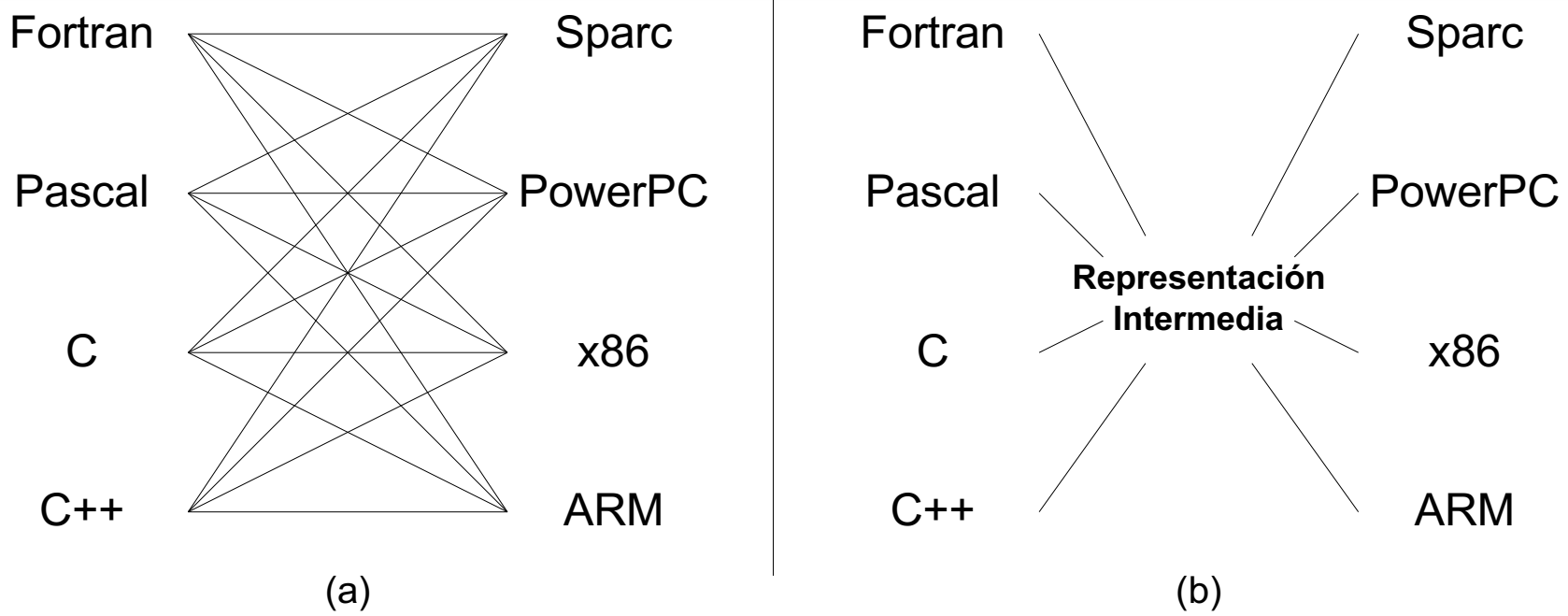
Tema 9:
Generación de Código Intermedio

1. Introducción
2. Código de tres direcciones
3. Generación de código intermedio mediante gramáticas de atributos
4. Estructuras y elementos de una matriz
5. Expresiones lógicas
6. Referencias no satisfechas. Relleno por retroceso
7. Instrucciones de control de flujo
8. Llamadas a subprogramas

1. Introducción

Representaciones intermedias



Tipos de representaciones intermedia

- Grafos Dirigidos Acíclicos (GDA)
- Árbol de sintaxis abstracta (AST)
- Formato SSA (Static Single-Assignment)
- Código de tres direcciones

2. Código de 3 direcciones

Juego de instrucciones

Asignación	x **:=** y op z
Salto incondicional	**goto** E
Salto condicional	**if** x op y **goto** E
Pila de activación	**push** x x := **pop**
Llamada y	**call** E
Retorno de subprograma	**ret**
Asignaciones relativas	a [i] := x x := a[i]
Acceso a la pila	**FP** **TOP**

3. Generación de CI mediante gramáticas atribuidas

- Usaremos un procedimiento emite con un efecto colateral: Almacena código intermedio
- *Ej.* emite (x ':=' 5 '+' 9)
- *SIGINST: Variable global con el número de la "SIGuiente INStrucción*

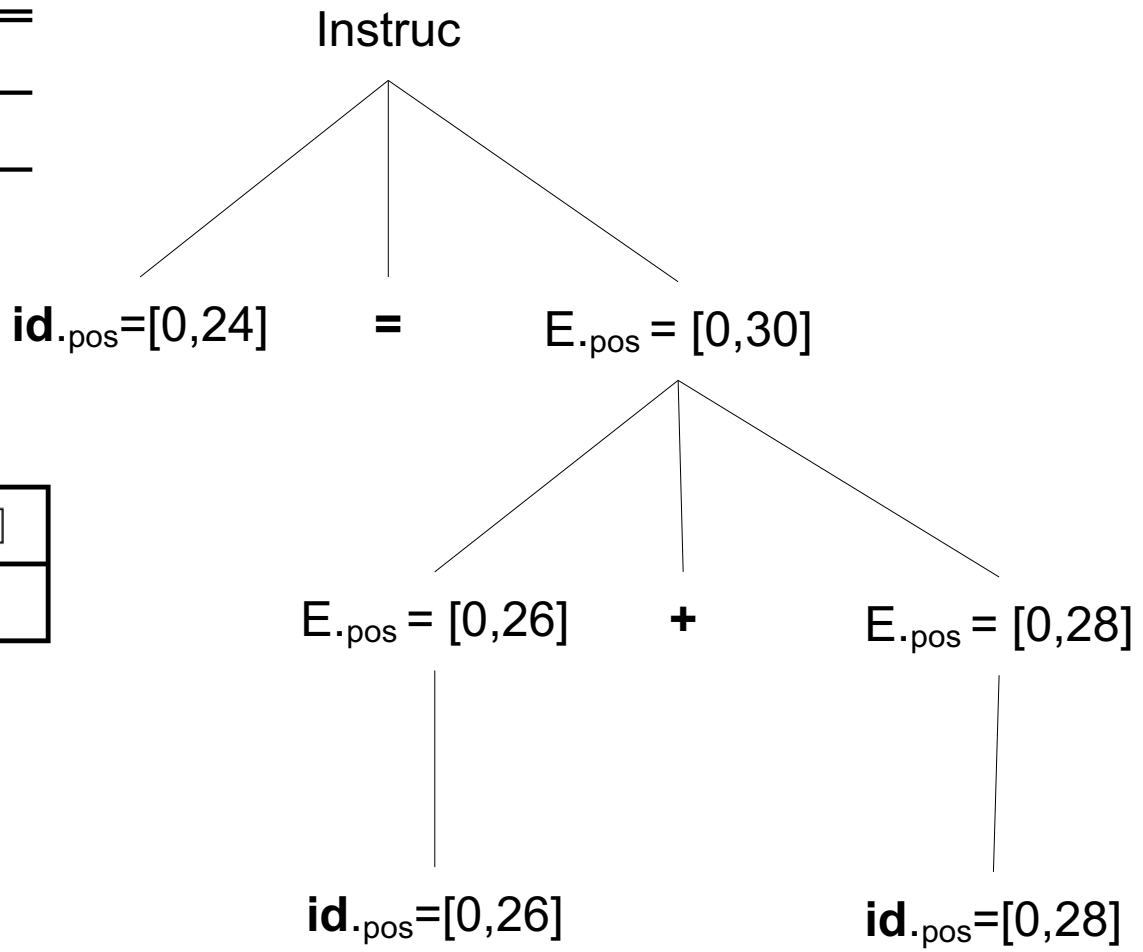
Asig → **id =** E
E → E **+** E
E → **id**

Asig → id = E	{ Asig.pos := BuscaPos (id.nom) ; emite (Asig.pos ':=' E.pos); }
$E \rightarrow E_1 + E_2$	{ E.pos := CrearVarTemp() ; emite (E.pos ':=' E_1.pos '+' E_2.pos) }
E → id	{ E.pos := BuscaPos (id.nom) }

Árbol anotado

Cadena: a = b + c

Nom	tipo	posición [nivel, desp]
a	tentero	[0,24]
b	tentero	[0,26]
c	tentero	[0,28]



Código tres direcciones generado

```
(1) [0,30] := [0,26] + [0,28]
(2) [0,24] := [0,30]
```

```
(1) t1 := b + c
(2) a := t1
```

4. Estructuras y elementos de una matriz

Acceso a miembros de una estructura

```
struct ej {
          int c1 ;
          float c2 ;
        }a, b;
```

TALLA_REAL = 4
TALLA_ENTERO = 2

La TDS quedaría:

TDS

nom	tipo	posición	...	ref
a	testructura	0 , 124		
b	testructura	0 , 130		

Tabla de registros

nom	tipo	posición
c1	tinteger	0
c2	treal	2

Acceso a miembros de una estructura

$E \rightarrow id_1.id_2$	{ si BuscaTipo (id_1.nom) <> testructura ent MemError(); sino base := BuscaPos (id_1.nom) ; si no EsMiembro (id_1.nom, id_2.nom) ent MemError() sino desp_miembro:=BuscaPosMiembro(id_1.nom, id_2.nom); E.pos := base + desp_miembro ; }
$Asig \rightarrow id_1.id_2 = E$	{ si BuscaTipo (id_1.nom) <> testructura ent MemError(); sino base := BuscaPos (id_1.nom) ; si no EsMiembro (id_1.nom, id_2.nom) ent MemError() sino desp_miembro:=BuscaPosMiembro(id_1.nom, id_2.nom); pos := base + desp_miembro ; emite(pos ':=' E.pos); }

Acceso a elemento de una matriz

Declaración: int A[n_1][n_2]... [n_n];

Acceso al elemento A[i_1][i_2]...[i_n]: $base + (((i_1 * n_2 + i_2) * n_3 + i_3 \ldots) * n_n + i_n) * Talla$

n_i es el número de elementos de la i-ésima dimensión

Inst_simple -> **id**	{ LI.nom := id.nom }
LI = E	{ emite (LI.pos := LI.pos '*' Talla (id.nom)) ; pos := BuscaPos (id.nom) ; emite (pos '[' LI.pos ']' := E.pos ; }
LI → [E]	{ LI.pos := CrearVarTemp() ; emite (LI.pos := E.pos); LI.ndim := 1; }
LI →	{ LI_1.nom := LI.nom ; }
LI_1 [E]	{ LI.ndim := LI_1.ndim + 1; LI.pos := LI_1.pos ; emite (LI.pos := LI.pos '*' Num_elementos(LI.nom, LI.ndim)) ; emite (LI.pos := LI.pos '+' E.pos) ; }

5. Expresiones lógicas

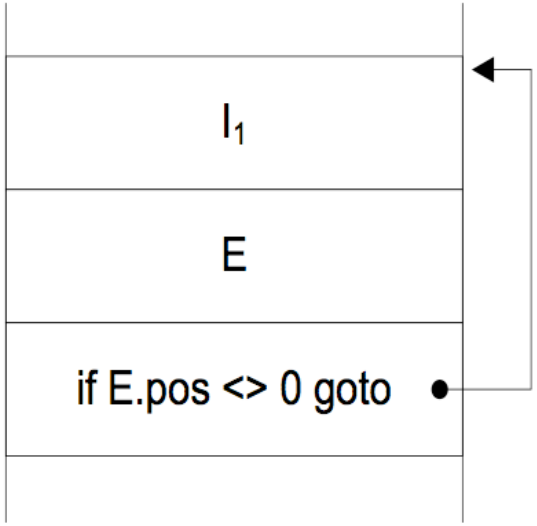
E → **true**	{ E.pos:= CrearVarTemp() ; emite(E.pos '**:=**' 1) }
E → **false**	{ E.pos:= CrearVarTemp() ; emite(E.pos '**:=**' 0) }
E → **(** E_1 **)**	{ E.pos := E_1.pos }
E → **id**	{ E.pos := BuscaPos(id.nom) }

E → E_1 **oprel** E_2	{ E.pos := CrearVarTemp() ; emite(E.pos '**:=**' 1) ; emite('if' E_1.pos oprel.op E_2.pos 'goto' SIGINST+ 2) ; emite(E.pos '**:=**' 0) }

$E \rightarrow E_1$ **or** E_2	{ E.pos:= CrearVarTemp() ; emite(E.pos '**:=**' E_1.pos '+' E_2.pos) emite('if' E.pos '<= 1' goto' SIGINST+2) ; emite(E.pos '**:= 1**')}
$\rightarrow E_1$ **and** E_2	{ E.pos:= CrearVarTemp() ; emite(E.pos '**:=**' E_1.pos '*' E_2.pos) }
$\rightarrow$ **not** E_1	{ E.pos:= CrearVarTemp() ; emite(E.pos ':=' 0) ; emite('if' E_1.pos '<> 0 goto' SIGINST+2) ; emite(E.pos ':=' 1) }

Instrucciones de control de flujo

I → **do** I_1 **while** (E)



I → **do**	{ I.inicio := SIGINST }
I_1	
while (E)	{ emite('if' E.pos '<> o goto' I.inicio) }

Instrucciones de control de flujo

I → while (E) I

I→ while	{ I.inicio:= SIGINST }
(E)	{ emite('if' E.pos '=o goto' I.fin) }
I_1	{ emite('goto' I.inicio) ;
	I.fin := SIGINST }

6. Relleno por retroceso

Perfil de las funciones usadas

ptro CreaLans(E)

Crea una lista que solo contiene un número de instrucción E a rellenar posteriormente. Devuelve un puntero a dicha lista.

ptro CompletaLans(ptro, E)

Rellena todas las instrucciones incompletas, cuyo número está contenido en la lista apuntada por ptro, con el valor de argumento E.

ptro FusionaLans (ptro, ptro)

Concatena las listas apuntadas por sus dos argumentos y devuelve un puntero a la nueva lista.

7. Instrucciones de control de flujo

While y if-else

I→ while	{ Iinicio:= SIGINST }
(E)	{ I.final:= CreaLans(SIGINST) ;
	emite('if' E.pos '=o goto' ---) }
I_1	{ emite('goto' I.inicio) ;
	CompletaLans(I.final, SIGINST) }

I → if E	{ I.falso:= CreaLans(SIGINST);
	emite('if' E.pos '=o goto' ---); }
I1 else	{ I.fin := CreaLans(SIGINST) ;
	emite('goto' ---) ;
	CompletaLans(I.falso, SIGINST) }
I2	{ CompletaLans (I.fin, SIGINST) }

for

I $\rightarrow$ for (I_1 ;	{ I.cond := SIGINST; }
E ;	{ I.fin := CreaLans (SIGINST);
	emite('if' E.pos '=o goto' ---);
	I.cuerpo := CreaLans (SIGINST);
	emite('goto' ---) ;
	I. incr := SIGINST ; }
I_2)	{ emite('goto' I.cond);
	CompletaLans(I.cuerpo, SIGINST) ; }
I	{ emite('goto' I.incr);
	CompletaLans(I.fin, SIGINST) ;
	}

8. Llamadas a subprogramas

Decl_Subprg → Tipo id (Param_Form)	{ emite('push FP');
	emite('FP := TOP');
	area_datos := CreaLans(SIGINST);
	emite('TOP := TOP + ---') ; }

Bloque ;	{ CompletaLans (area_datos, DESP)
	emite('TOP := FP');
	emite('FP := pop);
	emite('ret') }

E → id	{ E.pos := CreaVarTemp(); emite('TOP := TOP +' Talla_Tipo(id.nom)) }
(Param_Act)	{ emite('push' <estado máquina>); emite('call' BuscaPos(id.nom)) ; emite('pop' <estado máquina>); emite('TOP := TOP -' TallaParam(id.nom)) ; emite('pop' E.pos) }
Param_Act → E	{ emite('push' E.pos) }
Param_Act → E , Param_Act	{ emite('push' E.pos) }