6. Generación de Código Intermedio

- Introducción: necesidad de un Código Intermedio

6.1. GCI para expresiones e instrucciones

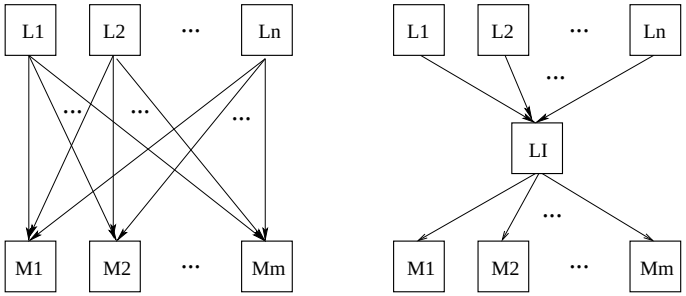
- Objetos simples
- Objetos estructurados: *registro*
- Objetos estructurados: *array*
- Expresiones lógicas

6.2. GCI para instrucciones que rompen el flujo de control

- Listas de referencias no satisfechas
- Instrucciones que rompen el flujo de control

6.3. GCI para procedimientos y funciones

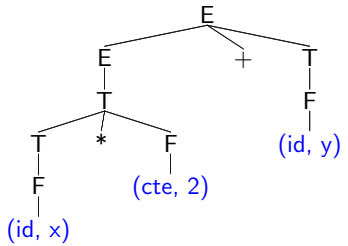
- Declaraciones de procedimientos y funciones
- Llamadas a procedimientos y funciones



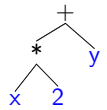
- Desarrollo de $n * m$ frente a $n + m$ compiladores.
- Descomposición inteligente de problemas.
- Parte independiente de la máquina $\gg$ parte dependiente de la máquina.
- Aparece la etapa de *Optimización Código Independiente de la Máquina*

Códigos Intermedios Gráficos

- Árbol sintáctico de análisis



- Árbol Sintáctico Abstracto



- Grafos Dirigidos Acíclicos

Códigos Intermedios Lineales

- Código máquina a pila

```
push x
push 2
multyply
push y
add
```

bytecodes es muy similar a este código máquina a pila

- Código 3-direcciones

$$
\begin{array}{lcl}
t_1 & \leftarrow & x \\
t_2 & \leftarrow & 2 \\
t_3 & \leftarrow & t_1 * t_2 \\
t_4 & \leftarrow & y \\
t_5 & \leftarrow & t_3 + t_4
\end{array}
$$

CÓDIGO 3-DIRECCIONES: INVENTARIO

$x \leftarrow y$ op z
$x \leftarrow$ op z
$x \leftarrow y$
$x \leftarrow cte$
goto e
call e
return e

if x oprel y goto e
$x \leftarrow$ pop
push x
$x \leftarrow a[i]$
$a[i] \leftarrow x$
halt

$$x \leftarrow a[i] \quad \equiv \quad x \leftarrow *(\&a + i) \quad \equiv \quad x \leftarrow *(a + i)$$

$$a[i] \leftarrow x \quad \equiv \quad *(\&a + i) \leftarrow x \quad \equiv \quad *(a + i) \leftarrow x$$

GENERACIÓN DE CÓDIGO INTERMEDIO

Expresiones e instrucciones: objetos simples

P ⇒ LD	$n = 0$; $\Delta = 0$; $\Omega = 0$;
E ⇒ E mod E	si **not** (E^1.t = E^2.t = tentero) { E.t = terror; MenError(.); } E.t = tentero; E.pos = CreaVarTemp(E.t); Emite(E.pos = E^1.pos mod E^2.pos);
⇒ (E)	E.t = E^1.t; E.pos = E^1.pos;
S ⇒ id = E	si **not** [ObtenerTds(id.nom, id.t, id.pos) **and** (id.t = E.t)] { MenError(.); } Emite(id.pos = E.pos);

$\Omega =$ primera instrucción libre en el *segmento de instrucciones*. Emite: genera una instrucción de código intermedio en la dirección Ω y posteriormente incrementa Ω.
CreaVarTemp(t): función que crea una variable temporal para un tipo dado.

$$CreaVarTemp = \Delta; \qquad \Delta = \Delta + talla(t);$$

GENERACIÓN DE CÓDIGO INTERMEDIO

Expresiones e instrucciones: objetos simples (cont.)

E ⇒ id	si **not** ObtenerTds(id.nom, E.t, id.pos) { MenError(.); E.t = terror; } E.pos = CreaVarTemp(E.t); Emite(E.pos = id.pos);
⇒ cte	E.t = cte.t; E.pos = CreaVarTemp(E.t); Emite(E.pos = cte.num);
⇒ - E	si (E^1.t ≠ tentero) { E.t=terror; MenError(.); } E.t = E^1.t; E.pos = CreaVarTemp(E.t); Emite(E.pos = - E^1.pos);

GENERACIÓN DE CÓDIGO INTERMEDIO

Expresiones e instrucciones: objetos estructurados (`registro`)

E ⇒ id . id	si **not** [ObtenerTDS(id^1.nom, id^1.t, id^1.pos) **and** (id^1.t = tregistro(id^1.lc)) **and** BuscarCampo(id^1.lc, id^2.nom, E.t, id^2.pos)] { E.t = terror; MenError(.); } pos = id^1.pos + id^2.pos; E.pos = CreaVarTemp(E.t); Emite(E.pos = pos);
S ⇒ id . id = E	si **not** [ObtenerTDS(id^1.nom, id^1.t, id^1.pos) **and** (id^1.t = tregistro(id^1.lc)) **and** BuscarCampo(id^1.lc, id^2.nom, id^2.t, id^2.pos) **and** (id^2.t = E.t)] { MenError(.); } pos = id^1.pos + id^2.pos; Emite(pos = E.pos);

BuscarCampo: función que obtiene el tipo y la posición relativa de un cierto campo, en una lista de campos de un registro. Devolverá el valor *false*, en caso de error.

Expresiones e instruciones: objetos estructurados (`array`)

E ⇒ id [E]	si **not** [ObtenerTDS(id.nom, id.t, id.pos) **and** (id.t = tvector(id.nel, E.t)) **and** (E^1.t = tentero)] { E.t = terror; MenError(.); } Emite(E^1.pos = E^1.pos ∗ talla(E.t)); E.pos = CreaVarTemp(E.t); Emite(E.pos = id.pos [E^1.pos]);
S ⇒ id [E] = E	si **not** [ObtenerTDS(id.nom, id.t, id.pos) **and** (id.t = tvector(id.nel, id.tel)) **and** (E^1.t = tentero) **and** (id.tel = E^2.t)] { MenError(.); } Emite(E^1.pos = E^1.pos ∗ talla(id.tel)); Emite(id.pos [E^1.pos] = E^2.pos);

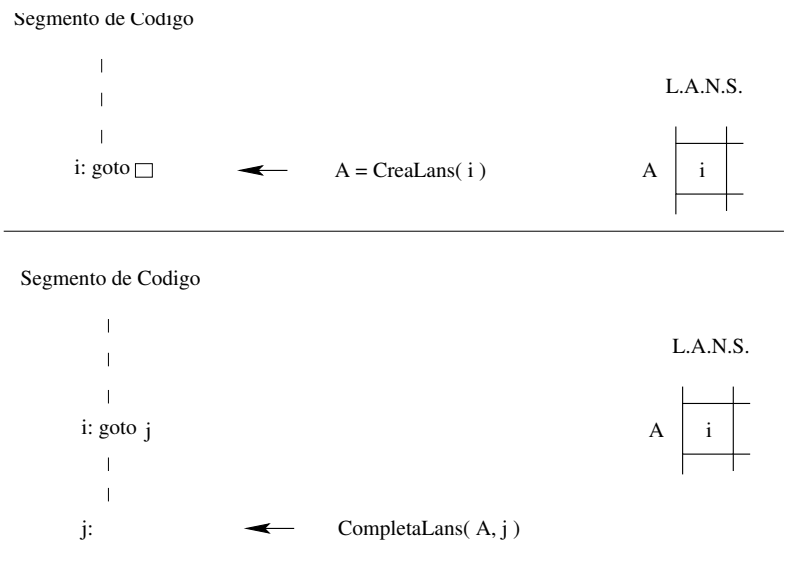
talla: función que calcula la talla asociada a un cierto tipo.

Expresiones lógicas

E ⇒ E and E	si **not** [(E^1, E^2).t = tlógico] { MenError(.); E.t = terror; } E.t = tlógico; E.pos = CreaVarTemp(E.t); Emite(E.pos = '0'); Emite(if E^1.pos = '0' goto Ω + 3); Emite(if E^2.pos = '0' goto Ω + 2); Emite(E.pos = '1');
⇒ E or E	si **not** [(E^1,E^2).t = tlógico) { MenError(.); E.t = terror; } E.t = tlógico; E.pos = CreaVarTemp(E.t); Emite(E.pos = '1'); Emite(if E^1.pos = '1' goto Ω + 3); Emite(if E^2.pos = '1' goto Ω + 2); Emite(E.pos = '0');
⇒ not E	si **not** (E^1.t = tlógico) { MenError(.); E.t = terror; } E.t = tlógico; E.pos = CreaVarTemp(E.t); Emite(E.pos = '0'); Emite(if E^1.pos = '1' goto Ω + 2); Emite(E.pos = '1');

Expresiones lógicas

E ⇒ true	E.t = tlógico; E.pos = CreaVarTemp(E.t); Emite(E.pos = '1');
⇒ false	E.t = tlógico; E.pos = CreaVarTemp(E.t); Emite(E.pos = '0');
⇒ E oprel E	si **not** [(E^1, E^2).t ∈ $\{tentero, treal\}$] { MenError(.); E.t = terror; } E.t = tlógico; E.pos = CreaVarTemp(E.t); Emite(E.pos = '1'); Emite(if E^1.pos oprel E^2.pos goto Ω + 2); Emite(E.pos = '0');



Instrucciones que implican rotura del flujo de control

S ⇒ if (E)	si (E.t ≠ tlógico)) { MenError(.); }
	S.lf = CreaLans(Ω); Emite(if E.pos = '0' goto ⊗);
S	S.fin = CreaLans(Ω); Emite (goto ⊗); CompletaLans(S.lf, Ω);
else S	CompletaLans(S.fin, Ω);
⇒ while	S.ini = Ω;
(E)	si (E.t ≠ tlógico) { MenError(.); }
	S.lf = CreaLans(Ω); Emite(if E.pos = '0' goto ⊗);
S	Emite (goto S.ini); CompletaLans(S.lf, Ω);

CreaLans: función que crea una lista de argumentos no satisfechos.

CompletaLans: completa una lista de argumentos no satisfechos.

Instrucciones que implican rotura del flujo de control (cont.)

S ⇒ do	S.ini = Ω;
S while (E)	si (E.t ≠ tlógico) { MenError(.); }
	Emite(if E.pos = '1' goto S.ini);
⇒ for (E ;	S.ini = Ω;
E ;	si (E_2.t ≠ tlógico) { MenError(.); }
	S.lv = CreaLans(Ω); Emite(if E_2.pos = '1' goto ⊗);
	S.lf = CreaLans(Ω); Emite(goto ⊗);
	S.aux = Ω;
E)	Emite(goto S.ini); CompletaLans(S.lv, Ω);
S	Emite(goto S.aux); CompletaLans(S.lf, Ω);

Funciones y parámetros

D ⇒	n++; D.aux = Δ; Δ = 0; Ω = 0;
T id (PF)	InsertarTDS(id.nom,"función",tfunción(PF.t, T.t, PF.tsp), n−1, Ω);
	Emite(push(fp)); Emite(fp = sp);
	D.d = CreaLans(Ω); Emite(sp = sp+⊗);
{ DL LI }	CompletaLans(D.d, Δ); Emite(sp = fp);
	Emite(fp = pop); Emite(return(pop));
	n−−; Δ = D.aux;
PF ⇒	LF.h = TallaSegEnlaces;
LF	PF.t = LF.t; PF.tsp = LF.talla - TallaSegEnlaces;
ϵ	PF.t=tvacio; PF.tsp = 0;
LF ⇒ DV	LF.t=DV.t; LF.talla = LF.h + DV.talla;
	insertarTds(DV.nom, "parámetro", DV.t, n, -LF.talla));
⇒ DV	LF'.h = LF.h + DV.talla;
	InsertarTds(DV.nom, "parámetro", DV.t, n, -LF'.h);
, LF	LF.t = LF'.t⊗DV.t; LF.talla = LF'.talla;

Llamadas a funciones

E ⇒ id (	Si **not** [ObtenerTDS(id.nom, id.t, id.dpi)
	and (id.t = tfunción(id.dom, E.t, id.tsp))]
	{ E.t=terror; MenError(.); }
	Emite(sp = sp + talla(E.t));
A)	Si (A.t ≠ id.dom) { E.t = terror; MenError(.); }
	Emite(push(Ω + 2)); Emite(call id.dpi);
	Emite(sp = sp − id.tsp);
	E.pos = CreaVarTemp; Emite(E.pos = pop);
A ⇒ ϵ	A.t = tvacio;
⇒ LA	A.t = LA.t;
LA ⇒ E	LA.t = E.t; Emite(push(E.pos));
⇒ E , LA	LA.t = E.t ⊗ LA'.t; Emite(push(E.pos));

Ejemplo-1

S	⇒ swich (E) {	Si (E.t ≠ tentero) MenError(.);
		L.pos = E.pos; L.h = `nil`;
	L }	CompletaLans(L.b, Ω); S.b = `nil`;
	⇒ break	S.b = CreaLans(Ω); Emite(goto ⊗);
L	⇒ case cte :	Si (cte.t ≠ tentero) MenError(.);
		L.fin = CreaLans(Ω); Emite(if cte.num ≠ L.pos goto ⊗);
		CompletaLans(L.h, Ω);
	S	L_1.h = CreaLans(Ω); Emite(goto ⊗);
		L_1.pos = L.pos; CompletaLans(L.fin, Ω);
	L	L.b = FusionaLans(S.b, L_1.b);
	⇒ ϵ	CompletaLans(L.h, Ω); L.b = `nil`;
	⇒ default :	CompletaLans(L.h, Ω);
	S	L.b = S.b;