**1. Especificación del léxico del lenguaje**

Especificación formal del léxico del lenguaje utilizando definiciones regulares.

**Definición de las categorías léxicas reconocidas por nuestro analizador.**

**Auxiliares**

LETRA -> ('a'..'z') | ('A'..'Z')

DIGITO-> '0'..'9'

DELIM\_PUNTO -> '.'

ENTERO -> ('1'|'2'|'3'|'4'|'5'|'6'|'7'|'8'|'9')(DIGITO)\*

REAL -> ENTERO (DELIM\_PUNTO) (DIGITO)+

TIPOREAL-> ('R'|'r')('E'|'e')('A'|'a')('L'|'l')

TIPOENT-> ('I'|'i')('N'|'n')('T'|'t')

OP\_IN -> ('I'|'i')('N'|'n')

OP\_OUT -> ('O'|'o')('U'|'u')('T'|'t')

ID -> (LETRA | '\_') (((LETRA) | ENTERO | '\_' ))\* ( '?')?

OP\_AS -> '='

OP\_IGUAL -> '=='

OP\_NOT -> '!'

OP\_COMP -> '!='

OP\_MENQ -> '<'

OP\_MAYQ -> '>'

OP\_MAYOI -> '>='

OP\_MENOI -> '<='

**Principales**

SALTODELINEA -> ('\n'|'\r'| "\r\n")

COMENTARIO -> @ ( ~('\n'|'\r') )\*

Whitespace -> (' ' | '\t')+

SEP -> '; '

DELIM\_PAREN\_A -> ' ('

DELIM\_PAREN\_C -> ') '

OP\_MAS -> '+ '

OP\_MENOS -> '-'

OP\_MOD -> '%'

OP\_MUL -> '\*'

OP\_DIV -> '/'

OP\_OR -> '||'

OP\_AND -> '&&'

FIN -> '$$'

**7. Acondicionamiento de las gramáticas de atributos**

Transformaciones realizadas sobre las gramáticas de atributos para permitir la traducción predictivo-recursiva.

Únicamente deben incluirse la transformación de las producciones que se ven afectadas. Si alguna de las gramáticas no necesitan acondicionamiento, dejar el correspondiente subapartado en blanco.

**7.1. Acondicionamiento de la Gramática para la Construcción de la tabla de símbolos**

**Producción original: DECS** -> **DEC** sep **DECS**

**DECS** -> λ

**Prod. tras acondicionamiento: DECS -> DEC RDECS**

**RDECS ->** sep **RRDECS**

**RRDECS -> DECS**

**DECS** -> λ

**7.2. Acondicionamiento de la Gramática para la Comprobación de las Restricciones Contextuales**

**Producción original: ACCS** -> **ACC** sep **ACCS**

**ACCS** -> **ACC**

**Prods. tras acondicionamiento: ACCS -> ACC RACS**

**RACS** -> sep **RRACCS**

**RRACCS -> ACCS |** λ

**Producción original: ACCCOMP** -> **ACCADIT** op\_comp **ACCADIT**

**Prods. tras acondicionamiento: ACCCOMP** ->  **ACCADIT RACCCOMP**

**RACCCOMP ->**  op\_comp **ACCADIT**

**RACCCOMP ->**  λ

**Producción original: ACCADIT** -> **ACCADIT** op\_adit **ACCMULT**

**Prod. tras acondicionamiento: ACCADIT** -> **ACCMULT** **RACCADIT**

**RACCADIT ->** op\_adit **ACCMULT RACCADIT**

**RACCADIT ->** λ

**Producción original: ACCMULT** -> **ACCMULT** op\_mult **ACCUN**

**Prod. tras acondicionamiento: ACCMULT** -> **ACCUN RACCMULT**

**RACCMULT ->** op\_mult **ACCUN RACCMULT**

**RACCMULT ->** λ

**7.3. Acondicionamiento de la Gramática para la Traducción**

**Producción original: ACCS** -> **ACC** sep **ACCS**

**ACCS** -> **ACC**

**Prods. tras acondicionamiento: ACCS -> ACC RACS**

**RACS** -> sep **RRACCS**

**RRACCS -> ACCS |** λ

**Producción original: ACCCOMP** -> **ACCADIT** op\_comp **ACCADIT**

**Prods. tras acondicionamiento: ACCCOMP** ->  **ACCADIT RACCCOMP**

**RACCCOMP ->**  op\_comp **ACCADIT**

**RACCCOMP ->**  λ

**Producción original: ACCADIT** -> **ACCADIT** op\_adit **ACCMULT**

**Prod. tras acondicionamiento: ACCADIT** -> **ACCMULT** **RACCADIT**

**RACCADIT ->** op\_adit **ACCMULT RACCADIT**

**RACCADIT ->** λ

**Producción original: ACCMULT** -> **ACCMULT** op\_mult **ACCUN**

**Prod. tras acondicionamiento: ACCMULT** -> **ACCUN RACCMULT**

**RACCMULT ->** op\_mult **ACCUN RACCMULT**

**RACCMULT ->** λ

**8. Esquema de traducción orientado a las gramáticas de atributos**

Esquema de traducción en el que se muestre, mediante acciones semánticas, los lugares en los que deben evaluarse las distintas ecuaciones contempladas en todas las gramáticas de atributos: la de construcción de la tabla de símbolos, la de comprobación de las restricciones contextuales y la de especificación de la traducción.

El recorrido del árbol será el realizado por un analizador predictivo-recursivo.

**SPROG -> PROG** fin

SPROG.cod := ACCS.cod

SPROG.tipo := ACCS.tipo

SPROG.err := ACCS.err

**PROG -> DECS ACCS**

Accs.TSh=decs.TSs

SPROG.cod := ACCS.cod

SPROG.tipo := ACCS.tipo

SPROG.err := ACCS.err

**DECS -> DEC RDECS**

DECS0.TSs:=añadeID(DECS1.TSs,id:DEC.id,tipo:DEC.tipo,dir, DECS1.TSs.dir)

DECS0.TSs.dir:= +1

**RDECS ->** sep **RRDECS**

RDECS.TSs := RRDECS.TSs

**RRDECS -> DECS**

RRDECS.TSs := DECS.TSs

**DECS** -> λ

DECS0.TS = nuevaTS()

**DEC** -> tipo id

DEC.nombre := id.lexema

DEC.tipo := TIPO.tipo

**ACCS -> ACC RACS**

RACS.TSh:= ACC.TSh := ACCS.TSh

ACCS.cod=ACC.cod++RACS.cod

ACCS.err=ACC.err++RACS.err

**ACCS** -> **ACC**

ACC.TSh := ACCS.TSh

ACCS.cod=ACC.cod

ACCS.err=ACC.err

**RACS** -> sep **RRACCS**

RRACS.TSh:= RACS.TSh

RACS.cod**=**RRACCS.cod

RACS.err= RRACCS.err

**RRACCS -> ACCS**

ACC.TSh := RRACCS.TSh

RRACCS.cod=ACCS.TSh

RRACCS.err=ACCS.TSh

**ACC** -> **IN**

IN.TSh := ACC.TSh

ACCS.cod := IN.cod

ACCS.err := IN.err

**ACC** -> **OUT**

OUT.TSh := ACC0.TSh

ACCS.cod := OUT.cod

ACCS.err := OUT.err

**ACC** -> **EXP**

EXP.TSh := ACC0.TSh

ACCS.cod := EXP.cod

ACCS.err := EXP.err

**IN** -> op\_in id

IN.tipo=si (isID(id.lexema)) IN.TSh.getTipo(id.lexema)

si no ”error”

IN.cod=si (isID(id.lexema)) IN.cod+”código de traducción”

IN.err=si(not(isID(id.lexema))) ”mensaje de error”

**OUT** -> op\_out **EXP**

EXP.TSh := OUT.TSh

OUT.cod=EXP.cod++”código de traducción”

OUT.tipo=EXP.tipo

OUT.err=EXP.err

**EXP** -> **ACCCOMP**

ACCCOMP.TSh :=EXP.TSh

EXP.cod= ACCCOMP.cod

EXP.tipo= ACCCOMP.tipo

EXP.err= ACCCOMP.err

**EXP** -> id op\_as **ACCASIGN**

ACCASIGN.TSh :=EXP.TSh

EXP.tipo= si (isID(id.lexema)) EXP.TSh.tipoResultante(EXP.TSh.getTipo(id.lexema),ACCASIGN.tipo,2,’=’)

si no ”error”

EXP.cod=si (isID(id.lexema)) EXP.cod+”código de traduccion”

EXP.err=si (EXP.tipo==”error”) “mensaje de error”

**ACCASIGN** -> **EXP**

EXP.TSh := ACCASIGN.TSh

EXP.cod:= ACCASIGN.cod

EXP.tipo:= ACCASIGN.tipo

EXP.err:= ACCASIGN.err

**ACCCOMP** ->  **ACCADIT RACCCOMP**

ACCADIT.TSh := RACCCOMP.TSh := ACCCOPM.TSh

RACCCOMP.codh := ACCADIT.cod

RACCCOMP.tipoh := ACCADIT.tipo

RACCCOMP.errh := ACCADIT.err

ACCCOMP.cod := RACCCOMP.cod

ACCCOMP.tipo := RACCCOMP.tipo

ACCCOMP.err := RACCCOMP.err

**RACCCOMP ->**  op\_comp **ACCADIT**

ACCADIT.TSh := RACCCOMP.TSh

RACCCOMP.cod := RACCCOMP.codh ++ ACCADIT.cod ++ “código de traducción”

RACCCOMP.tipo:= TBh.tipoResultante ( RACCCOMP.tipoh, ACCADIT.tipo,2,op\_comp)

RACCCOMP.err := si (RACCCOMP.tipo==”error”) RACCCOMP.errh ++ “mensaje de error”

**RACCCOMP ->**  λ

RACCCOMP.cod = RACCCOMP.codh

RACCCOMP.tipo = RACCCOMP.codh

RACCCOMP.err = RACCCOMP.codh

**ACCADIT** -> **ACCMULT** **RACCADIT**

ACCMULT.TSh := RACCADIT.TSh := ACCADIT.TSh

RACCADIT.tipoh := ACCMULT.tipo

RACCADIT.codh := ACCMULT.cod

RACCADIT.errh := ACCMULT.err

ACCADIT.err := RACCADIT.err

ACCADIT.tipo := RACCADIT.tipo

ACCADIT.cod := RACCADIT.cod

**RACCADIT ->** op\_adit **ACCMULT RACCADIT**

RACCADIT1.TSh := ACCMULT.TSh := RACCADIT0.TSh

RACCADIT0.cod := RACCADIT0.codh ++ ACCMULT.cod ++ “mensaje de error”

RACCADIT0.tipo := RACCADIT0.TSh.TBh.tipoResultante( RACCADIT0.tipoh,ACCMULT.tipo,2,op\_mult)

RACCADIT0.err := si (RACCADIT0.tipo=”error”) RACCADIT0.errh ++ ACCMULT.err ++ “mensaje de error”

RACCADIT1.codh := RACCADIT0.cod

RACCADIT1.tipoh := RACCADIT0.tipo

RACCADIT1.errh := RACCADIT0.err

**RACCADIT ->** λ

RACCADIT.cod := RACCADIT.codh

RACCADIT.tipo := RACCADIT.tipoh

RACCADIT.err := RACCADIT.errh

**ACCMULT** -> **ACCUN RACCMULT**

ACCUN.TSh := RACCMULT.TSh := ACCMULT.TSh

RACCMULT.errh := ACCUN.err

RACCMULT.tipoh := ACCUN.tipo

RACCMULT.codh := ACCUN.cod

ACCMULT.err := RACCMULT.err

ACCMULT.tipo := RACCMULT.tipo

ACCMULT.cod := RACCMULT.cod

**RACCMULT ->** op\_mult **ACCUN RACCMULT**

RACCMULT1.TSh := ACCUN.TSh := RACCMULT0.TSh

RACCMULT0.cod := RACCMULT0.codh ++ ACCUN.cod ++ “mensaje de error”

RACCMULT0.tipo := RACCMULT0.TSh.TBh.tipoResultante( RACCMULT0.tipoh,ACCUN.tipo,2,op\_mult)

RACCMULT0.err := si (RACCMULT0.tipo=”error”) RACCMULT0.errh ++ ACCUN.err ++ “mensaje de error”

RACCMULT1.codh := RACCMULT0.cod

RACCMULT1.tipoh := RACCMULT0.tipo

RACCMULT1.errh := RACCMULT0.err

**RACCMULT ->** λ

RACCMULT.cod := RACCMULT.cod

RACCMULT.tipo := RACCMULT.tipo

RACCMULT.err := RACCMULT.err

**ACCUN -> FACTOR**

ACCUN.cod := FACTOR.cod

ACCUN.tipo := FACTOR.tipo

ACCUN.err := FACTOR.err

**ACCUN ->** op\_un **FACTOR**

ACCUN.cod := FACTOR.cod ++ “código de traducción”

ACCUN.tipo := TBh.tipoResultante(FACTOR.cod,””,1,op\_un)

ACCUN.err := si (ACCUN.tipo = “error”) FACTOR.err ++ “mensaje de error”

**ACCUN ->** delim\_paren\_a **TIPO** delim\_paren\_c **FACTOR**

ACCS.cod := FACTOR.cod ++ “código de traducción”

ACCS.tipo := TIPO.tipo

ACCS.err := FACTOR.err

**FACTOR -> NUM**

ACCS.cod := NUM.cod

ACCS.tipo := NUM.tipo

ACCS.err := NUM.err

**FACTOR ->** id

FACTOR.cod := si (FACTOR.TSh.isID(id.lexema)) “código de traducción”

FACTOR.tipo := si (not(FACTOR.TSh.isID(id.lexema))) “error”

FACTOR.err := si (FACTOR.err = ”error”) “mensaje de error”

**FACTOR ->** delim\_paren\_a **ACC** delim\_paren\_c

FACTOR.cod := ACC.cod

FACTOR.tipo := ACC.tipo

FACTOR.err :=ACC.err

**NUM ->** real

NUM.cod := “codigo de traducción”

NUM.tipo := “real”

NUM.err := ” ”

**NUM ->** entero

NUM.cod := “codigo de traducción”

NUM.tipo := “entero”

NUM.err := ” ”

**TIPO ->** tiporeal

TIPO.cod := “”

TIPO.tipo := “real”

TIPO.err := “”

**TIPO ->** tipoent

TIPO.cod := “”

TIPO.tipo := “entero”

TIPO.err := “”

**9. Esquema de traducción orientado al traductor predictivo – recursivo**

Esquema de traducción en el que se haga explícito los parámetros utilizados para representar los atributos, así como en los que se muestre la implementación de las ecuaciones semánticas como asignaciones a dichos parámetros.

**9.1. Variables globales**

Descripción y propósito de las variables globales usadas.

Si no se usan variables globales, dejar este apartado en blanco.

**ErrDecs:** Se utiliza la variable global de tipo String errDecs en el analizador sintáctico MiParser, utilizada para concatenar la sucesión de posibles errores semánticos en la sección de declaraciones. La restricción contextual que establece la imposibilidad de declarar dos veces el mismo identificador, es la única restricción que puede producir concatenaciones a dicha variable.

**9.2. Nuevas operaciones y transformación de ecuaciones semánticas**

En caso de introducir nuevas operaciones (por ejemplo: el procedimiento “emite”), deben describirse aquí.

Debe describirse, así mismo, qué ecuaciones semánticas se ven transformadas y cómo (por ejemplo: la generación de código se implementa emitiendo la última instrucción).

**9.3. Esquema de traducción**

**SPROG(out err, out cod, out tipo) -> PROG(out err, out cod, out tipo)** finSPROG.cod := ACCS.cod

SPROG.tipo := ACCS.tipo

SPROG.err := ACCS.err

**PROG(out err, out cod, out tipo) -> DECS() ACCS(out err, out cod, out tipo)**

Accs.TSh=decs.TSs

SPROG.cod := ACCS.cod

SPROG.tipo := ACCS.tipo

SPROG.err := ACCS.err

**DECS(out TS) -> DEC(out nombre, out tipo) RDECS(out TS)**

DECS0.TSs := añadeID(DECS1.TSs,id:DEC.id,tipo:DEC.tipo,dir, DECS1.TSs.dir)

DECS0.TSs.dir := +1

**RDECS(out TS) ->** sep **RRDECS(out TS)**

RDECS.TSs := RRDECS.TSs

**RRDECS(TS) -> DECS(out TS)**

RRDECS.TSs := DECS.TSs

**DECS(out TS)** -> λ

DECS0.TS = nuevaTS()

**DEC(out nombre, out tipo)** -> TIPO Reconoce(id)

DEC.nombre := id.lexema

DEC.tipo := TIPO.tipo

**ACCS(out err, out cod, out tipo, in TS) -> ACC(out err, out cod, out tipo, in TS) RACS(out err, out cod, out tipo, in TS)**

RACS.TSh:= ACC.TSh := ACCS.TSh

ACCS.cod=ACC.cod++RACS.cod

ACCS.err=ACC.err++RACS.err

**ACCS(out err, out cod, out tipo, in TS)** -> **ACC(out err, out cod, out tipo, in TS)**

ACC.TSh := ACCS.TSh

ACCS.cod=ACC.cod

ACCS.err=ACC.err

ACCS.tipo=ACC.tipo

**RACS(out err, out cod, out tipo, in TS)** -> Reconoce(sep) **RRACCS(out err, out cod, out tipo, in TS)**

RRACS.TSh:= RACS.TSh

RACS.cod**=**RRACCS.cod

RACS.err= RRACCS.err

**RRACCS(out err, out cod, out tipo, in TS) -> ACCS(out err, out cod, out tipo, in TS)**

ACC.TSh := RRACCS.TSh

RRACCS.cod=ACCS.TSh

RRACCS.err=ACCS.TSh

**ACC(out err, out cod, out tipo, in TS)** -> **IN(out err, out cod, out tipo, in TS)**

IN.TSh := ACC.TSh

ACCS.cod := IN.cod

ACCS.err := IN.err

**ACC(out err, out cod, out tipo, in TS)**  -> **OUT(out err, out cod, out tipo, in TS)**

OUT.TSh := ACC0.TSh

ACCS.cod := OUT.cod

ACCS.err := OUT.err

**ACC(out err, out cod, out tipo, in TS)** -> **EXP(out err, out cod, out tipo, in TS)**

EXP.TSh := ACC0.TSh

ACCS.cod := EXP.cod

ACCS.err := EXP.err

**IN(out err, out cod, out tipo, in TS)** -> Reconoce(op\_in) Reconoce(id)

IN.tipo=si (isID(id.lexema)) IN.TSh.getTipo(id.lexema)

si no ”error”

IN.cod=si (isID(id.lexema)) IN.cod+”código de traducción”

IN.err=si(not(isID(id.lexema))) ”mensaje de error”

**OUT(out err, out cod, out tipo, in TS)** -> Reconoce(op\_out) **EXP(out err, out cod, out tipo, in TS)**

EXP.TSh := OUT.TSh

OUT.cod=EXP.cod++”código de traducción”

OUT.tipo=EXP.tipo

OUT.err=EXP.err

**EXP(out err, out cod, out tipo, in TS)** -> **ACCCOMP(out err, out cod, out tipo, in TS)**

ACCCOMP.TSh :=EXP.TSh

EXP.cod= ACCCOMP.cod

EXP.tipo= ACCCOMP.tipo

EXP.err= ACCCOMP.err

**EXP(out err, out cod, out tipo, in TS)** -> Reconoce(id) Reconoce(op\_as) **ACCASIGN(out err, out cod, out tipo, in TS)**

ACCASIGN.TSh :=EXP.TSh

EXP.tipo= si (isID(id.lexema)) EXP.TSh.tipoResultante(EXP.TSh.getTipo(id.lexema),ACCASIGN.tipo,2,’=’)

si no ”error”

EXP.cod=si (isID(id.lexema)) EXP.cod+”código de traduccion”

EXP.err=si (EXP.tipo==”error”) “mensaje de error”

**ACCASIGN(out err, out cod, out tipo, in TS)** -> **EXP(out err, out cod, out tipo, in TS)**

EXP.TSh := ACCASIGN.TSh

EXP.cod:= ACCASIGN.cod

EXP.tipo:= ACCASIGN.tipo

EXP.err:= ACCASIGN.err

**ACCCOMP(out err, out cod, out tipo, in TS)** ->  **ACCADIT(out err, out cod, out tipo, in TS) RACCCOMP(out err, out cod, out tipo, in TS)**

ACCADIT.TSh := RACCCOMP.TSh := ACCCOPM.TSh

RACCCOMP.codh := ACCADIT.cod

RACCCOMP.tipoh := ACCADIT.tipo

RACCCOMP.errh := ACCADIT.err

ACCCOMP.cod := RACCCOMP.cod

ACCCOMP.tipo := RACCCOMP.tipo

ACCCOMP.err := RACCCOMP.err

**RACCCOMP(out err, out cod, out tipo, in TS, in err, in cod, in tipo) ->**  Reconoce(op\_comp) **ACCADIT(out err, out cod, out tipo, in TS)**

ACCADIT.TSh := RACCCOMP.TSh

RACCCOMP.cod := RACCCOMP.codh ++ ACCADIT.cod ++ “código de traducción”

RACCCOMP.tipo:= TBh.tipoResultante ( RACCCOMP.tipoh, ACCADIT.tipo,2,op\_comp)

RACCCOMP.err := si (RACCCOMP.tipo==”error”) RACCCOMP.errh ++ “mensaje de error”

**RACCCOMP(out err, out cod, out tipo, in TS, in err, in cod, in tipo) ->**  λ

RACCCOMP.cod = RACCCOMP.codh

RACCCOMP.tipo = RACCCOMP.codh

RACCCOMP.err = RACCCOMP.codh

**ACCADIT(out err, out cod, out tipo, in TS)** -> **ACCMULT(out err, out cod, out tipo, in TS)** **RACCADIT(out err, out cod, out tipo, in TS)**

ACCMULT.TSh := RACCADIT.TSh := ACCADIT.TSh

RACCADIT.tipoh := ACCMULT.tipo

RACCADIT.codh := ACCMULT.cod

RACCADIT.errh := ACCMULT.err

ACCADIT.err := RACCADIT.err

ACCADIT.tipo := RACCADIT.tipo

ACCADIT.cod := RACCADIT.cod

**RACCADIT(out err, out cod, out tipo, in TS, in err, in cod, in tipo) ->** Reconoce(op\_adit) **ACCMULT(out err, out cod, out tipo, in TS) RACCADIT(out err, out cod, out tipo, in TS)**

RACCADIT1.TSh := ACCMULT.TSh := RACCADIT0.TSh

RACCADIT0.cod := RACCADIT0.codh ++ ACCMULT.cod ++ “mensaje de error”

RACCADIT0.tipo := RACCADIT0.TSh.TBh.tipoResultante( RACCADIT0.tipoh,ACCMULT.tipo,2,op\_mult)

RACCADIT0.err := si (RACCADIT0.tipo=”error”) RACCADIT0.errh ++ ACCMULT.err ++ “mensaje de error”

RACCADIT1.codh := RACCADIT0.cod

RACCADIT1.tipoh := RACCADIT0.tipo

RACCADIT1.errh := RACCADIT0.err

**RACCADIT(out err, out cod, out tipo, in TS, in err, in cod, in tipo) ->** λ

RACCADIT.cod := RACCADIT.codh

RACCADIT.tipo := RACCADIT.tipoh

RACCADIT.err := RACCADIT.errh

**ACCMULT(out err, out cod, out tipo, in TS)** -> **ACCUN(out err, out cod, out tipo, in TS) RACCMULT(out err, out cod, out tipo, in TS)**

ACCUN.TSh := RACCMULT.TSh := ACCMULT.TSh

RACCMULT.errh := ACCUN.err

RACCMULT.tipoh := ACCUN.tipo

RACCMULT.codh := ACCUN.cod

ACCMULT.err := RACCMULT.err

ACCMULT.tipo := RACCMULT.tipo

ACCMULT.cod := RACCMULT.cod

**RACCMULT(out err, out cod, out tipo, in TS, in err, in cod, in tipo,) ->** Reconoce(op\_mult) **ACCUN(out err, out cod, out tipo, in TS) RACCMULT(out err, out cod, out tipo, in TS)**

RACCMULT1.TSh := ACCUN.TSh := RACCMULT0.TSh

RACCMULT0.cod := RACCMULT0.codh ++ ACCUN.cod ++ “mensaje de error”

RACCMULT0.tipo := RACCMULT0.TSh.TBh.tipoResultante( RACCMULT0.tipoh,ACCUN.tipo,2,op\_mult)

RACCMULT0.err := si (RACCMULT0.tipo=”error”) RACCMULT0.errh ++ ACCUN.err ++ “mensaje de error”

RACCMULT1.codh := RACCMULT0.cod

RACCMULT1.tipoh := RACCMULT0.tipo

RACCMULT1.errh := RACCMULT0.err

**RACCMULT(out err, out cod, out tipo, in TS, in err, in cod, in tipo) ->** λ

RACCMULT.cod := RACCMULT.cod

RACCMULT.tipo := RACCMULT.tipo

RACCMULT.err := RACCMULT.err

**ACCUN(out err, out cod, out tipo, in TS) -> FACTOR(out err, out cod, out tipo, in TS)**

ACCUN.cod := FACTOR.cod

ACCUN.tipo := FACTOR.tipo

ACCUN.err := FACTOR.err

**ACCUN(out err, out cod, out tipo, in TS) ->** Reconoce(op\_un) **FACTOR(out err, out cod, out tipo, in TS)**

ACCUN.cod := FACTOR.cod ++ “código de traducción”

ACCUN.tipo := TBh.tipoResultante(FACTOR.cod,””,1,op\_un)

ACCUN.err := si (ACCUN.tipo = “error”) FACTOR.err ++ “mensaje de error”

**ACCUN(out err, out cod, out tipo, in TS) ->** Reconoce(delim\_paren\_a) **TIPO** Reconoce(delim\_paren\_c) **FACTOR(out err, out cod, out tipo, in TS)**

ACCS.cod := FACTOR.cod ++ “código de traducción”

ACCS.tipo := TIPO.tipo

ACCS.err := FACTOR.err

**FACTOR(out err, out cod, out tipo, in TS) -> NUM(out err, out cod, out tipo, in TS)**

ACCS.cod := NUM.cod

ACCS.tipo := NUM.tipo

ACCS.err := NUM.err

**FACTOR(out err, out cod, out tipo, in TS) ->** Reconoce(id)

FACTOR.cod := si (FACTOR.TSh.isID(id.lexema)) “código de traducción”

FACTOR.tipo := si (not(FACTOR.TSh.isID(id.lexema))) “error”

FACTOR.err := si (FACTOR.err = ”error”) “mensaje de error”

**FACTOR(out err, out cod, out tipo, in TS) ->** Reconoce(delim\_paren\_a) **ACC(out err, out cod, out tipo, in TS)** Reconoce(delim\_paren\_c)

FACTOR.cod := ACC.cod

FACTOR.tipo := ACC.tipo

FACTOR.err :=ACC.err

**NUM(out err, out cod, out tipo, in TS) ->** Reconoce(real)

NUM.cod := “codigo de traducción”

NUM.tipo := “real”

NUM.err := ” ”

**NUM(out err, out cod, out tipo, in TS) ->** Reconoce(entero)

NUM.cod := “codigo de traducción”

NUM.tipo := “entero”

NUM.err := ” ”

**TIPO(out err, out cod, out tipo) ->** Reconoce(tiporeal)

TIPO.cod := “”

TIPO.tipo := “real”

TIPO.err := “”

**TIPO(out err, out cod, out tipo) ->** Reconoce(tipoent)

TIPO.cod := “”

TIPO.tipo := “entero”

TIPO.err := “”

**11 Notas sobre la Implementación**

Descripción de la implementación realizada.

**11.1. Descripción de archivos**

Enumeración de los archivos con el código fuente de la implementación, y descripción de lo que contiene cada archivo.

El código en sí deberá estar adecuadamente comentado, pero NO se incluirá en la memoria de la práctica. Aquí sólo debe describirse someramente qué contiene cada archivo.

**Traductor.java**

Dicha clase contiene todos los atributos sintetizados que desde las hojas del árbol se propagan hasta la raíz. La clase contiene los siguientes atributos de tipo String:

-*Tipo*: contiene el tipo sintetizado hasta el momento para esa producción, es decir, el tipo evaluado desde las hojas del subárbol cuya raíz es la producción en cuestión. Podemos encontrarlo evaluado a “entero”, “real” y “error” respectivamente, para expresiones evaluadas a tipo entero, real u operaciones entre tipos erróneos.

-*Err*: contiene la concatenación de los mensajes de error para los defectos semánticos que se localicen en dicho análisis. Cada mensaje se encuentra en una línea del String, identificado por los números de fila y columna del fichero de entrada donde se produjo; incorporando un mensaje explicativo del error.

-*Cod:* contiene las instrucciones del lenguaje objeto que se van sintetizando a lo largo del análisis de las restricciones contextuales. Cada línea contiene una instrucción del código P.

Cabe destacar que dicha clase contiene, también, atributos heredados desde la perspectiva de las producciones RACCMULT, RACCADIT y RACCOMP; introducidos con motivo de la factorización necesaria para acondicionar la gramática de partida.