Esquemas de traducción (2º Cuatrimestre)

Nuevas operaciones introducidas:

- **parchea**: Copia el valor entero en n2 a n1, que ha sido usado con anterioridad en otra instrucción pero su valor no era conocido.
  parchea (n1: Entero, n2: Entero)

- **emiteCodigo**: Concatena la variable global código con la variable de tipo String pasada por parámetro.
  emiteCodigo (cod:String)

Prog () ::=
{var dirs, etqs}
Cabecera(out dir)
{emiteCodigo(inicio(nivel, ?));
dirs←dir;
etq←etq+ longInicio;
emiteCodigo(ir_a(?));
etqs←etq
etq←etq+1;
dirh ←dir
}
Decs(in dirh, out dir)
{parchea(dirs,dir);
parchea(etqs, etq);
}
Bloque()
{emiteCodigo(stop);}

Cabecera(out direccion) ::=
PROGRAM
id (out lex)
PYCOMA
{añadeID(creaTS(),lex, <valor:0, tipo: tipoError>);
dir ← 0;
nivel ← 0;
etq ← 0;
err ← false;
cod ← λ;}

Decs(in dirh, out dir) ::=
DTipos()
{dirh1 ←dirh;}
RDecs(in dirh1, out dir1)
{dir ←dir1;}

RDecs(in dirh, out dir) ::= λ
{dir ←dirh;}

RDecs(in dirh, out dir) ::=
{dir1 ←dirh;}
Procs(in dir1, out dir2)
{dir ←dir2;}

RDecs(in dirh, out dir) ::=
{dir1 ←dirh;}
Vars(in dir1, out dir2)
{dir3 ←dir2;}
RDecs2(in dir3, out dir4)
{dir←dir4;}

RDecs2(in dirh, out dir) ::=
{dir1 ←dirh;}
Procs(in dir1, out dir2)
{dir ←dir2;}

RDecs2(in dirh, out dir) ::= λ
{dir ←dirh;}

Decs(in dirh, out dir) ::=
{dir1 ←dirh;}
Vars(in dir1, out dir2)
{dir3 ←dir2;}
RDecs3(in dir3, out dir4)
{dir←dir4;}

RDecs3(in dirh, out dir) ::=
{dir1 ←dirh;}
Procs(in dir1, out dir2)
{dir ←dir2;}

RDecs3(in dirh, out dir) ::= λ
{dir ←dirh;}

Decs(in dirh, out dir) ::= λ
{dir ←dirh;}

DTipos()::=
SECTIPOS
RTipos()

RTipos()::=
RTipos2()
RTipos()

RTipos()::=λ

RTipos2 ::=
id(out lex)
IGUAL
Tipo(out tipo)
PYCOMA
{err←err v (existeID(TS, lex) ^ (TS[lex].nivel = nivel)
TS ←añadeID(TS, lex, <tipo: tipo.t, clase: tipo, tam: tipo.tam, nivel: nivel>);}

Tipo(out tipo) ::=
TIPENT
{tipo ← <t: Entero, tam: 1>;}

Tipo(out tipo) ::=
TIPBOOL
{tipo ← <t: Boolean, tam: 1>;}

Tipo(out tipo) ::=
Id (out lex)
{tipo ←<t: ref, id: lex, tam: TS[lex].tipo.tam>;
err ←err v if existeID(TS, lex) then (TS[lex].clase ≠ tipo) else true;}

Tipo(out tipo) ::=
TPUNTERO
Tipo(out tipo1)
{tipo ←<t: Puntero, tBase: tipo1, tam: 1>;^Tipo(out tipo1);}

Tipo(out tipo) ::= TARRAY [0..numero] of Tipo
TARRAY
[
0
..
numero (out lex)
]
of
Tipo (out tipo1)
{tipo ←<t: Array, numElems: valorDe(lex), tBase: tipo1, tam:
(valorDe(lex)+1)* tipo1.tam>;
err ←err v Referencia(tipo1, TS);}

Vars (in dirh, out dir) ::=
VAR
{dir1 ← dirh;}
Tvar2 (in dir1, out dir2)
{dir ← dir2; }

Tvar2 (in dirh, out dir) ::= λ
{dir ← dirh;}

Tvar2 (in dirh, out dir,lex)::=
id(out lex)
{dir1 ← dirh;
lex1←lex;}
RTvar2(in dir1,lex , out dir2)
{dir ← dir2;}

RTvar2 (in dirh, out dir) ::=
COMA
{dir1 ← dirh;}
RTvar2(in dir1, out dir2)
{dir ← dir2;}

RTvar2 (in dirh,lex, out dir) ::=
2PUNTOS
Tipos(out tip)
PYCOMA
{ err←err v existeID(TS, lex)^TS[lex].nivel= nivel;
añadeID(TS,lex,<direccion: dirh, tipo:tip, clase: variable, tam:tip.tam, nivel:nivel,
modo: error>);
dir← dirh + tip.tam
}

Tipos(out tipo) ::=
TIPENT
{tipo ← <t: Entero, tam: 1>;}

Tipos(out tipo) ::=
TIPBOOL
{tipo ← <t: Boolean, tam: 1>;}

Tipos(out tipo) ::=
Id (out lex)
{tipo ← <t: ref, id: lex, tam: TS[lex].tipo.tam>;
err ← err v if existeID(TS, lex) then (TS[lex].clase ≠ tipo) else true;}

Procs (in dirh, out dir) ::=
TDProc (out dir1)
{dirh1 ← dirh + dir1; }
Procs (in dirh1, out dir2)
{dir ← dir2; }

Procs (in dirh, out dir) ::= λ
{dir ← dirh;}

TDProc (out dir) ::=
PROC
id (out lex)
Params (out params, dir1)
PYCOMA
{var ini1, ini2, TS1;
TS1 ← TS;
TS ← creaTS (TS);
err ← err v existeID(TS, lex) ^ TS[lex].nivel = nivel;
TS ← añadeID (TS, lex, <clase: proc, tipo:<t: proc, params: params>, nivel: nivel, inicio: ? >);
ini1 ← inicio;}
nivel ← nivel+1;
TS ← añadeID(TS, lex, <clase: proc, tipo:<t: proc, params: params>, nivel: nivel, inicio: ini2>);
ini2 ← inicio;
dir2 ← dir1;}
BloqProc (in dir2, out inicio, dir3)
{parchea(ini1, inicio);
parchea(ini2, inicio);
dir ← direccion3;
TS ← TS1;}

Params(out params, out dir) ::=
PA
ListaParams(out params1, dir1)
PC
{params←params1;
dir←dir1;
}

Params(out params, out dir) ::= λ
{params←[];
dir←0;
}

ListaParams(out params, out dir)::=
{dir1←0;}
Params2(in dir1, out params1, out dir2)
{dir3←dir2;}
RListaParams(in dir3, out params2, out dir4)
{params←[params1]++params2;
dir←dir4;
}

RListaParams(in dirh, out params, out dir)::=
PYCOMA
{dir1←dirh;}
ListaParams(in dir1, out params1, out dir2)
{params←params1;
dir←dir2;
}

RListaParams(in dirh, out params, out dir)::= λ
{params←[];
dir←dirh;
}

ListaParams (in dirh, out params, dir) ::=
TVAR
{modo ←variable;
dir1 ←dirh;}
RListaParams2(in modo, dir1, out tipo, dir2)
{dir ←dir2;}

RListaParams2(in modo, dirh, out tipo, dir) ::=
{modo1 ←modo;
dir1 ←dirh;}
Params2(in modo, dir1, out tipo, dir2)
{dir ←dir2;}

RListaParams2(in modo, dirh, out tipo, dir) ::=
{modo1 ←modo;
dir1 ←dirh;}
Params2(in modo1, dir1, out tipo, dir2)
PYCOMA
{dir3 ←dir2;}
ListaParams(in dir3, out params, dir)
{dir ←dir3;}

Params2(in modo, dirh, out tipo, dir) ::=
Id(out lex)
{modo1 ←modo;
dir1 ←dirh;}
RParams2(in modo1, dir1, out tipo1, dir1)
{dir ←dir1 + tipo1.tam;
err ←err v existeID(TS, lex) ^ TS[lex].nivel = nivel;
TS ←añadeID (TS, lex, <direccion: dir, tipo: tipo1, clase: if modo1 = variable then pVariable else valor, tam: if clase = pVariable then 1 else tipo1.tam, nivel: nivel>);
tipo ←tipo1;}

RParams2 (in modo, dirh, out tipo, dir) ::=
COMA
{modo1 ←modo;

dir1 ←dirh;}
Params2(in modo1, dir1, out tipo, dir2)
{dir ←dir2;}

RParams2 (in modo, dirh, out tipo, dir)::=
2PUNTOS
{dir1 ←dirh;}
Tipos(out tip)
{tipo ←tip;
dir ←dir1;}

BloqProc(in dirh, out ini, out dir)::=
{dirh1 ←dirh;}
Decs2(in dirh1, out dir1)
{inicio ←etq;
emiteCodigo(prologo(nivel, dir1));
etq ←etq + longPrologo;
}
Bloque()
{emiteCodigo(epilogo(nivel));
emiteCodigo(ir_ind);
etq ←etq+long Epilogo+1;
dir ←dir1;}

Bloque() ::=
INICIO
TBloque2()
FIN

TBloque2() ::=
Tsentencia()
TBloque2()

TBloque2() ::= λ

Tsentencia() ::=
TAsig()

TSentencia ::=
TRead()

TSentencia ::=
TWrite()

TSentencia ::=
TNPunt()

TSentencia ::=
TLiberar()

TSentencia ::=
TLlamadaProc()

TSentencia ::=
TIf()

TSentencia ::=
TWhile()

TAsig() ::=
Descriptor(out tipo)
ASIG
{parh←false;}
Exp (in parh, out tipo2, modo2)
{err← err v ¬compatible(tipo, tipo2,TS);
If compatible(tipo, <t:Integer>, TS) v compatible(tipo, <t:Boolean>, TS) then
emiteCodigo(desapilaIndice()) else mueve(tipo.tam);
etq ← etq+1; }

Descriptor(out tipo)::=
Descriptor2(out tipo1)
{tipo ←tipo1;}

Descriptor2(out tipo)::=
Descriptor2(out tipo1)
CA
{parh ← false}
Exp (in parh, out tipo2, modo2)
CC
{tipo ← if (tipo1.t = Array ^ tipo2.t = Integer) then referencia(tipo1.tBase, TS)
Else <t:tipErr>;
emiteCod(apila(tipo1.TBase.tam));
emiteCod(multiplica);
emiteCod(Suma);
etq ←etq+3
}

Descriptor2(out tipo)::=
id (out lex)
{tipo= if(existeID(TS, lex) ^ TS[lex].clase = variable) then
referencia(TS[lex].tip,TS) else <t:tipErr>;}

Descriptor2 (out tipo)::=
^
Descriptor2 (out tipo2)
{tipo = if tipo2.t= puntero then referencia(tipo2.tBase, TS) else <t:tipErr>;}

TRead() ::=
LEER
PA
id(out lex)
PC
PYCOMA
{ err ← ¬existeID(TS, lex) v (TS[lex].clase < > variable)
emiteCodigo(lecturaPantalla(TS[lex].dir));}

TWrite() ::=
ESCRIBIR
PA
id(out lex)
PC
PYCOMA
{err← ¬existeID(TS, lex) v ((TS[lex].clase < > variable) ^ ((TS[lex].tipo.t ==
Entero) v (TS[lex].tipo.t == boolean)))
emiteCodigo (escrituraPantalla(cimaPila()));}

TNPunt() ::=
NUEVO
PA
Id(out lex)
PC
{err ← ¬existeID(TS, lex) v (TS[lex].tipo.t < > puntero)
emiteCodigo(reservar(TS[lex].tam))
emiteCodigo(desapila_ind);
etq← etq+2;}

TLiberar()::=
LIBERAR
PA
Id(out lex)
PC
{err ← ¬existeID(TS, lex) v (TS[lex].tipo.t < > puntero)
emiteCodigo(liberar(TS[lex].tam))
etq← etq+1;}

TIf()::=
SI
PA
{parh ← false;}
Exp (in parh, out tipo, out modo)
PC
ENTONCES
INICIO
{emiteCodigo(ir_f(?));
etq←etq+1;
etq1 ←etq;}
TBloque2 ()

{emiteCodigo(ir_a(?));
etq←etq+1;
etq2 ←etq;
parchea(etq1,etq+1);}
FIN
RTif()
{parchea(etq2, etq+1);
err←err v (tipo.t<>Boolean); }

RTif::=
SINO
INICIO
TBloque2()
FIN

RTif::= λ

TWhile ::=
MIENTRAS
PA
{parh ← false;
etq2←etq}
Exp (in parh, out tipo, out modo)
{emiteCodigo(ir_f(?))
etq←etq+1;
etq1 ←etq;}
PC
HACER
INICIO
TBloque2()
FIN
{emiteCodigo(ir_a(?))
etq←etq+1;
etq2 ←etq;
parchea(etq1,etq+1);
parchea(etq,etq2);
err ← err v (tipo.t<>Boolean);
}

TLlamadaProc()::=
id (out lex)
{emiteCodigo(apilaDirRetorno(etq));
etq ← etq + longApilaRetorno;}
PA
Params3()
PC
PYCOMA
{err←err v ¬existeID(TS, lex) v TS[lex].clase<>proc;

emiteCodigo(ir_a(TS[lex].inicio));
etq ←eq+1;
}

Params3(in paramsh)::=
(
{paramsh1 ← paramsh;
emiteCodigo (inicioPaso);
etq ← etq + longInicioPaso;}
ListaParams3()
)
{err ← err v \|fparamsh\| ≠ nparams1;
emiteCodigo (longFinPaso);
etq ← etq + longFinPaso;}

Params3(in paramsh)::= λ
{err ← err v \|paramsh\| > 0;}

ListaParams3(in paramsh, out nparams)::=
{emiteCodigo(copia);
etq ←etq+1;
parh ← paramsh[1].modo=variable;}
Exp(in parh, out tipo, out modo)
{nparams1 ← 1;
paramsh1 ← paramsh;
emiteCodigo(pasoParametro(modo, paramsh[1]));
etq ← etq + longPasoParametro;}
RListaParams3(in nparams1, paramsh1)
{err ← err v \|paramsh\|= 0 v paramsh[1].modo = variable ^ modo = valor) v
¬compatibles(paramsh[1].tipo, tipo, TS);}

RListaParams3 (in nparams, paramsh)::=
COMA
{emiteCodigo (copia);
emiteCodigo (direccionPFormal(paramsh[nparams]));
etq ← etq + longPFormal + 1;
parh ← paramsh[nparams].modo = variable;}
Exp (in parh, out tipo, modo)
{nparams1 ← nparams +1;
paramsh1 ← paramsh;
emiteCodigo (pasoParametro (modo, paramsh[nparams])}
etq ← etq + longPasoParametro;}
RListaParams3 (in nparamsh1, fparamsh1)
{error ← error v nparamh > \|paramsh\| v (paramsh[1].modo = variable ^ modo
= valor) v ¬compatibles(paramsh[1].tipo, tipo, TS);}

RListaParams3()::=λ

Exp (in parh, out tipo, modo)::=
{parh1 ←parh;}
ExpSum (in parh1, out tipo1, modo1)
{tipoh2 ←tipo1;
modoh2 ←modo1;}
RExp (in tipoh2,modoh2, out tipo2, modo2)
{tipo ←tipo2
modo ←modo2}

RExp (in tipoh, modoh, out tipo, modo)::=
OpRel (out op)
{parh1 ←false;}
ExpSum (in parh1, out tipo1, modo1)
RExp (in tipoh2,modoh2, out tipo2, modo2)
{tipoh2 ←tipoRelacion(tipoh.t, tipo1.t);
tipo ←tipo2;
modo ←valor;
emiteCodigo(op);
etq = etq +1;}

RExp (in tipoh, modoh, out tipo, modo) ::= λ
{tipo ←tipoh;
modo ←modoh;}

ExpSum (in parh, out tipo, modo)::=
{parh1 ←parh;}
ExpProd (in parh1, out tipo1, modo1)
{tipoh2 ←tipo1;
modoh2 ←modo1;}
RExpSum (in tipoh2,modoh2, out tipo2, modo2)
{tipo ←tipo2
modo ←modo2}

RExpSum (in tipoh,modoh, out tipo, modo)::=
OpAd(out op)
{parh1 ←false;}
ExpProd (in parh1, out tipo1, modo1)
RExpSum (in tipoh2,modoh2, out tipo2, modo2)
{tipoh2 ←tipoNum(tipoh.t, tipo1.t);
tipo ←tipo2;
modo ←valor;
emiteCodigo(op);
etq = etq +1;}

RExpSum (in tipoh,modoh, out tipo, modo)::=
OR
{parh1 ←false;}
ExpProd (in parh1, out tipo1, modo1)
RExpSum (in tipoh2,modoh2, out tipo2, modo2)

{tipoh2 ←tipoBoolean(tipoh.t, tipo1.t);
tipo ←tipo2;
modo ←valor;
emiteCodigo(or);
etq = etq +1;}

RExpSum (in tipoh, modoh, out tipo, modo) ::= λ
{tipo ←tipoh;
modo ←modoh;}

ExpProd (in parh, out tipo, modo)::=
{parh1 ←parh;}
ExpFact (in parh1, out tipo1, modo1)
{tipoh2 ←tipo1;
modoh2 ←modo1;}
RExpProd (in tipoh2,modoh2, out tipo2, modo2)
{tipo ←tipo2
modo ←modo2}

RExpProd(in tipoh, in modoh, out tipo, out modo)::=
OpProd(out op)
{parh1←false}
ExpFact(in parh1, out tipo1, modo1)
{tipoh2←tipoRelacion(tipoh.t, tipo1.t);
modoh2←valor;}
RExpProd(in tipoh2, in modoh2, out tipo2, out modo2)
{emiteCodigo(op);
etq ←etq+1;
tipo←tipo2;
modo←modo2;}

RExpProd (in tipoh, in modoh, out tipo, out modo)::=
AND
{parh1←false}
ExpFact(in parh1, out tipo1, modo1)
{tipoh2←tipoRelacion(tipoh.t, tipo1.t);
modoh2←valor;}
RExpProd(in tipoh2, in modoh2, out tipo2, out modo2)
{emiteCodigo(and);
etq ←etq+1;
tipo←tipo2;
modo←modo2;}

RExpProd (in tipoh, in modoh, out tipo, out modo)::= λ
{tipo←tipoh;
modo←modoh;}

ExpFact(in parh, out tipo, out modo)::=
PA
{parh1← parh;}
Exp (in parh1, out tipo1, out modo1)
PC
{tipo←tipo1;
modo←modo1;}

ExpFact (in parh, out tipo, modo) :=
numero (out lex)
{tipo ←Entero;
modo ←valor;
emiteCodigo (apila(valorDe(lex)));
etq ←etq + 1;}

ExpFact(in parh, out tipo, out modo)::=
{tipo←Boolean;
modo← valor;
emiteCodigo(apila(True));
etq←etq+1;}

ExpFact(in parh, out tipo, out modo)::=
{tipo←Boolean;
modo← valor;
emiteCodigo(apila(False));
etq←etq+1;}

ExpFact (in parh, out tipo, modo)::=
Not
ExpFact (in parh1, out tipo1, modo1)
{tipo ←if tipo1. = Boolean then tipo.t else <t: tipoError>;
modo ←valor;
emiteCodigo(not);
etq ←etq + 1;
parh ←false;}

ExpFact (in parh, out tipo, out modo)::=
Descriptor(out tipo1, lex)
{modo←variable;
tipo← if existeID(TS, lex) then tipoDeID(TS, lex) else <t: tipErr>;
if(compatible(tipo1, <t: Integer>) v compatible(tipo1, <t: Boolean>) ^ ¬parh) then
emiteCodigo(apila_ind);
if(compatible(tipo1, <t: Integer>) v compatible(tipo1, <t: Boolean>) ^ ¬parh) then
etq ←etq+1;
}