Proyecto Final

Analizador Léxico, Sintáctico y Semántico con Generación de Código Intermedio

Compiladores

Grupo 7006

Facultad de Ciencias, UNAM

Profesor:
Adrián Ulises Mercado

Alumno:
Gutiérrez Velázquez Héctor Ernesto

GRAMÁTICA INICIAL

programa → declaraciones funciones
declaraciones → tipo lista_var **;** declaraciones | ε
tipo → basico compuesto
basico → **int** | **float** | **char** | **double** | **void**
compuesto → **[numero]** compuesto | ε
lista_var → lista_var**,** **id** | **id**
funciones → **func** tipo **id (**argumentos**)** bloque funciones | ε
argumentos → lista_args | ε
lista_args → lista_args**,** tipo **id** | tipo **id**
bloque → **{** declaraciones instrucciones **}**
instrucciones → instrucciones sentencia | sentencia
sentencia → parte_izquierda **=** bool **;** | **if(**bool**)** sentencia
| **if(**bool**)** sentencia **else** sentencia
| **while(**bool**)** sentencia | **do** sentencia **while(**bool**)** | **break ;**
| bloque | **return** exp **;** | **return;** | **switch(** bool **) {** casos **}**
| **print** exp **;** | **scan** parte_izquierda
casos → caso casos | predeterminado | ε
caso → **case numero:** instrucciones
predeterminado → **default:** instrucciones
parte_izquierda → **id** localizacion | **id**
bool → bool **||** comb | comb
comb → comb **&&** igualdad | igualdad
igualdad → igualdad **==** rel | igualdad **!=** rel | rel
rel → exp **<** exp | exp **<=** exp | exp **>=** exp | exp **>** exp | exp
exp → exp **+** term | exp – term | term
term → term ***** unario | term **/** unario | term **%** unario | unario
unario → **!**unario | **-**unario | factor
factor → **(**bool**)** | **id** localizacion | **numero** | **cadena** | **true** | **false** |
id(parametros**)** | **id**
parametros → lista_param | ε
lista_param → lista_param **,** bool | bool
localizacion → localizacion **[** bool **]** | **[** bool **]**

Los símbolos terminales están resaltados.

DISEÑO DE EXPRESIONES REGULARES

Dentro de la gramática hay dos tipos de terminales: las palabras y símbolos reservados y las literales (incluyendo a los identificadores). Adicionalmente, se especifica que se deben admitir comentarios de la forma:

- **<* comentario *>** para comentarios de varias líneas.
- **–comentario** para comentarios de una línea.

Al analizar la gramática, encontramos los siguientes símbolos y palabras reservadas:

;	**,**	**:**	**[**	**]**	**(**	**)**	**{**	**}**	**int**
float	**double**	**void**	**=**	**!**	**&&**	**||**	**==**	**!=**	**<**
>	**<=**	**>=**	**+**	**-**	*****	**/**	**%**	**func**	**if**
else	**while**	**do**	**break**	**switch**	**case**	**default**	**true**	**false**	**return**
print	**scan**								

Estos símbolos y palabras reservadas son ellos mismos sus expresiones regulares. El único detalle de implementación es que en *flex* son las primeras expresiones que deben aparecer para no ser atrapadas por otro patrón.

Ahora, para el diseño de las literales nos basamos en las expresiones que se especificaron en la primera entrega. A esto tuvimos que añadir una forma de diferenciar entre flotantes y doubles, y optamos por una forma similar a como Java lo hace. Los floats deben terminar con una f. Finalmente, para los chars los decidimos poner al estilo C, entre comillas simples y aceptando un único carácter, incluyendo escapados como \t o \n.

Identificadores: Los identificadores pueden contener letras o números, pero deben iniciar con una letra. Decidimos no incluir el uso de guiones bajos como en la primera entrega, en donde por ambigüedad se resolvió preguntando al profesor si se admitían o no.
No se nos pide una longitud máxima, por lo que la expresión entonces consiste en una letra seguida de letras o números:

[a-zA-Z][a-zA-Z0-9]*

Números Enteros: Los enteros los manejamos sin signo (este se puede agregar mediante producciones de la gramática) y con la posibilidad de separar sus digitos mediante un guión bajo. El número no puede terminar o iniciar en guiones bajos o tener dos seguidos. En el diseño de nuestra expresión, para forzar a que termine en número añadimos un [0-9]+ al final. Para forzar las otras dos restricciones, permitimos los guiones bajos siempre que tengan al menos un entero antes, con [0-9]+_, y esto se puede repetir cuantas veces queramos o 0. Así, la expresión queda como:

([0-9]+_)*{[0-9]+}

Números de punto flotante (floats y double). Estos dos puntos tienen dos tipos de representación: una con punto decimal y otra con notación científica.

En la primera es forzoso que aparezca un punto y al menos un número antes o después de él, pero no es forzoso que aparezca en ambas posiciones. Así, llegamos a la siguiente expresión (usando la que ya teníamos de entero): **(entero.(entero)? | .entero)**
A las expresiones anteriores se les puede añadir opcionalmente un exponente, el cual consiste de una e (mayúscula o minúscula) seguida opcionalmente de un signo y forzosamente de un entero. La expresión para el exponente es **[eE][+\-]?entero**
Y entonces juntando con lo anterior: **(entero.(entero)? | .entero)([eE][+\-]?entero)?**
Finalmente, la notación científica admite también enteros seguidos forzosamente de exponentes: **entero[eE][+\-]?entero**
Los floats deberán terminar en una f o F, y los doubles no. Así, tenemos para los floats:

((entero.(entero)? | .entero)([eE][+\-]?entero)? | entero[eE][+\-]?entero)[Ff]

Y pata los doubles:

((entero.(entero)? | .entero)([eE][+\-]?entero)? | entero[eE][+\-]?entero)[Ff]

Caracteres: Los caracteres siguen el estilo C. Pueden ser un único símbolo entre comillas simples (sin permitir el salto de línea explícito y la propia comilla simple), o un carácter escapado (es decir una diagonal seguido de otro símbolo), de entre los cuales admito \t, \n y \'. Así, la expresión es:

'[^\n']' | '\\t' | '\\n' | '\\''

Nota: pongo doble \\ porque la diagonal se toma como carácter de escape en flex, por lo que para simbolizar el carácter \ tenemos que escaparlo: \\.

Cadenas: Dentro de las cadenas tenemos dos casos: con comillas simples y comillas dobles. Las primeras admiten un salto de línea dentro, así como comillas dobles, pero las simples deben ir escapadas (\'). Así, para estas cadenas la expresión es:

'([^\"'\n] | \\' | \" | \n)*'

Nota: las comillas dobles aparecen como \", pero simbolizan solo ", pues en flex las comillas dobles están reservadas.
Para las comillas dobles es lo mismo, pero reemplazando las simples por las dobles y sin permitir el salto de línea. Las comillas dobles en su representación de forma escapada, por cuestiones de flex se ve como \\\" pues hay que escapar \ y ". La expresión es:

\"([^\"'\n] | ' | \\\")*\"

Comentarios: Estas ya son las últimas expresiones. La de una línea es bastante sencilla, sólo son dos guiones seguidos de lo que sea excepto un salto de línea:

--[^\n]*

Para la de líneas múltiples nos basamos en la expresión para comentarios de C correcta que era pregunta de una de las tareas de teoría, pero reemplazando la / por < y >:

"<*"([^*]|*[^>])*"*>"

Esa expresión nos dice que debe empezar con <*, y luego dentro puede aparecer lo que sea, excepto *>. Para forzar eso en el primer rango dentro de la estrella permite un único carácter cualquiera excepto *; si este aparece debe estar seguido de otro que no sea >. Finalmente, la cadena debe terminar en *>.

ELIMINACIÓN DE LA RECURSIVIDDAD IZQUIERDA Y FACTORES COMUNES.

La producción lista_var → lista_var id | id tiene que ser reemplazada por:
lista_var → **id** lista_varP
lista_varP → **,** **id** lista_varP | ε

La producción lista_args → lista_args, tipo id | tipo id se reemplaza por:
lista_args → tipo **id** lista_argsP
lista_argsP → **,** tipo **id** lista_argsP | ε

La producción instrucciones → instrucciones sentencia | sentencia se reemplaza por:
instrucciones → sentencia instruccionesP
instruccionesP → sentencia instruccionesP | ε

Las producciones sentencia → **if(**bool**)** sentencia | **if(**bool**)** sentencia **else** sentencia contienen un factor común, por lo que las reemplazamos por:
sentencia → **if(**bool**)** sentencia sentenciaP
sentenciaP → ε | **else** sentencia

Lo mismo ocurre con sentencia→ **return** exp**;** | **return;**. Las reemplazamos por:
sentencia → **return** sentenciaPP, sentenciaPP → exp**;** | **;**

Las producciones parte_izquierda → **id** localizacion | **id** tienen un factor común, por lo que se factorizan de la siguiente forma:
parte_izquierda → **id** parte_izquierdaP
parte_izquierdaP → localizacion | ε

La producción bool → bool **||** comb | comb se reemplaza por:
bool → comb boolP
boolP → **||** comb boolP | ε

La producción comb → comb **&&** igualdad | igualdad se reemplaza por:
comb → igualdad combP
combP → **&&** igualdad combP | ε

Las producciones igualdad → igualdad **==** rel | igualdad **!=** rel | rel se reemplazan por:
igualdad → rel igualdadP
igualdadP → **==** rel igualdadP | **!=** rel igualdadP | ε

Las producciones rel → exp **<** exp | exp **<=** exp | exp **>=** exp | exp **>** exp | exp tienen un factor compun, por lo que las reemplazamos por:
rel → exp relP y relP → **<** exp | **<=** exp | **>=** exp | **>** exp | ε

Las producciones exp → exp **+** term | exp **–** term | term se reemplazan por:
exp → term expP
expP → **+** term expP | **-** term expP | ε

Reemplazamos term → term ***** unario | term **/** unario | term **%** unario | unario por:
term → unario termP
termP → ***** unario termP | **/** unario termP | **%** unario termP | ε

Las producciones factor → **id** localizacion | **id(**parametros**)** | **id** poseen un factor común, por lo que las reemplazamos por:
factor → **id** factorP
factorP → localizacion | **(**parametros**)** | ε

Reemplazamos lista_param → lista_param **,** bool | bool por:
lista_param → bool lista_paramP
lista_paramP → **,** bool lista_paramP | ε

Finalmente, reemplazamos localizacion → localizacion **[** bool **]** | **[** bool **]** por:
localizacion → **[** bool **]** localizacionP

localizacionP → **[** bool **]** localizacionP | ε

En la página siguiente se muestra la gramática completa resultante de este proceso.

GRAMÁTICA SIN RECURSIVIDAD IZQUIERDA

programa → declaraciones funciones
declaraciones → tipo lista_var **;** declaraciones | ε
tipo → basico compuesto
basico → **int** | **float** | **char** | **double** | **void**
compuesto → **[numero]** compuesto | ε
lista_var → **id** lista_varP
lista_varP → **,** **id** lista_varP | ε
funciones → **func** tipo **id (**argumentos**)** bloque funciones | ε
argumentos → lista_args | ε
lista_args → tipo **id** lista_argsP
lista_argsP → **,** tipo **id** lista_argsP | ε
bloque → **{** declaraciones instrucciones **}**
instrucciones → sentencia instruccionesP
instruccionesP → sentencia instruccionesP | ε
sentencia → parte_izquierda **=** bool **;** | **if(**bool**)** sentencia sentenciaP
| **while(**bool**)** sentencia | **do** sentencia **while(**bool**)** | **break ;**
| bloque | **return** sentenciaPP | **switch(** bool **) {** casos **}**
| **print** exp **;** | **scan** parte_izquierda
sentenciaP → ε | **else** sentencia
sentenciaPP → exp **;** | **;**
casos → caso casos | predeterminado | ε
caso → **case numero:** instrucciones
predeterminado → **default:** instrucciones
parte_izquierda → **id** parte_izquierdaP
parte_izquierdaP → localizacion | ε
bool → comb boolP
boolP → **||** comb boolP | ε
comb → igualdad combP
combP → **&&** igualdad combP | ε
igualdad → rel igualdadP
igualdadP → **==** rel igualdadP | **!=** rel igualdadP | ε
rel → exp relP
relP → **<** exp | **<=** exp | **>=** exp | **>** exp | ε
exp → term expP
expP → **+** term expP | **-** term expP | ε
term → unario termP
termP → ***** unario termP | **/** unario termP | **%** unario termP | ε
unario → **!**unario | **-**unario | factor
factor → **(**bool**)** | **numero** | **cadena** | **true** | **false** | **id** factorP
factorP → localizacion | **(**parametros**)** | ε
parametros → lista_param | ε
lista_param → bool lista_paramP
lista_paramP → **,** bool lista_paramP | ε
localizacion → **[** bool **]** localizacionP
localizacionP → **[** bool **]** localizacionP | ε

ADAPTACIÓN DE LA DEFINICIÓN DIRIGIDA POR SINTAXIS DE LA GRAMÁTICA ORIGINAL A LA GRAMÁTICA SIN RECURSIVIDAD Y FACTORES IZQUIERDOS.

REGLAS DE PRODUCCIÓN	**REGLAS SEMÁNTICAS**
programa → declaraciones funciones	PilaTS.push(nuevaTablaTS()) PilaTT.push(nuevaTablaTT()) dir = 0
declaraciones → tipo lista_var**;** declaraciones	lista_var.tipo = tipo.tipo
declaraciones → ε	
tipo → basico compuesto	compuesto.base = basico.tipo tipo.tipo = compuesto.tipo
basico → **int**	basico.tipo = int
basico → **float**	basico.tipo = float
basico → **char**	basico.tipo = char
basico → **double**	basico.tipo = double
basico → **void**	basico.tipo = void
compuesto→ **[numero]** $compuesto_1$	Si numero.lextipo = int entonces; $compuesto_1$.base = compuesto.base Sino: error(“Indexar sólo funciona con enteros”) FinSi compuesto.tipo = PilaTT.top().insertar(“array”, numero.val, $compuesto_1$.tipo)
compuesto → ε	compuesto.tipo = compuesto.base
lista_var → **id** lista_varP	lista_varP.tipo = lista_var.tipo Si ! PilaTS.top().buscar(id) Entonces: PilaTS.top.insertar(id, lista_var.tipo, dir, “var”, NULO)

	dir = dir+PilaTT.top().getTam(lista_var.tipo) Sino error("El id ya está declarado") FinSi
lista_varP → **,** **id** lista_varP$_1$	lista_varP$_1$.tipo = lista_varP.tipo Si ! PilaTS.top().buscar(id) Entonces: PilaTS.top.insertar(id, lista_varP.tipo, dir, "var", NULO) dir = dir+PilaTT.top().getTam(lista_varP.tipo) Sino error("El id no está declarado") FinSi
lista_varP → ε	
funciones → **func** tipo **id** **(**argumentos**)** bloque funciones	ListaRetorno = NULO Si ! PilaTS.top().buscar(id) Entonces: PilaTS.push(nuevaTablaSimbolos) PilaTT.push(nuevaTablaTipos) PilaDir.push(dir) dir = 0 Si equivalentesLista(ListaRetorno, tipo.tipo): PilaTS.top().insertar(id, tipo.tipo, -, 'func',argumentos.lista) genCod(label(id)) bloque.sig = nuevaEtq genCod(label(bloque.sig)) Sino: error("Los tipos de retorno no coinciden con los tipos de retorno de la función") FinSI Sino: error("El id ya está declarado") FinSI PilaTS.pop() PilaTT.pop() dir = PilaDir.pop() PilaTS.fondo().insertar(id, tipo.tipo, -, 'func',argumentos.lista)
funciones → ε	

argumentos → lista_args	argumentos.lista = lista_args.lista
argumentos → ε	argumentos.lista = NULO
lista_args → tipo **id** lista_argsP	Si ! PilaTS.top().buscar(id) Entonces: PilaTS.top.insertar(id, tipo.tipo, dir, "param", NULO) dir = dir + PilaTT.top().getTam(tipo.tipo) Sino error("El id ya está declarado") FinSi lista_argsP.listaH = nuevaListaArgs() lista_argsP.listaH.agregar(tipo.tipo) lista_args.lista = lista_argsP.listaS
lista_argsP → **,** tipo **id** lista_argsP$_1$	Si ! PilaTS.top().buscar(id) Entonces: PilaTS.top.insertar(id, tipo.tipo, dir, "param", NULO) dir = dir + PilaTT.top().getTam(tipo.tipo) Sino error("El id ya está declarado") FinSi lista_argsP$_1$.listaH = lista_argsP.listaH lista_argsP$_1$.listaH.agregar(tipo.tipo) lista_argsP.listaS = lista_argsP$_1$.listaS
lista_argsP → ε	lista_argsP.listaS = lista_argsP.listaH
bloque→**{**declaraciones instrucciones**}**	instrucciones.sig = bloque.sig
instrucciones → sentencia instruccionesP	sentencia.sig = nuevaEtq() genCod(label(sentencia.sig))
instruccionesP→sentencia instruccionesP$_1$	sentencia.sig = nuevaEtq() genCod(label(sentencia.sig))
instruccionesP → ε	
sentencia → parte_izquierda **=** bool **;**	Si equivalentes(parte_izquierda.tipo, bool.tipo): // Cambiamos el valor de parte_izquierda d$_1$=reducir(bool.dir, bool.tipo,parte_izquierda.tipo)

	genCode(parte_izquierda.dir '=' d_1) Sino error("Tipos incompatibles") FinSi
sentencia → **if(**bool**)** $sentencia_1$ sentenciaP	bool.vddr = nuevaEtq() bool.fls = nuevoIndice() $sentencia_1$.sig = sentencia.sig sentenciaP.sig = sentencia.sig sentenciaP.lista_indices = nuevaListaIndices() sentenciaP.lista_indices.agregar(bool.fls) genCod(label(bool.vddr)) genCod(label(bool.vddr))
sentencia → **while(**bool**)** $sentencia_1$	$sentencia_1$.sig = nuevaEtq() bool.vddr = nuevaEtq() bool.fls = sentencia.sig genCod(label($sentencia_1$.sig)) genCod(label(bool.vddr)) genCod('goto' $sentencia_1$.sig)
sentencia →**do** $sentencia_1$ **while(**bool**)**	bool.vddr = nuevaEtq() bool.fls = sentencia.sig $sentencia_1$.sig = nuevaEtq() genCod(label(bool.vddr)) genCod(label($sentencia_1$.sig))
sentencia → **break ;**	genCod(goto sentencia.sig)
sentencia → bloque	bloque.sig = sentencia.sig
sentencia → **return** sentenciaPP	// Se pasa a sentenciaPP
sentencia → **switch(** bool **) { ** casos **}**	casos.etqprueba = nuevaEtq() genCode('goto' casos.etqprueba) casos.sig = sentencia.sig casos.id = bool.dir finswitch = nuevaEtq() genCode("goto" finswitch) genCode(label(casos.etqprueba)) genCode(casos.prueba) genCode(label(finswitch))

sentencia → **print** exp	genCode(‘print’, exp.dir) // Creo que sólo se genera el código.

sentencia → **scan** parte_izquierda	// No entiendo qué significaría semánticamente scan id para el programa.
	genCode(‘scan’, parte_izquierda.dir)

sentenciaP → **else** sentencia	sentencia.sig = sentenciaP.sig reemplazarIndices(sentenciaP.lista_indices, nuevaEtq(), cuadruplas)
sentenciaP → ε	reemplazarIndices(sentenciaP.lista_indices, sentenciaP.sig, cuadruplas)
sentenciaPP → exp **;**	ListaRetorno.agregar(exp.tipo) genCod(‘return’ exp.dir)
sentenciaPP → **;**	ListaRetorno.agregar(void) genCod(‘return’)
casos → caso $casos_1$	$casos_1$.sig = casos.sig caso.sig = casos.sig casos.prueba = caso.prueba || $casos_1$.prueba
casos → predeterminado	predeterminado.sig = casos.sig casos.prueba = predeterminado.prueba
casos → ε	casos.prueba = “”
caso → **case numero:** instrucciones	caso.inicio = nuevaEtq() instrucciones.sig = caso.sig caso.prueba = genCod(if caso.id ‘==’ numero.lexval ‘goto’ caso.inicio) genCode(label(caso.inicio))
predeterminado → **default:** instrucciones	predeterminado.inicio = nuevaEtq() instrucciones.sig = predeteriminado.sig predeterminado.prueba = genCod(‘goto’, predeterminado.inicio) genCode(label(predeterminado.inicio))
parte_izquierda →**id** parte_izquierdaP	parte_izquierdaP.base = id.lexval parte_izquierda.dir = parte_izquierdaP.dir

	parte_izqiuerda.tipo = parte_izquierdaP.tipo
parte_izquierdaP → localizacion	localizacion.base = parte_izquierdaP.base parte_izquierdaP.dir = localizacion.dir parte_izqiuerdaP.tipo = localizacion.tipo
parte_izquierdaP → ε	Si PilaTS.top().buscar(parte_izquierdaP.base) Entonces: parte_izquierdaP.dir = parte_izquierdaP.base parte_izquierdaP.tipo = PilaTS.top(),getTipo(parte_izquierdaP.dir) Sino error(”El id no está declarado”) FinSi
bool → comb boolP	comb.vddr = bool.vddr comb.fls = nuevoIndice() boolP.tipoH = comb.tipo boolP.lista_indices = nuevaListaIndices() boolP.lista_indices.agregar(comb.fls) boolP.first = true boolP.hfls = comb.fls bool.tipo = boolP.tipoS
boolP → **||** comb $boolP_1$	Si equivalentes(boolP.tipoH, comb.tipo) Entonces: Si boolP.first: genCod(label(boolP.hfls)) comb.vddr = boolP.vddr comb.fls = nuevoIndice() $boolP_1$.tipoH = comb.tipo $boolP_1$.vddr = boolP.vddr $boolP_1$.fls = boolP.fls $boolP_1$.lista_indices = boolP.lista_indices $boolP_1$.lista_indices.agregar(comb.fls) $boolP_1$.first = false boolP.tipoS = $boolP_1$.tipoS genCod(label($boolP_1$.fls)) Sino: error(“Tipos Incompatibles”) FinSi
boolP → ε	reemplazarIndice(boolP,lista_indices, boolP.fls, cuadruplas)

	boolP.tipoS = boolP.tipoH
comb → igualdad combP	igualdad.vddr = nuevoIndice() igualdad.fls = comb.fls combP.tipoH = igualdad.tipo combP.lista_indices = nuevaListaIndices() combP.lista_indices.agregar(igualdad.vddr) combP.first = true combP.vddr = igualdad.vddr comb.tipo = combP.tipoS genCod(label(igualdad.vddr))
combP → **&&** igualdad $combP_1$	Si equivalentes(combP.tipoH, igualdad.tipo) ent: Si combP.first: genCod(label(combP.vddr) igualdad.vddr = nuevoIndice() igualdad.fls = combP.fls $combP_1$.tipoH = igualdad.tipo $combP_1$.vddr = combP.vddr $combP_1$.fls = combP.fls $combP_1$.lista_indices = combP.lista_indices $combP_1$.lista_indices.agregar(igualdad.vddr) $combP_1$.first = false combP.tipoS = $combP_1$.tipoS genCod(label(igualdad.vddr)) Sino error(“Tipos Incompatibles”) FinSi
combP → ε	reemplazarIndices(combP.lista_indices, combP.vddr, cuadruplas) combP.tipoS = combP.tipoH
igualdad → rel igualdadP	igualdadP.fls = igualdad.fls igualdadP.vddr = igualdad.vddr igualdadP.tipoH = rel.tipo igualdadP.dirH = rel.dir igualdad.tipo = igualdadP.tipoS igualdad.dir = igualdadP.dirS
igualdadP → **==** rel $igualdadP_1$	Si equivalentes(igualdadP.tipoH, rel.tipo) entonces: $igualdadP_1$.tipoH = int

	igualdadP.dirS = nuevaTemporal() tTemp = maximo(igualdadP.tipoH, rel.tipo) d_1=ampliar(igualdadP.dirH, igualdadP.tipoH, tTemp) d_2 = ampliar(rel.dir, rel.tipo, tTemp) $igualdadP_1$.dirH = igualdadP.dirS igualdadP.tipoS = $igualdadP_1$.tipoS genCod(igualdadP.dirS '=' d_1.dir, '==', d_2.dir) genCod('if' igualdad.dir 'goto' rel.vddr) genCod('goto' rel.fls) Sino error("Tipos Incompatibles") FinSi
igualdadP → **!=** rel $igualdadP_1$	Si equivalentes(igualdad.tipoH, rel.tipo) entonces: $igualdadP_1$.tipoH = rel.tipo igualdadP.dirS = nuevaTemporal() tTemp = maximo(igualdadP.tipoH, rel.tipo) d_1=ampliar(igualdadP.dirH, igualdadP.tipoH,tTemp) d_2 = ampliar(rel.dir, rel.tipo, tTemp) $igualdadP_1$.dirH = igualdadP.dirS igualdadP.tipoS = $igualdadP_1$.tipoS genCod(igualdadP.dirS '=' d_1.dir, '!=', d_2.dir) genCod('if' igualdad.dir 'goto' rel.vddr) genCod('goto' rel.fls) Sino error("Tipos Incompatibles") FinSi
igualdadP → ε	igualdadP.tipoS = igualdadP.tipoH igualdadP.dirS = igualdad.dirH
rel → exp relP	relP.vddr = rel.vddr relP.fls = rel.fls relP.tipoH = exp.tipo relP.dirH = exp.dir rel.tipo = relP.tipoS rel.dir = relP.dirS
relP → **<** exp	Si equivalentes(relP.tipoH, exp.tipo) entonces: relP.tipo = int relP.dirS = nuevaTemporal()

	tTemp = maximo(relP.tipoH, exp.tipo) d_1= ampliar(relP.dirH, relP.tipoH,tTemp) d_2 = ampliar(exp.dir, exp.tipo, tTemp) genCod(relP.dirS '=' d_1.dir, '<', d_2.dir) genCod('if' relP.dirS 'goto' relP.vddr) genCod('goto' relP.fls) Sino error("Tipos Incompatibles") FinSi
relP → **<=** exp	Si equivalentes(relP.tipoH, exp.tipo) entonces: relP.tipo = int relP.dirS = nuevaTemporal() tTemp = maximo(relP.tipoH, exp.tipo) d_1=ampliar(relP.dirH, relP.tipoH,tTemp) d_2 = ampliar(exp.dir, exp.tipo, tTemp) genCod(relP.dirS '=' d_1.dir, '<=', d_2.dir) genCod('if' relP.dirS 'goto' relP.vddr) genCod('goto' relP.fls) Sino error("Tipos Incompatibles") FinSi
relP → **>=** exp	Si equivalentes(relP.tipoH, exp.tipo) entonces: relP.tipo = int relP.dirS = nuevaTemporal() tTemp = maximo(relP.tipoH, exp.tipo) d_1=ampliar(relP.dirH, relP.tipoH,tTemp) d_2 = ampliar(exp.dir, exp.tipo, tTemp) genCod(relP.dirS '=' d_1.dir, '>=', d_2.dir) genCod('if' relP.dirS 'goto' relP.vddr) genCod('goto' relP.fls) Sino error("Tipos Incompatibles") FinSi
relP → **>** exp	Si equivalentes(relP.tipoH, exp.tipo) entonces: relP.tipo = int relP.dirS = nuevaTemporal() tTemp = maximo(relP.tipoH, exp.tipo) d_1=ampliar(relP.dirH, relP.tipoH,tTemp) d_2 = ampliar(exp.dir, exp.tipo, tTemp)

	genCod(relP.dirS '=' d_1.dir, '>', d_2.dir) genCod('if' relP.dirS 'goto' relP.vddr) genCod('goto' relP.fls) Sino error("Tipos Incompatibles") FinSi
relP → ε	relP.tipoS = relP.tipoH relP.dirS = relP.dirH
exp → term expP	expP.tipoH = term.tipo expP.dirH = term.dir exp.tipo = expP.tipoS exp.dir = expP.dirS
expP → **+** term $expP_1$	Si equivalentes(expP.tipoH, term.tipo) Entonces: $expP_1$.tipoH = maximo(expP.tipoH,term.tipo) $expP_1$.dirH = nuevaTemporal() expP.tipoS = $expP_1$.tipoS expP.dirS = $expP_1$.dirS d_1= ampliar(expP.dirH,expP.tipoH,$expP_1$.tipo) d_2= ampliar(term.dir, term.tipo,$expP_1$.tipoH) genCod(expP.dirS '=' d_1.dir, '+', d_2.dir) Sino error("Tipos Incompatibles") FinSi
expP → **-** term $expP_1$	Si equivalentes(expP.tipoH, term.tipo) Entonces: $expP_1$.tipoH = maximo(expP.tipoH,term.tipo) $expP_1$.dirH = nuevaTemporal() expP.tipoS = $expP_1$.tipoS expP.dirS = $expP_1$.dirS d_1= ampliar(expP.dirH,expP.tipoH,$expP_1$.tipo) d_2= ampliar(term.dir, term.tipo,$expP_1$.tipoH) genCod(expP.dirS '=' d_1.dir, '-', d_2.dir) Sino error("Tipos Incompatibles") FinSi
expP → ε	expP.tipoS = expP.tipoH

	expP.dirS = expP.dirH
term → unario termP	termP.tipoH = unario.tipo termP.dirH = unario.dir term.tipo = termP.tipoS term.dir = termP.dirS
termP → ***** unario $termP_1$	Si equivalentes(termP.tipoH, unario.tipo) Entonces: $termP_1$.tipoH = max(termP.tipoH,unario.tipo) $termP_1$.dirH = nuevaTemporal() termP.tipoS = $termP_1$.tipoS termP.dirS = $termP_1$.dirS d_1=ampliar(termP.dirH, termP.tipoH, $termP_1$.tipoH) d_2= ampliar(unario.dir, unario.tipo, $termP_1$.tipoH) genCod($termP_1$.dirH '=' d_1.dir, '*', d_2.dir) Sino error("Tipos Incompatibles") FinSi
termP → **/** unario $termP_1$	Si equivalentes(termP.tipoH, unario.tipo) Entonces: $termP_1$.tipoH = max(termP.tipoH,unario.tipo) $termP_1$.dirH = nuevaTemporal() termP.tipoS = $termP_1$.tipoS termP.dirS = $termP_1$.dirS d_1=ampliar(termP.dirH, termP.tipoH, $termP_1$.tipoH) d_2= ampliar(unario.dir, unario.tipo, $termP_1$.tipoH) genCod($termP_1$.dirH '=' d_1.dir, '/', d_2.dir) Sino error("Tipos Incompatibles") FinSi
termP → **%** unario $termP_1$	Si termP.tipoH==int and unario.tipo==int entonces: $termP_1$.tipoH = int $termP_1$.dirH = nuevaTemporal() termP.tipoS = $termP_1$.tipoS termP.dirS = $termP_1$.dirS

	genCod(termP$_1$.dirH ‘=’ termP.dirH, ‘%’, unario.dir) Sino error(“Tipos Incompatibles”) FinSi
termP → ε	termP.tipoS = termP.tipoH termP.dirS = termP.dirH
unario → **!**unario$_1$	unario.dir = nuevaTemporal() unario.tipo = unario$_1$.tipo genCod(unario.dir ‘=’ ‘!’ unario$_1$.dir)
unario → **-**unario$_1$	unario.dir = nuevaTemporal() unario.tipo = unario$_1$.tipo genCod(unario.dir ‘=’ ‘-’ unario$_1$.dir)
unario → factor	unario.dir = factor.dir unario.tipo = factor.tipo
factor → **(**bool**)**	factor.tipo = bool.tipo factor.dir = bool.dir

factor → **numero**	//factor.tipo = numero.lexval
	factor.val = numero.lexval
	factor.tipo = numero.lextipo

factor → **cadena**	TablaCadenas.agregar(cadena.lexval) factor.dir = TablaCadenas.getUltimaPos() factor.tipo = cadena
factor → **true**	factor.dir = ‘true’ factor.tipo = int
factor → **false**	factor.dir = ‘false’ factor.tipo = int
factor → **id** factorP	factorP.base = id.lexval factor.dir = factorP.dir factor.tipo = factorP.tipo
factorP → localizacion	localizacion.base = factorP.base factorP.dir = nuevaTemporal() factorP.tipo = localizacion.tipo

	genCod(factorP.dir '=' factorP.base '[' localizacion.dir ']')
factorP → **(**parametros**)**	Si PilaTs.fondo().buscar(factorP.base) Entonces: Si PilaTs.fondo().getVar(factorP.base) = 'func': Si equivalenteListas(PilaTS.fondo().getArgs(factorP.base), parametros.lista) Entonces: factor.tipo = PilaTS.top.getTipo(id) factor.dir = nuevaTemporal() genCod(factor.dir '=' 'call' factorP.base parametros.lista.tam) Sino error("El número o tipo de parámetros no coicndice") FinSi Sino error("El id no es función") FinSi Sino error("El id no está declarado") FinSi
factorP → ε	factorP.dir = factorP.base factorP.tipo =PilaTS.top().getTipo(factorP.dir)
parametros → lista_param	parametros.lista = lista_param.lista
parametros → ε	parametros.lista = NULO
lista_param → bool lista_paramP	lista_paramP.listaH = nuevaLista() lista_paramP.listaH.agregar(bool.tipo) lista_param.lista = lista_paramP.listaS genCode('param' bool.dir)
lista_paramP → **,** bool lista_paramP$_1$	lista_paramP$_1$.listaH = lista_paramP.listaH lista_paramP$_1$.listaH.agregar(bool.tipo) lista_paramP.listaS = lista_paramP$_1$.listaS genCode('param' bool.dir)
lista_paramP → ε	lista_paramP.listaS = lista_paramP.listaH
localizacion →**[** bool **]** localizacionP	%alias loc=localizacion Si PilaTS.top().buscar(loc.base) Entonces Si bool.tipo = int Entonces

	tipoTmp = PilaTS.top().getTipo(loc.base) Si PilaTT.top().getNombre(tipoTmp)='array': locP.tipoH=PilaTT.top().getTipoBase(tipoTmp) locP.dirH = nuevaTemporal() locP.tam = PilaTT.top().getTam(locP.tipoH) genCod(locP.dirH '=' bool.dir '*' locP.tam) localizacion.dir = localizacionP.dirS localizacion.tipo = localizacionP.tipoS Sino error("El id no es un arreglo") FinSi Sino error("El indice del arreglo debe ser entero") FinSi Sino error("El id no está declarado") FinSi
localizacionP→**[**bool**]** localizacionP$_1$	% alias loc = localizacion Si bool.tipo = int Entonces tipoloc = PilaTT.top().getNombre(locP.tipo) Si tipoloc = 'array' Entonces locP$_1$.tipoH=PilaTT.top().getBase(locP.tipo) dirTmp = nuevaTemporal() locP$_1$.dirH = nuevaTemporal() locP$_1$.tam = PilaTT.top().getTam(locP.tipo) genCod(dirTmp '=' bool.dir '*' locP$_1$.tam) genCod(locP$_1$.dir '=' locP.dirH '+' dirTmp) locP.dirS = locP$_1$.dirS locP.tipoS = locP$_1$.tipoS Sino error("El id no es un arreglo") FinSi Sino error("El indice del arreglo debe ser entero") FinSi
localizacionP → ε	localizacionP.dirS = localizacionP.dirH localizacionP.tipoS = localizacionP.tipoH

Modificaciones Adicionales a la eliminación de recursión.

Primera modificación: En **compuesto → [numero] compuesto$_1$** volteamos las reglas semánticas pues necesitamos primero el tipo de compuesto$_1$ antes de insertar el tipo de compuesto. Además, revisamos que el numero sea de tipo entero, si no mandamos error.

En **funciones** también se cambia para buscar si el id está en el ambiente anterior (pues en la regla que se nos dio se busca en una tabla de símbolos nueva y por lo tanto vacía) y

después ya se agregan las nuevas tablas a la pila. Al terminar, después de los pops, agregamos el id de la función también al ambiente anterior (que por la estructura de nuestra gramática siempre será el global). Esto lo hacemos porque si sólo metiéramos a las funciones únicamente dentro de su propio ambiente no podríamos llamar a la función dentro de otras funciones, lo cual no es ideal para un lenguaje de programación,

Agregamos una etiqueta al final del switch para siempre saltar al código después de las instrucciones de prueba para el switch (si no se ciclaría)
La base de factor tiene que heredarse hacia factorP para el genCode y llegar a localizacion pata obtener el primer tipo en el descenso de los tiposBase.

En boolP y combP agregamos un atributo heredado, first que nos indican si son el primero de la recursión. Esto lo hacemos para empujar la generación de las etiquetas que se hacía en bool y comb a estas producciones, con el fin de evitar etiquetas incorrectas generadas sólo del proceso de descenso (cuando no se usa || o &&) Esto es para emular la “concatenación” del código que vimos en clase a través de la recursión.

En **exp→ +term** (y -) se cambió una función maximo por ampliar, pues es a lo que se refería.

En localizacion se multiplica por el tamaño del tipo de localizacionP, no de localizacion.

ESQUEMA DE TRADUCCIÓN

Aquí, a diferencia de lo anterior donde los terminales estaban resaltados, toda la parte sintáctica de las producciones se presenta resaltado para hacer más facil la lectura del esquema.

programa → { PilaTS.push(nuevaTablaTS())
PilaTT.push(nuevaTablaTT())
dir = 0
} **declaraciones funciones**

declaraciones → **tipo** { lista_var.tipo = tipo.tipo } **lista_var ; declaraciones**
declaraciones → ε

tipo → **basico** { compuesto.base = basico.tipo } **compuesto** { tipo.tipo = compuesto.tipo }

basico → **int** {basico.tipo = int }
basico → **float** {basico.tipo = float}
basico → **char** {basico.tipo = char}
basico → **double** {basico.tipo = double}
basico → **void** {basico.tipo = void}

compuesto → **[numero]**
{ Si numero.lextipo = int:
$compuesto_1$.base = compuesto.base
Sino:
error(“Indexar sólo funciona con enteros”)

FinSi
} **compuesto$_1$**
{ compuesto.tipo = PilaTT.top().insertar("array", numero.valor, compuesto$_1$.tipo) }

compuesto → ε{compuesto.tipo = compuesto.base}

lista_var → **id** {lista_varP.tipo = lista_var.tipo
Si ! PilaTS.top().buscar(id) Entonces:
PilaTS.top.insertar(id, lista_var.tipo, dir, "var", NULO)
dir = dir+PilaTT.top().getTam(lista_var.tipo)
Sino
error("El id ya está declarado")
FinSi } **lista_varP**

lista_varP → **,** **id** {lista_varP$_1$.tipo = lista_varP.tipo
Si ! PilaTS.top().buscar(id) Entonces:
PilaTS.top.insertar(id, lista_varP.tipo, dir, "var", NULO)
dir = dir+PilaTT.top().getTam(lista_varP.tipo)
Sino
error("El id ya está declarado")
FinSi } **lista_varP$_1$**

lista_varP → ε

funciones → { ListaRetorno = NULO } **func tipo id**
{ Si PilaTS.top().buscar(id) Entonces:
error("El id ya está declarado")
PilaTS.push(nuevaTablaSimbolos)
PilaTT.push(nuevaTablaTipos)
PilaDir.push(dir)
dir = 0
genCod(label(id))
} **(argumentos)** {bloque.sig = nuevaEtq} **bloque**
{Si equivalentesLista(ListaRetorno, tipo.tipo):
PilaTS.top().insertar(id, tipo.tipo, -, 'func',argumentos.lista)
genCod(label(bloque.sig)
Sino:
error("Los tipos de retorno no coinciden
con los tipos de retorno de la función")
FinSi
PilaTS.pop()
PilaTT.pop()
dir = PilaDir.pop()
PilaTS.fondo().insertar(id, tipo.tipo, -, 'func',argumentos.lista)
} **funciones**

funciones → **ε**

argumentos → **lista_args** {argumentos.lista = lista_args.lista }
argumentos → **ε** {argumentos.lista = NULO}

lista_args → **tipo id**
{Si PilaTS.top().buscar(id) Entonces:
error("El id ya está declarado")
PilaTS.top.insertar(id, tipo.tipo, dir, "param", NULO)
dir = dir + PilaTT.top().getTam(tipo.tipo)
lista_argsP.listaH = nuevaListaArgs()
lista_argsP.listaH.agregar(tipo.tipo)
} **lista_argsP**
{ lista_args.lista = lista_argsP.listaS }

lista_argsP → **, tipo id**
{Si PilaTS.top().buscar(id) Entonces:
error("El id ya está declarado")
PilaTS.top.insertar(id, tipo.tipo, dir, "param", NULO)
dir = dir + PilaTT.top().getTam(tipo.tipo)
lista_argsP$_1$.listaH = lista_argsP.listaH
lista_argsP$_1$.listaH.agregar(tipo.tipo)
} **lista_argsP$_1$**
{ lista_argsP.lista = lista_argsP$_1$.listaS}

lista_argsP → **ε**{lista_argsP.listaS = lista_argsP.listaH}

bloque → **{ declaraciones** {instrucciones.sig = bloque.sig} **instrucciones }**

instrucciones → {sentencia.sig = nuevaEtq()} **sentencia**
{genCod(label(sentencia.sig)) } **instruccionesP**

instruccionesP → {sentencia.sig = nuevaEtq()} **sentencia**
{genCod(label(sentencia.sig)) } **instruccionesP$_1$**

instruccionesP → **ε**

sentencia → **parte_izquierda = bool ;** {
Si equivalentes(parte_izquierda.tipo, bool.tipo):
d$_1$=reducir(bool.dir, bool.tipo,parte_izquierda.tipo)
genCode(parte_izquierda.dir '=' d$_1$)
Sino
error("Tipos incompatibles")
FinSi}

sentencia → **if** {{bool.vddr = nuevaEtq() bool.fls = nuevoIndice()}

(bool) { sentencia$_1$.sig = sentencia.sig, genCod(label(bool.vddr))}
sentencia$_1$ {sentenciaP.lista_indices = nuevaListaIndices()
sentenciaP.lista_indices.agregar(bool.fls)
genCod(label(bool.vddr))}
sentenciaP

sentencia → **while (** { {sentencia$_1$.sig = nuevaEtq()
bool.vddr = nuevaEtq()
bool.fls = sentencia.sig}
genCod(label(sentencia$_1$.sig))}
bool) {genCod(label(bool.vddr))} **sentencia$_1$** { genCod('goto' sentencia$_1$.sig)}

sentencia→ **do** { bool.vddr = nuevaEtq()
bool.fls = sentencia.sig
sentencia$_1$.sig = nuevaEtq()
genCod(label(bool.vddr))
genCod(label(sentencia$_1$.sig))}
sentencia$_1$ while (bool)

sentencia→ **break** {genCod(goto sentencia.sig)} **;**

sentencia→ {bloque.sig = sentencia.sig} **bloque**

sentencia → **return sentenciaPP**

sentencia → **switch(bool)** {casos.etqprueba = nuevaEtq()
genCode('goto' casos.etqprueba)
casos.sig = sentencia.sig
casos.id = bool.dir}
{ casos } { finswitch = nuevaEtq()
genCode("goto" finswitch)
genCode(label(casos.etqprueba))
genCode(casos.prueba)
genCode(label(finswitch))}

sentencia → **print exp** {genCode('print', exp.dir)}

sentencia → **scan exp** {genCode('scan', parte_izquierda.dir) }

sentenciaP → **else** {sentencia.sig = sentenciaP.sig
sentencia {
reemplazarIndices(sentenciaP.lista_indices, nuevaEtq(), cuadruplas)} }

sentenciaP → **ε** {reemplazarIndices(sentenciaP.lista_indices, sentenciaP.sig,cuadruplas)}

sentenciaPP → **exp ;** {ListaRetorno.agregar(exp.tipo)
genCod('return' exp.dir)}

sentenciaPP → **;** { ListaRetorno.agregar(void)
genCod('return')}

casos → {caso.sig = casos.sig} **caso** {casos$_1$.sig = casos.sig} **casos$_1$**
{casos.prueba = caso.prueba || casos$_1$.prueba}

casos → {predeterminado.sig = casos.sig} **predeterminado**

{casos.prueba =predeterminado.prueba}

caso → {caso.inicio = nuevaEtq()} **case numero:**
{caso.prueba = genCod(if caso.id '==' numero.lexval 'goto' caso.inicio)
genCode(label(caso.inicio))}
instrucciones

predeterminado → {predeterminado.inicio = nuevaEtq()
instrucciones.sig = predeteriminado.sig
predeterminado.prueba = genCod('goto', predeterminado.inicio)
genCode(label(predeterminado.inicio))}
default: instrucciones

parte_izquierda → **id** {parte_izquierdaP.base = id.lexval} **parte_izquierdaP**
{parte_izquierda.dir = parte_izquierdaP.dir
parte_izqiuerda.tipo = parte_izquierdaP.tipo}

parte_izquierdaP → {localizacion.base = parte_izquierdaP.base } **localizacion**
{parte_izquierdaP.dir = localizacion.dir
parte_izqiuerdaP.tipo = localizacion.tipo}

parte_izquierda →**ε**{Si PilaTS.top().buscar(parte_izquierdaP.base) Entonces:
parte_izquierdaP.dir = parte_izquierdaP.base
parte_izquierdaP.tipo = PilaTS.top(),getTipo(parte_izquierdaP.dir)
Sino
error("El id no está declarado")
FinSi}

bool → {comb.vddr = bool.vddr
comb.fls = nuevoIndice()}
comb
{boolP.tipoH = comb.tipo
boolP.lista_indices = nuevaListaIndices()
boolP.lista_indices.agregar(comb.fls)}
boolP.first = true
boolP.hfls = comb.fls}
boolP
{bool.tipo = boolP.tipoS}

boolP → **||** {comb.vddr = boolP.vddr
comb.fls = nuevoIndice()}
comb {Si !equivalentes(boolP.tipoH, comb.tipo): error("Tipos Incompatibles")
Si boolP.first: genCod(label(comb.hfls))
$boolP_1$.tipoH = comb.tipo
$boolP_1$.vddr = boolP.vddr
$boolP_1$.fls = boolP.fls

boolP$_1$.lista_indices = boolP.lista_indices
boolP$_1$.lista_indices.agregar(comb.fls)
boolP$_1$.first = false
genCod(label(boolP$_1$.fls)) }
boolP$_1$ { boolP.tipoS = int }

boolP → **ε** {reemplazarIndice(boolP,lista_indices, boolP.fls, cuadruplas)
boolP.tipoS = bool.tipoH}

comb → {igualdad.vddr = nuevoIndice()
igualdad.fls = comb.fls}
igualdad
{combP.tipoH = igualdad.tipo
combP.lista_indices = nuevaListaIndices()
combP.lista_indices.agregar(igualdad.vddr)
combP.first = false
combP.vddr = igualdad.vddr}
combP
{comb.tipo = combP.tipoS}

combP → **&&** {igualdad.vddr = nuevoIndice(); igualdad.fls = combP.fls}
igualdad
{Si !equivalentes(combP.tipoH, igualdad.tipo)
error(“Tipos Incompatibles)
Si combP.first: genCod(label(combP.vddr)
combP$_1$.tipoH = igualdad.tipo
combP$_1$.vddr = combP.vddr
combP$_1$.fls = combP.fls
combP$_1$.lista_indices = combP.lista_indices
combP$_1$.lista_indices.agregar(igualdad.vddr)
genCod(label(igualdad.vddr))}
combP$_1$
{combP.tipoS = combP$_1$.tipoS ¿o int?}

combP → **ε**{reemplazarIndices(combP.lista_indices, combP.vddr, cuadruplas)
quedarseConUltima(combP.lista_indices, combP.vddr)
combP.tipoS = combP.tipoH}

igualdad → **rel** {igualdadP.fls = igualdad.fls
igualdadP.vddr = igualdad.vddr
igualdadP.tipoH = rel.tipo
igualdadP.dirH = rel.dir}
igualdadP
{igualdad.tipo = igualdadP.tipoS
igualdad.dir = igualdadP.dirS}

igualdadP → **== rel**
{Si !equivalentes(igualdadP.tipoH, rel.tipo)
error("Tipos Incompatibles")
igualdadP$_1$.tipoH = int
igualdadP.dirS = nuevaTemporal()
tTemp = maximo(igualdadP.tipoH, rel.tipo)
d_1=ampliar(igualdadP.dirH, igualdadP.tipoH,tTemp)
d_2 = ampliar(rel.dir, rel.tipo, tTemp)
igualdadP$_1$.dirH = igualdadP.dirS
genCod(igualdadP.dir '=' d_1.dir, '==', d_2.dir)
genCod('if' igualdad.dir 'goto' rel.vddr)
genCod('goto' rel.fls)}
}
igualdadP$_1$
{igualdadP.tipoS = igualdadP$_1$.tipoS}

igualdadP → **!= rel**
{Si !equivalentes(igualdadP.tipoH, rel.tipo)
error("Tipos Incompatibles")
igualdadP$_1$.tipoH = int
igualdadP.dirS = nuevaTemporal()
tTemp = maximo(igualdadP.tipoH, rel.tipo)
d_1=ampliar(igualdadP.dirH, igualdadP.tipoH,tTemp)
d_2 = ampliar(rel.dir, rel.tipo, tTemp)
genCod(igualdadP.dirS '=' d_1.dir, '!=', d_2.dir)
genCod('if' igualdad.dir 'goto' rel.vddr)
genCod('goto' rel.fls)
}
igualdadP$_1$
{igualdadP.tipoS = igualdadP$_1$.tipoS}

igualdadP → **ε** {igualdadP.tipoS = igualdadP.tipoH
igualdadP.dirS = igualdad.dirH}

rel → **exp** {relP.vddr = rel.vddr
relP.fls = rel.fls
relP.tipoH = exp.tipo
relP.dirH = exp.dir
relP { rel.tipo = relP.tipoS
rel.dir = relP.dirS}

relP → **< exp** {Si equivalentes(relP.tipoH, exp.tipo) entonces:
relP.tipo = int
relP.dirS = nuevaTemporal()
tTemp = maximo(relP.tipoH, exp.tipo)
d_1=ampliar(relP.dirH, relP.tipoH,tTemp)

d_2 = ampliar(exp.dir, exp.tipo, tTemp)
genCod(relP.dirS '=' d_1.dir, '<', d_2.dir)
genCod('if' relP.dirS 'goto' relP.vddr)
genCod('goto' relP.fls)
Sino
error("Tipos Incompatibles")
FinSi}

relP → **<= exp** {Si equivalentes(relP.tipoH, exp.tipo) entonces:
relP.tipo = int
relP.dirS = nuevaTemporal()
tTemp = maximo(relP.tipoH, rel.tipo)
d_1=ampliar(relP.dirH, relP.tipoH,tTemp)
d_2 = ampliar(exp.dir, exp.tipo, tTemp)
genCod(relP.dirS '=' d_1.dir, '<=', d_2.dir)
genCod('if' relP.dirS 'goto' relP.vddr)
genCod('goto' relP.fls)
Sino
error("Tipos Incompatibles")
FinSi}

relP → **>= exp** {Si equivalentes(relP.tipoH, exp.tipo) entonces:
relP.tipo = int
relP.dirS = nuevaTemporal()
tTemp = maximo(relP.tipoH, exp.tipo)
d_1=ampliar(relP.dirH, relP.tipoH,tTemp)
d_2 = ampliar(exp.dir, exp.tipo, tTemp)
genCod(relP.dirS '=' d_1.dir, '>=', d_2.dir)
genCod('if' relP.dirS 'goto' relP.vddr)
genCod('goto' relP.fls)
Sino
error("Tipos Incompatibles")
FinSi}

relP → **> exp** {Si equivalentes(relP.tipoH, exp.tipo) entonces:
relP.tipo = int
relP.dirS = nuevaTemporal()
tTemp = maximo(relP.tipoH, exp.tipo)
d_1=ampliar(relP.dirH, relP.tipoH,tTemp)
d_2 = ampliar(exp.dir, exp.tipo, tTemp)
genCod(relP.dirS '=' d_1.dir, '>', d_2.dir)
genCod('if' relP.dirS 'goto' relP.vddr)
genCod('goto' relP.fls)
Sino
error("Tipos Incompatibles")
FinSi}

relP → **ε** {relP.tipoS = relP.tipoH, relP.dirS = relP.dirH}

exp → **term** {expP.tipoH = term.tipo, expP.dirH = term.dir} **expP**
{exp.tipo = expP.tipoS, exp.dir = expP.dirS}

expP → **+ term**
{Si !equivalentes(expP.tipoH, term.tipo)
error("Tipos Incompatibles")
expP_1.tipoH = maximo(expP.tipoH, term.tipo)
expP_1.dirH = nuevaTemporal()
d_1 = ampliar(expP.dirH, expP.tipoH,$expP_1$.tipoH)
d_2 = ampliar(term.dir, term.tipo, $expP_1$.tipoH)
genCod(expP.dirS '=' d_1.dir, '+', d_2.dir)
} **$expP_1$** {
expP.tipoS = $expP_1$.tipoS
expP.dirS = $expP_1$.dirS}

expP → **- term**
{Si !equivalentes(expP.tipoH, term.tipo)
error("Tipos Incompatibles")
expP_1.tipoH = maximo(expP.tipoH, term.tipo)
expP_1.dirH = nuevaTemporal()
d_1 = ampliar(expP.dirH, expP.tipoH,$expP_1$.tipoH)
d_2 = ampliar(term.dir, term.tipo, $expP_1$.tipoH)
genCod(expP.dirS '=' d_1.dir, '-', d_2.dir)
} **$expP_1$** {
expP.tipoS = $expP_1$.tipoS
expP.dirS = $expP_1$.dirS}

expP → **ε**{expP.tipoS = expP.tipoH, expP.dirS = expP.dirH}

term → **unario** {termP.tipoH = unario.tipo, termP.dirH = unario.dir}
termP {term.tipo = termP.tipoS, term.dir = termP.dirS}

termP → *** unario**
{Si ! equivalentes(termP.tipoH, unario.tipo))
error("Tipos Incompatibles")
$termP_1$.tipoH = max(termP.tipoH,unario.tipo)
$termP_1$.dirH = nuevaTemporal()
d_1=ampliar(termP.dirH, termP.tipoH, $termP_1$.tipoH)
d_2= ampliar(unario.dir, unario.tipo, $termP_1$.tipoH)
genCod($termP_1$.dirH '=' d_1.dir, '*', d_2.dir)
} **$termP_1$**{
termP.tipoS = $termP_1$.tipoS
termP.dirS = $termP_1$.dirS}

termP → **/ unario**
{Si ! equivalentes(termP.tipoH, unario.tipo))
error(“Tipos Incompatibles”)
$termP_1$.tipoH = max(termP.tipoH,unario.tipo)
$termP_1$.dirH = nuevaTemporal()
genCod($termP_1$.dirH ‘=’ d_1.dir, ‘/’, d_2.dir)
d_1=ampliar(termP.dirH, termP.tipoH, $termP_1$.tipoH)
d_2= ampliar(unario.dir, unario.tipo, $termP_1$.tipoH)
} **$termP_1$**{
termP.tipoS = $termP_1$.tipoS
termP.dirS = $termP_1$.dirS}

termP → **% unario**
{Si ! (termP.tipoH==int and unario.tipo==int)
error(“Tipos Incompatibles”);
$termP_1$.tipoH = int
$termP_1$.dirH = nuevaTemporal()
genCod($termP_1$.dirH‘=’ termP.dirH, ‘%’, unario.dir)
} **$termP_1$** {
termP.tipoS = $termP_1$.tipoS
termP.dirS = $termP_1$.dirS}

termP → **ε** {termP.tipoS = termP.tipoH, termP.dirS = termP.dirH}

unario → **!** {unario.dir = nuevaTemporal()}
$unario_1$ {unario.tipo = $unario_1$.tipo
genCod(unario.dir ‘=’ ‘!’ $unario_1$.dir)}

unario → **-** {unario.dir = nuevaTemporal()}
$unario_1$ {unario.tipo = $unario_1$.tipo
genCod(unario.dir ‘=’ ‘-’ $unario_1$.dir)}
unario → **factor** {unario.dir = factor.dir, unario.tipo = factor.tipo}

factor → **(bool)** {factor.tipo = bool.tipo, factor.dir = bool.dir}
factor → **numero** {factor.dir = numero.lexval, factor.tipo = numero.lextipo}
factor → **cadena** {TablaCadenas.agregar(cadena.lexval)
factor.dir = TablaCadenas.getUltimaPos()
factor.tipo = cadena}
factor → **true** {factor.dir = ‘true’, factor.tipo = int}
factor → **false** {factor.dir = ‘false’, factor.tipo = int}
factor → **id** {factorP.base = id.lexval} **factorP** {factor.dir = factorP.dir,
factor.tipo = factorP.tipo}

factorP → **localizacion**
{factorP.dir = nuevaTemporal()

factorP.tipo = localizacion.tipo
genCod(factorP.dir '=' factorP.base '[' localizacion.dir ']')}

factorP → (parametros)
{Si PilaTs.fondo().buscar(factorP.base) Entonces:
Si PilaTs.fondo().getVar(factorP.base) = 'func':
Si equivalenteListas(PilaTS.fondo().getArgs(factorP.base), parametros.lista):
factor.tipo = PilaTS.top.getTipo(id)
factor.dir = nuevaTemporal()
genCod(factor.dir '=' 'call' factorP.base parametros.lista.tam)
Sino
error("El número o tipo de parámetros no coicndice")
FinSi
Sino
error("El id no es función")
FinSi
Sino
error("El id no está declarado")
FinSi}
factorP → ε {factorP.dir = factorP.base, factorP.tipo = PilaTS.top().getTipo(factorP.dir)}

parametros → {parametros.lista = lista_param.lista} **lista_param**
parametros → ε {parametros.lista = NULO}

lista_param → bool {lista_paramP.listaH = nuevaLista()
lista_paramP.listaH.agregar(bool.tipo)}
lista_paramP { lista_param.lista = lista_paramP.listaS
genCode('param' bool.dir)}

lista_paramP → , bool { lista_paramP$_1$.listaH = lista_paramP.listaH
lista_paramP$_1$.listaH.agregar(bool.tipo)
genCode('param' bool.dir)}
lista_paramP {lista_paramP.listaS = lista_paramP$_1$.listaS}
lista_paramP → ε {lista_paramP.listaS = lista_paramP.listaH}

localizacion → [bool]
{Si ! PilaTS.top().buscar(localizacion.base) Entonces
error("El id no está declarado")
Si bool.tipo != int Entonces
error("El indice del arreglo debe ser entero")
tipoTmp = PilaTS.top().getTipo(localizacion.base)
Si ! PilaTT.top().getNombre(tipoTmp)='array':
error("El id no es un arreglo")
localizacionP.tipoH = PilaTT.top().getTipoBase(tipoTmp)
localizacionP.dirH = nuevaTemporal()
localizacionP.tam = PilaTT.top().getTam(localizacionP.tipoH)
genCod(localizacionP.dirH '=' bool.dir '*' localizacionP.tam)

} **localizacionP**
{localizacion.dir = localizacionP.dirS
localizacion.tipo = localizacionP.tipoS}

localizacionP → **[bool]**
{Si bool.tipo != int Entonces:
error(”El indice del arreglo debe ser entero”)
Si ! PilaTT.top.getNombre(localizacionP.tipoH)=’array’ Entonces:
error(”El id no es un arreglo”)
$localizacionP_1$.tipoH = PilaTT.top().getTipoBase(localizacionP.tipo)
dirTmp = nuevaTemporal()
$localizacionP_1$.dirH = nuevaTemporal()
$localizacionP_1$.tam = PilaTT.top().getTam($localizacionP_1$.tipoH)
genCod(dirTmp ’=’ bool.dir ’*’ $localizacionP_1$.tam)
genCod($localizacionP_1$.dirH ’=’ localizacionP.dirH ’+’ dirTmp)
}**$localizacionP_1$**
{localizacionP.dirS = $localizacionP_1$.dirS
localizacionP.tipoS = $localizacionP_1$.tipoS}

localizacionP → **ε** {localizacionP.dirS = localizacionP.dirH
localizacionP.tipoS = localizacionP.tipoH}

NOTA: En el esquema de traducción cuando es necesario ponemos primero el else de los ifs para que nos detengamos inmediatamente si se da un error. También los genCod se ponen de forma en la que la pila de ejecución los genere en el orden correcto.