**syntax-directed translation**

synthesized attributes:

Var TYPE.datatype;

DEFINITION\_B.lexeme

Var EXPRESSION.datatype;

Inherited attributes:

Var DEFINITION.datatype;

Var DEFINITION\_B.datatype;

Var EXPRESSION\_B.datatype;

Var ID\_LIST.datatype;

1. PROGRAM 🡪 **start {make\_table()}** DEFINITIONS **{** COMMANDS **}** **finish { pop\_table ();}**

//although it doesn’t specified in the requirements, this rule should have a make\_table() and pop\_table(), since it’s have definitions and commands in it.

1. DEFINITIONS 🡪 DEFINITION DEFINITIONS\_B
2. DEFINITIONS\_B 🡪 **;** DEFINITIONS
3. DEFINITIONS\_B 🡪 Ɛ
4. DEFINITION 🡪 id **{ id\_entry = insert ( id\_name); if(id\_entry == NULL){** **DUPLICATED DECLARATION ERROR}} :** **exception {** **set\_id\_type (id\_entry, ‘exception’)}**
5. DEFINITION 🡪 TYPE id **{ id\_entry = insert (current\_table, id\_name);**

**if(id\_entry == NULL){ DUPLICATED DECLARATION ERROR };**

**DEFINITION\_B.datatype = TYPE.datatype; DEFINITION\_B.lexeme = id.lexeme}** DEFINITION\_B

1. DEFINITION\_B 🡪 **{ ID\_LIST.datatype = DEFINITION\_B.datatype }** ID\_LIST
2. DEFINITION\_B 🡪 **const {id\_entry = lookup(cur\_table\_node->hash\_table, DEFINITION\_B.lexeme); set\_as\_const(id\_entry)} =** number
3. TYPE 🡪 **integer {TYPE.datatype = ‘integer’}**
4. TYPE 🡪 **real {TYPE.datatype = ‘real’}**
5. ID\_LIST 🡪 Ɛ
6. ID\_LIST 🡪 **,** id **{id\_entry = insert ( id\_name);**

**if(id\_entry == NULL){ DUPLICATED DECLARATION ERROR }**

**else{** **set\_id\_type(id\_entry, ID\_LIST.datatype) };**

**ID\_LIST\_R.datatype = ID\_LIST\_L.datatype }** ID\_LIST

1. COMMANDS 🡪 COMMAND COMMANDS\_B
2. COMMANDS\_B 🡪 Ɛ
3. COMMANDS\_B 🡪**;** COMMANDS
4. COMMAND 🡪 id **= { id\_entry = lookup(id.lexeme);**

**If ( id\_entry == NULL) { VARIABLE\_NOT\_DECLARED\_ERROR};**

**If ( is\_const(id\_entry)) { ASSIGNMENT\_TO\_CONSTANT\_ERROR};**

**If ( is\_exception(id\_entry) ){** **EXCEPTION NAME REFERENCED ERROR}; }**

EXPRESSION

**{ If ( is\_integer(id\_entry) && EXPRESSION.datatype == real ){** **TYPE CONSISTENCY ERROR};**

1. COMMAND 🡪 **while** CONDITION **do** COMMANDS **od**
2. COMMAND 🡪 **raise** id **{ id\_entry = find(id.lexeme);**

**if(id\_entry == null){ VARIABLE\_NOT\_DECLARED\_ERROR}**

**else(!is\_exception(id\_entry){** **NON-EXCEPTION IN RAISE ERROR} }**

1. COMMAND 🡪 **block** **{make\_table()}** DEFINITIONS **{** COMMANDS **}** **end\_block { pop\_table ();}**
2. EXPRESSION 🡪 number **{EXPRESSION.datatype = number. datatype}**
3. EXPRESSION 🡪 id **{ id\_entry = find(id\_name);**

**if(id\_entry == NULL){**

**EXPRESSION.datatype = error\_type; VARIABLE NOT DECLARED ERROR} }**

EXPRESSION\_B

**{ if (EXPRESSION\_B.datatype == error\_type){**

**EXPRESSION.datatype = error\_type;}**

**elseif (is\_integer(id\_entery) && EXPRESSION\_B.datatype == integer) {**

**EXPRESSION.datatype = integer; }**

**else{**

**EXPRESSION.datatype = real; }**

1. EXPRESSION\_B 🡪 Ɛ **{ EXPRESSION\_B.datatype = integer }**
2. EXPRESSION\_B 🡪 binary\_ar\_op EXPRESSION **{ EXPRESSION\_B.datatype = EXPRESSION.datatype; }**
3. CONDITION 🡪 **(** id **{ id\_entry = find(id\_name);**

**if(id\_entry == NULL ) {VARIABLE NOT DECLARED ERROR}**

**elseif(is\_exception(id\_entry)){** **EXCEPTION NAME REFERENCED ERROR}**

rel\_op id

**{ id\_entry = find(id\_name);**

**if(id\_entry == NULL ) {VARIABLE NOT DECLARED ERROR}**

**elseif(is\_exception(id\_entry)){** **EXCEPTION NAME REFERENCED ERROR}**