, KW\_T(IF), KW\_T(WHILE), KW\_T(READ), KW\_T(WRITE) } |
| **<statements’>->** <statement><statements’>|E  FIRST(statements’) = { FIRST(statement), E } = { AVID\_T, SVID\_T, KW\_T(IF), KW\_T(WHILE), KW\_T(READ), KW\_T(WRITE), E } |

**3.2 Statements**

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

|  |
| --- |
| **<statement> ->**<assignment statement>| <selection statement>| <iteration statement>| <input statement>| <output statement>  FIRST(statement) = { FIRST(assignment statement), FIRST(selection statement), FIRST(iteration statement), FIRST(input statement), FIRST(output 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>;

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

|  |
| --- |
| **<assignment statement> ->**<assignment expression>;  FIRST(assignment statement) = { FIRST(assignment expression) } = { AVID\_T, SVID\_T } |
| **< assignment expression> ->**AVID = <arithmetic expression>| SVID = <string expression>  FIRST(assignment expression) = { AVID\_T, SVID\_T } |

CST8152 – Compilers, A2, F18 Page 8 of 12

**3.2.2 Selection Statement( the if statement)**

<selection statement> ->IF <pre-condition> (<conditional expression>) THEN { <opt\_statements> }

ELSE { <opt\_statements> } ;

|  |
| --- |
| **<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>**};**

**<**pre-condition> ->TRUE | FALSE

|  |
| --- |
| **<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) } |

**3.2.4 Input Statement**

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

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

|  |
| --- |
| **<input statement> ->**READ (<variable list>);  FIRST(input statement) = { KW\_T(READ) } |
| **<variable list> ->** <variable identifier><variable list’>  FIRST(variable list) = { FIRST(variable identifier)} ={ SVID\_T, AVID\_T } |
| **<variable identifier> ->** SVID\_T | AVID\_T  FIRST(variable identifier) = { SVID\_T, AVID\_T } |
| **<variable list’> ->** ,<variable list> | E  FIRST(variable list’) = { , , E } |

CST8152 – Compilers, A2, F18 Page 9 of 12

**3.2.5 Output Statement**

<output statement> ->WRITE (<opt\_variable list>);| WRITE (<string literal>);

***Left Factoring non-predictive***

|  |
| --- |
| **<output statement> ->** WRITE(<output list>);  FIRST(output statement) = { KW\_T(WRITE) } |
| **<output list> ->** <variable list> | STR\_T | E  FIRST(output list) = { SVID\_T, AVID\_T, STR\_T, E } |

**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> ->** <arithmetic variable identifier>| <floating-point literal>| <integer literal>

| (<arithmetic expression>)

***Transforming due to left recursive***

|  |
| --- |
| **<arithmetic expression> - >**<unary arithmetic expression>| <additive arithmetic expression>  FIRST(arithmetic expression) = { FIRST(unary arithmetic expression), FIRST(additive arithmetic expression) } = { AVID\_T, FPL\_T, INL\_T, (,+, - } |
| **<unary arithmetic expression> ->**- <primary arithmetic expression>| + <primary arithmetic expression>  FIRST(unary arithmetic expression) = { -, + } |
| **<additive arithmetic expression> ->** <multiplicative arithmetic expression><additive arithmetic expression’>  FIRST(additive arithmetic expression) = { FIRST(multiplicative arithmetic expression)} = { AVID\_T, FPL\_T, INL\_T, ( } |
| **<additive arithmetic expression’>** -> +<multiplicative arithmetic expression><additive arithmetic expression’>|-<multiplicative arithmetic expression><additive arithmetic expression’>|E  FIRST(additive arithmetic expression’) = { +, -, E } |
| **<multiplicative arithmetic expression> ->** <primary arithmetic expression> <multiplicative arithmetic expression’>  FIRST(multiplicative arithmetic expression) = { FIRST(primary arithmetic expression)} = { AVID\_T, FPL\_T, INL\_T, ( } |
| **<multiplicative arithmetic expression’> ->** \*<primary arithmetic expression><multiplicative arithmetic expression’> | /<primary arithmetic expression><multiplicative arithmetic expression’>| E  FIRST(multiplicative arithmetic expression’) = { \*, /, E } |
| **<primary arithmetic expression> ->** <arithmetic variable identifier>| <floating-point literal>| <integer literal>  | (<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>

<primary string expression> -><string variable identifier>| <string literal>

***Transformation due to left recursive***

|  |
| --- |
| **<string expression> ->** <primary string expression><string expression’>  FIRST(string expression) -> { FIRST(primary string expression) } = { SVID\_T, STR\_T } |
| **<string expression’> -> #**<primary string expression><string expression’>|E  FIRST(string expression’) -> { #, SVID\_T, STR\_T, E } |
| **<primary string expression> ->**<string variable identifier>| <string literal>  FIRST(primary string expression) = { SVID\_T, STR\_T } |
| **<string literal> ->** STR\_T  FIRST(string literal) = { STR\_T } |

**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>

***Transformation due to left recursive of both <logical OR expression> and <logical AND expression>***

|  |
| --- |
| **<conditional expression>** **->**<logical OR expression>  FIRST(conditional expression) = { FIRST(logical OR expression) } = { AVID\_T, FPL\_T, INL\_T, SVID\_T, STR\_T } |
| **<logical OR expression>** **->** <logical AND expression><logical OR expression’>  FIRST(logical OR expression) = { FIRST(logical AND expression) } = { AVID\_T, FPL\_T, INL\_T, SVID\_T, STR\_T } |
| **<logical OR expression’>** **->** .OR.<logical AND expression><logical OR expression’>|E  FIRST(logical OR expression’) = { LOG\_OP\_T(OR), E } |
| **<logical AND expression>** **->**<relational expression><logical AND expression’>  FIRST(logical AND expression) = { FIRST(relational expression) } = { AVID\_T, FPL\_T, INL\_T, SVID\_T, STR\_T } |
| **<logical AND expression’> ->.**AND.<relational expression><logical AND expression’>|E  FIRST(logical AND expression’) = { LOG\_OP\_T(AND), E } |

**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> ->

<floating-point literal>| <integer literal>| <arithmetic variable identifier>

<primary s\_relational expression> -><primary string expression>

***Reworking Left factoring non-predictive grammar until production does not contain a left-factor anymore.***

|  |
| --- |
| **<relational expression> ->** <primary a\_relational expression> < primary a\_relational expression’>| <primary s\_relational expression> < primary s\_relational expression’>  FIRST (relational expression) = {FIRST (primary a\_relational expression), FRIST (primary s\_relational expression)} = {AVID\_T, FPL\_T, INL\_T, SVID\_T, STR\_T} |
| **<primary a\_relational expression> ->** <floating-point literal>| <integer literal>| <arithmetic variable identifier>  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} |
| **<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’) = {REL\_OP\_T(EQ), REL\_OP\_T (NE), REL\_OP\_T (GT), REL\_OP\_T (LT)} |
| **<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’) = {REL\_OP\_T(EQ), REL\_OP\_T (NE), REL\_OP\_T (GT), REL\_OP\_T (LT)} |