1 Definiciones Dirigidas y su Esquema de traducción

1.1 Declaración de Variables

1.1.1 Definición Dirigida por la Sintaxis

REGLAS DE PRODUCCIÓN	REGLAS SEMÁNTICAS
D → T L ;	L.tipo = T.tipo L.dim = T.dim
T → **int**	T.tipo = int T.dim = 2
L → L , **id** C	L1.tipo = L.tipo L1.dim = L.dim C.tipo = L.tipo C.dim = L.dim C.id = id.lexval
L → **id** C	C.tipo = C.tipo C.dim = L.dim C.id = id.lexval

C → ε	**if**(!simbolos.existe(C.id)) **then**
	tipo = tipos.insert(entero.lexval, C.tipo)
	simbolos.insert(id, "var", tipo, dir)
	dir = dir + C.dim
	else
	error("el id ya existe");
	endif

C → [**entero**]	**if**(!simbolos.existe(C.id))**then**
	tipo = tipos.insert(entero.lexval, C.tipo)
	simbolos.insert(id, "var", tipo, dir)
	dir = dir + C.dim*entero.lexval
	else
	error("el id ya existe");
	endif

1.1.2 Esquema de Traducción Dirigido por la Sintaxis

- D → T{

  L.tipo = T.tipo; L.dim = T.dim;

  } L ;

- T → **int** {

  T.tipo = int; T.dim = 2;

  }

- L →{

  L1.tipo = L.tipo; L1.dim = L.dim;

  }L , **id** {

  C.tipo = L.tipo; C.dim = L.dim; C.id = id.lexval;

  } C

- L → **id** {

  C.tipo = L.tipo; C.dim = L.dim; C.id = id.lexval;

  } C

- C → ε {

```
if(!simbolo.existe(C.id)) then
   tipo = tipos.getTipo(C.tipo);
   simbolos.insert(id, "var", tipo , dir);
   dir += C.dim;
else
   error("el id ya existe")
endif
```

}

- C →[**entero**] {

```
if(!simbolo.existe(C.id)) then
   tipo = tipos.insert(entero.lexval, C.tipo);
   simbolos.insert(id, "var", tipo , dir);
   dir += C.dim*entero.lexval;
else
   error("el id ya existe");
endif
```

}

1.1.3 Esquema de Traducción en Yacc

```
%{
import java.io.*;
%}

%token <sval>ID
%token INT
%token COMA PYC
%token LCOR RCOR
%token <ival> ENTERO

%type <obj> list
%type <obj> type

%start decl

%%

decl : type{ currentType = (Tipo)$1;}
  list PYC;

type : INT { $$ = new ParserVal(new Tipo("int",2);};

list: list {$1 = $$;} COMA
    ID {currentId = $4;} comp
  | ID {currentId = $1;} comp;

comp : LCOR ENTERO RCOR
  {if(!simbolos.existe(currentId)){
    int tipo = tipos.insert($2,currentType.getType());
    simbolos.insert(currentID, "var", tipo, dir);
    dir += currentType.getDim()*$2;
  }else{
    error("el id esta duplicado");
  }
}
  | {if(!simbolos.existe(currentId)){
    int tipo = tipos.getType(currentType.getType());
    simbolos.insert(currentID, "var", tipo, dir);
    dir += currentType.getDim();
  }else{
    error("el id esta duplicado");
  }
};



%%

Tipo curretType;
String currentId;
int dir = 0;
Tabla simbolos;
Tabla tipos;
```

1.2 Declaración de Funciones

1.2.1 Definición Dirigida por la Sintaxis

REGLAS DE PRODUCCIÓN	REGLAS SEMÁNTICAS

F → T **id(** P) B	`if(!simbolos.existe(id)) then`
	`  dirLocal = 0;`
	`  Label L = new LabelId();`
	`  dir = dir + T.dim`
	`  if(tipos.getType(T.tipo)!="void") then`
	`  retorno = true;`
	`  else`
	`  retorno = false;`
	`  endif`
	`  if(retorno != B.retorno = false )then`
	`  error("La funcion requiere una sentencia return")`
	`  endif`
	`  F.cod = genCod(label, L) ||= P.codigo ||= B.codigo`
	`  simbolos.pop();`
	`  simbolos.insert(id, "func", T.tipo, dir,P.num))`
	`else`
	`  error("id ya definido")`
	`endif`

P → Y	`P.num = Y.num` `P.params = Y.params`
P → **void**	`P.num = 0;` `P.params = new Params();`
Y → Y , M	`Y.num = Y1.num + 1;` `Y.params = Y1.params.insert(M.param);`

Y → M	`Y.num = 1`
	`Y.params = new List()`
	`Y.params.insert(M.param)`

M → T **id** N	`N.tipo = T.tipo`
	`N.dim = T.dim`
	`N.id = id.lexval`
	`M.param = new Param(id.lexval, N.tipo)`

N → [ ]	`if(!simbolos.existe(N.id)) then`
	`  int tipo = tipos.insert(N.tipo,"array")`
	`  simbolos.insert(N.id, "param", tipo, dirLocal)`
	`  dirLocal = dirLocal + N.dim`
	`  N.tipo = tipo`
	`else`
	`  error("La variable ya fue declarada en este ambito");`
	`endif`

N → ε	`if(!simbolos.existe(N.id)) then`
	`  int tipo = tipos.getTipo(N.tipo)`
	`  simbolos.insert(N.id, "param", tipo, dirLocal)`
	`  dirLocal = dirLocal + N.dim`
	`  N.tipo = tipo`
	`else`
	`  error("La variable ya fue declarada en este ambito");`
	`endif`

1.2.2 Esquema de Traducción

- F → T{

```
if(T.tipo == "void") retorno = false else retorno = true endif
```

} **id**({

```
if(!simbolos.existe(id)) then
   simbolos.push(new Tabla());
   dirLocal = 0;
   Label L = new LabelId();
else
  error("id duplicado)
endif
```

} P) B {

```
if(retorno == B.retorno) then
  F.codigo = genCod("label", L) || B.codigo
  simbolos.pop();
  simbolos.insert(id, "func", T.tipo, dir,P.num))
else
  error( "La funcion debe retornar algun valor")
endif
```

}

- P → Y {

```
P.num = Y.num;
P.params = Y.params;
```

  }

- P → **void** {

```
P.num = 0;
P.params = new List();
```

  }

- Y → Y, M {

```
Y.num = Y1.num +1;
Y.params = Y1.params.insert(M.param);
```

  }

- Y → M {

```
Y.num = 1;
Y.params = new List();
Y.params.insert(M.param)
```

  }

- M → T **id** {

```
N.tipo = T.tipo;
N.dim = T.dim;
N.id = id.lexval;
M.param = new Param(id.lexval, N.tipo)
```

  }N

- N → [ ] {

```
if(!simbolos.existe(N.id)) then
   int tipo = tipos.insert(N.tipo,"array")
   simbolos.insert(N.id, "param", tipo, dirLocal)
   dirLocal = dirLocal + N.dim
   N.tipo = tipo
else
  error("La variable ya fue declarada en este ambito");
endif
```

  }

- N → ε {

```
if(!simbolos.existe(N.id)) then
   int tipo = tipos.getTipo(N.tipo)
   simbolos.insert(N.id, "param", tipo, dirLocal)
   dirLocal = dirLocal + N.dim
   N.tipo = tipo
else
  error("La variable ya fue declarada en este ambito");
endif
```

  }

1.2.3 Esquema de Traducción en yacc

```
%{
import java.io.*;
%}

%token LPAR RPAR
%token <sval> ID
%token INT VOID
%token LCOR RCOR
%token COMA
%token RETURN

%type <obj> type
%type <obj> params param
%type <obj> bloque
%type <sval> funcion
%type <obj> var
%type <ival> complemento

%start funcion

%%
funcion : type {currentType = $1;} ID
  { if(!simbolos.existe($3)){
    simbolos.push(new Tabla());
    dirLocal =0;
    Label L = new Label($3);
    if(((Tipo)$1).getType.equals("void"))
      retorno = false;
    else
      retorno = true;
   }else{
    error("ID duplicado");
  }} LPAR params RPAR
  bloque{
  if(retorno != ((Retorno)$8).retorno){
    error("Se requiere retornar algun valor");
  }
  $$ =  genCod("label", L) + ((Retorno)$8).codigo;
  simbolos.pop()
  simbolos.insert($3, "func", currenType, dir, P.num, P.params);
  };

type : INT { $$ = new Tipo("int",2);}
  | VOID {$$ = new Tipo("void",0);};

params : param { ((Params)$$).num = (Params)$1).num;
    (Params)$$).list = (Params)$1).list }
  | VOID { (Params)$$).num=0;
    (Params)$$).list = new List();};

param : param COMA var{(Params)$$).num= (Params)$1).num+ 1;
    (Params)$$).list = (Params)$1).list.insert($3);}
  | var { (Params)$$).num=  1;
      (Params)$$).list = new List();
      (Params)$$).list.insert($1);};

var : type{ currentType = $1;}
  ID
  { currenType = $1;
    currentId = $3;
  }
  complemento {$$ = new Param( $3, $5)}
  ;

complemento : LCOR RCOR
  { if(!simbolos.existe(currentId)){
    int tipo = tipos.insert(currentType.getTipo(),"array");
    simbolos.insert(currentId, "param", tipo, dirLocal);
    dirLocal += tipos.getDim(tipo);
    $$ = tipo;
  }else
    error("id duplicado");
  }
  | { if(!simbolos.existe(currentId)){
    int tipo = tipos.insert(currentType.getTipo(),"array");
    simbolos.insert(currentId, "param", tipo, dirLocal);
    dirLocal += tipos.getDim(tipo);
    $$= tipo;
  }else
    error("id duplicado");
  };


bloque: RETURN{ $$.obj = new Retorno(true); }| {$$.obj = new Retorno(false);};
%%

Pila<Tabla> simbolos;
Pila<Tabla> tipos;
Tipo currentType;
String currentId;
int dirLocal;
boolean retorno;
```

1.3 Llamadas a procedimientos

REGLAS DE PRODUCCIÓN	REGLAS SEMÁNTICAS

C → **id**(A)	`if( simbolos.existe(id)) then`
	`  if(A.num == simbolos.getNum(id)) then`
	`  if(A.num != 0) then`
	`  Lista = simbolos.getArg(id,"param")`
	`  for(i=0 hasta A.num) do`
	`  if(A.args.get(i)<>Lista.get(i)) then`
	`  error("El argumento es de tipo diferente")`
	`  endif`
	`  endfor`
	`  endif`
	`  else`
	`  error("El No. de argumentos no coincide")`
	`  endif`
	`else`
	`  error("La funcion no ha sido declarada")`
	`endif`
	`C.tipo = simbolos.getTipo(id)`
	`C.cod = A.cod || "call " || id.lexval || A.num`

A → Z	`A.args = Z.args`
	`A.num = Z.num`
	`A.cod = Z.cod || Z.cod2`

A → ε	`A.args = new List()` `A.num = 0`

$Z \rightarrow Z$,E	`Z1.args.insert(E)`
	`Z.args= Z1.args`
	`Z.num = Z1.num + 1`
	`Z.cod = Z1.cod || E.cod`
	`Z.cod2 = Z1.cod2 || "param" || E.dir || "," || Z.num`

$Z \rightarrow E$	`Z.args = new List()`
	`Z.args.insert(E)`
	`Z.num = 0;`
	`Z.cod = E.cod`
	`Z.cod2 = "param" || E.dir || "," || Z.num`

1.3.1 Equema de traducción

- C → **id**({

```
if(!simbolos.existe(id))then
  error("La funcion no ha sido declarada")
endif
```

} A) {

```
if(A.num == simobolos.getNum(id)) then
   if(A.num !=0) then
      lista = simbolos.getArgs(id)
      for(i=0 hasta A.num) do
         if(A.args.get(i)!= lista.get(i)) then
            error("el argumento es de tipo diferente")
         endif
      endfor
   endif
else
  error("El No. de argumentos no coincide")
endif
C.tipo = simbolos.getTipo(id)
C.cod = A.cod || "call " || id.lexval || A.num
```

}

- A → Z {

```
A.agrs = Z.args
A.num = Z.num
A.cod = Z.cod || Z.cod2
```

}

- $A \rightarrow \varepsilon$ {

```
A.args = new List()
A.num = 0
```

}

- $Z \rightarrow Z$, E {

```
Z.args = Z1.args.insert(E)
Z.num = Z1.num +1
Z.cod = Z1.cod || E.cod
Z.cod2 = Z1.cod2 || "param" || E.dir || "," || Z.num
```

}

- $Z \rightarrow E$ {

```
Z.args = new list()
Z.args.insert(E)
Z.num =0
Z.cod = E.cod
Z.cod2 = "param" || E.dir || "," || Z.num
```

}

1.3.2 Parser en yacc, Scanner en lex

En este apartado se presenta un ejemplo completo de como trabajar con lex y yacc en su versión para java, también el código de algunas de las clases que son necesarias para que funcione. Las clases Lista, Pila, Tabla y TablaT las tendra que implementar el lector como ejercicio para compildar el proyecto completo en java. Como nota podemos decir que al esquema de traducción se le agregaron unas producciones más para tenerlo completo.

1.3.3 Archivo en yacc

```
%{
package funciones;
import java.io.*;

%}



%token <sval> ID
%token LPAR RPAR
%token COMA
%token <obj> NUM

%left MAS

%type <obj> args arg
%type <obj> exp term
%type <obj> call


%start call

%%

// call --> id (args)
call : {init();} ID LPAR{
  if(!simbolos.existe($2))
    yyerror("la funcion no ha sido declarada");
  }
  args RPAR
  {
   if(((Args)$4).num == simbolos.getNum(id)){
    if(((Args)$4).num !=0){
      Lista list = simbolos.getArgs(id);
      for(int i =0; i<list.size(); i++){
        if(((Args)$4).args.get(i).equals(list.get(i)))
          yyerror("El argumento es de tipo distinto");
      }
    }
  }else
    yyerror("El numero de argumentos no coincide");
```

```
  Call call = new Call();
  call.codigo = ((Args)$4).codigo + "call" + $1 + ((Args)$4).num;
  call.tipo = simbolo.getTipo($1);
};


args :  // args --> arg
  arg {
  ((Args)$$).args = ((Args)$1).args;
  ((Args)$$).num = ((Args)$1).num;
  ((Args)$$).codigo = ((Args)$1).codigo + ((Args)$1).codigo2;
  }
  // args --> epsilon
  | {
          ((Args)$$).args = new Lista();
    ((Args)$$).num = 0;
  };


arg : // arg --> arg , exp
  arg COMA exp {
  ((Args)$1).args.insert($3);
  ((Args)$1).num += 1;
  $$ = $1;
  ((Args)$$).codigo += $3.codigo;
  ((Args)$1).codigo2 += "param " + ((exp)$1).dir + ", " + ((Args)$$).num;
  }
  |// arg --> exp
  exp
  {
  ((Args)$$).args = new Lista();
  ((Args)$$).args.insert($1);
  ((Args)$$).num =0;
  ((Args)$$).codigo += $1.codigo;
  ((Args)$1).codigo2 += "param " + ((exp)$1).dir + ", " + ((Args)$$).num;
  };

exp : //exp--> exp + term
  exp MAS term{
  if(((Expresion)$1).type.equals(((Expresion)$3).type)){
    Expresion exp = new Expresion();
    exp.type = ((Expresion)$1).type;
    exp.dim = tipos.getDim(exp.type);
    exp.dir = temporal;
    temporal += exp.dim;
    exp.codigo = ((Expresion)$1).codigo + ((Expresion)$3).codigo;
    exp.codigo += exp.dir + "=" + ((Expresion)$1).dir + "+" + ((Expresion)$3).dir;
    $$ = exp;
  }else
    error("incopatibilidad de tipos");
  }
  | //exp --> term
  term
  { $$ = $1;};

term : // term --> id
  ID{
  Expresion term = new Expresion();
  term.type = simbolos.getType($1);
  term.dim = tipos.getDim(term.type);
  term.dir = simbolos.getDir($1);
  term.codigo="";
  $$ = term;
  }
  | //term -->num
  NUM{
  Expresion term = new Expresion();
  term.type = ((Expresion) $1).type;
  term.dim = tipos.getDim(term.type);
  term.dir = simbolos.getDir($1);
  term.codigo = term.dir"=" ((Expresion)$1).valor;
  $$ = term;
};
```

%%

```
int temporal;
Pila<Tabla> simbolos;
Pila<TablaT> tipos;

private void init(){
  simbolos = new Pila();
  tipos = new Pila();
  simbolos.push(new Tabla());
  tipos.push(new TablaT());
}

private int yylex(){
  int yyl_return = -1;
  try{
    yyl_return = lexer.yylex();
  }catch(IOException e){
    System.err.println("IO  error: " +e);
  }
  return yyl_return;
```

```
}

/* Error reporting */
public void yyerror(String error){
  System.err.println("Error: "+ error);
}


/* Lexer is created in the constructor */
public Parser(Reader r){
  lexer = new Yylex(r, this);
}


public static void mian(String args[]){
  Parser yyparser = new Parser(new FileReader(args[0]));
  //Metodo que realiza el proceso de analisis
  yyparser.yyparse();
}
```

1.3.4 Archivo de lex

```
package funciones;
import java.io.*;

%%

%byaccj

%line
%char

%{
private Parser yyparser;

public Yylex(Reader r, Parser yyparser){
  this(r);
  this.yyparser = yyparser;
}
%}


ID = [a-zA-Z_][a-zA-Z0-9_]*
ENT = [0-9]+
REAL = [0-9]+("."[0-9]+)?(E[+-][0-9]+)?

%%

{ID} { yyparser.yylval = new ParserVal(yytext());
  return Parser.ID;}

{ENT} {Expresion ent = new Expresion();
   ent.type = 0;
   ent.valor = Integer.parseInt(yytext());
   yyparser.yylval = new ParserVal(ent);
   return Parser.NUM;}

{REAL}  {Expresion real = new Expresion();
   real.type =1;
   real.valor = Float.parseFloat(yytext());
   yyparser.yylval = new ParserVal(real);
   return Parser.NUM;}

[ \t\n\r]+  {}

"," {return Parser.COMA;}
"(" {return Parser.LPAR;}
")" {return Parser.RPAR;}
"+" {return Parser.MAS;}
. {yyparser.yyerror("erro lexico en: linea "+ yyline +" columna " + yycolum);}
```

1.3.5 Código de java complementarios

```
package fuciones;

public class Expresion{
  int type;
  int ivalor;
  float fvalor;
  int dir;
  int dim;
  String codigo;
}
```

```
package funciones;

public class Call{
  String codigo;
  int type;
}

package funciones;

public class Args{
  String codigo;
  String codigo2;
  int num;
  Lista args;
}
```