6. Generación de Código Intermedio

- Introducción: necesidad de un Código Intermedio

6.1. GCI para expresiones e instrucciones

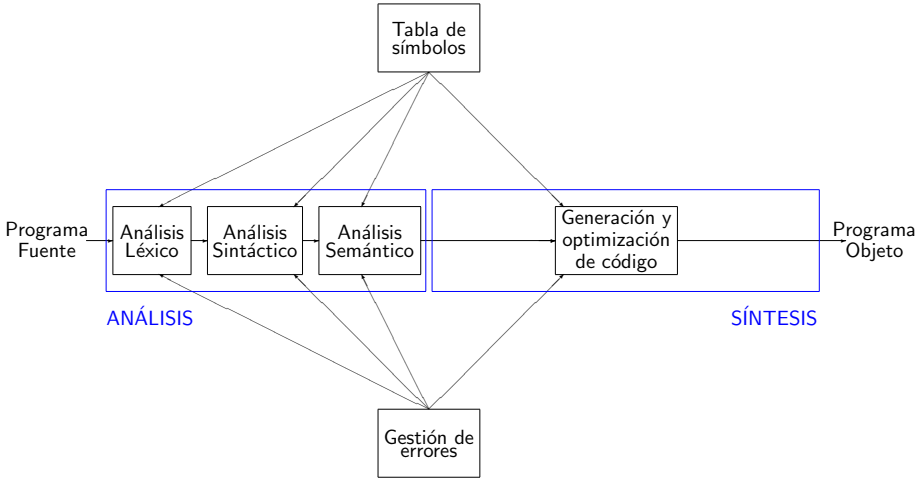
- Objetos simples
- Objetos estructurados: *registro*
- Objetos estructurados: *array*
- Expresiones lógicas

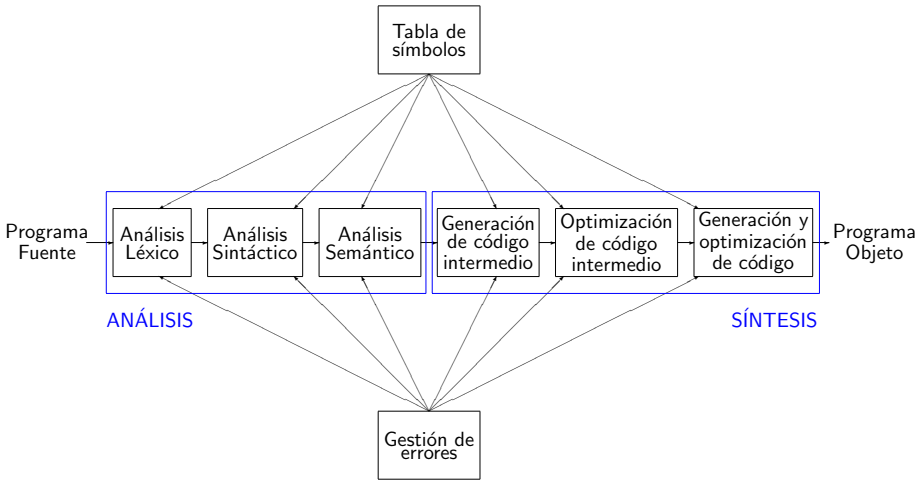
6.2. GCI para instrucciones que rompen el flujo de control

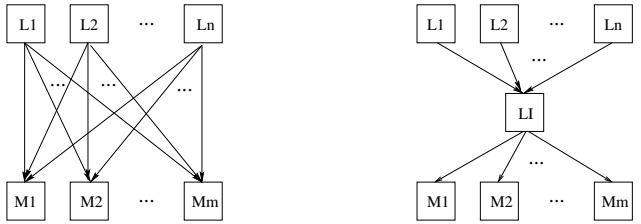
- Listas de referencias no satisfechas
- Instrucciones que rompen el flujo de control

6.3. GCI para procedimientos y funciones

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





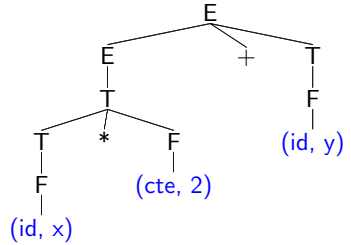


- Desarrollo de $n * m$ frente a $n + m$ compiladores.
- Descomposición inteligente de problemas.
- LI de muy bajo nivel
  ⇒ permite una posterior *Generación de Código* sencilla y fácil de implementar
- LI independiente de la máquina
  ⇒ permite una etapa de *Optimización Código Intermedio*, independiente de la máquina, de importancia creciente
- Parte independiente de la máquina $\gg$ parte dependiente de la máquina.

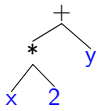
Código Intermedio: taxonomía

Códigos Intermedios Gráficos

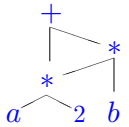
Árbol sintáctico de análisis $[x * 2 + y]$



Árbol Sintáctico Abstracto $[x * 2 + y]$



Grafos Dirigidos Acíclicos $[a * 2 + a * 2 * b]$



Código Intermedio: taxonomía

Códigos Intermedios Lineales

Código de una máquina a pila $[x * 2 + y]$

[por ejemplo: **bytecodes**]

```
iload x
ldc 2
imul
iload y
iadd
```

Código 3-direcciones $[x * 2 + y]$

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

Código 3-direcciones: inventario

$x \leftarrow y$ **op** z
$x \leftarrow$ **op** z
$x \leftarrow y$
$x \leftarrow cte$
$x \leftarrow$ **pop**
push x
halt

goto e
call e
return e
if x **oprel** y **goto** e
$x \leftarrow a[i]$
$a[i] \leftarrow x$

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

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

Generación de código intermedio

Expresiones e instrucciones: objetos simples

P ⇒ LD	$n = 0$; $\Delta = 0$; $\Omega = 0$;
E ⇒ E mod E	SI ¬[E^1.t = E^2.t = tentero] { E.t = terror; MenError(.); }
	SINO E.t = tentero; E.d = **CreaVarTemp**(E.t);
	Emite(E.d = E^1.d mod E^2.d);
⇒ (E)	E.t = E^1.t; E.d = E^1.d;
S ⇒ id = E	SI ¬[**obtTdS**(id.n, id.t, id.d) ∧ (id.t = E.t)] { MenError(.); }
	SINO **Emite**(id.d = E.d);

Ω 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, devuelve el valor altual de Δ y hace $\Delta = \Delta + talla(t)$;

Expresiones e instrucciones: objetos simples (cont.)

E ⇒ id	SI ¬[**obtTdS**(id.n, id.t, id.d) { MenError(.); E.t = terror; } SINO E.t = id.t; E.d = **CreaVarTemp**(E.t); **Emite**(E.d = id.d);
⇒ cte	E.t = cte.t; E.d = **CreaVarTemp**(E.t); **Emite**(E.d = cte.num);
⇒ - E	SI (E^1.t ≠ tentero) { E.t = terror; MenError(.); } SINO E.t = E^1.t; E.d = **CreaVarTemp**(E.t); **Emite**(E.d = - E^1.d);

Expresiones e instrucciones: objetos estructurados (registro)

E ⇒ id . id	SI ¬[**obtTdS**(id^1.n, id^1.t, id^1.d) ∧ (id^1.t = tregistro(id^1.lc)) ∧ **obtCampo**(id^1.lc, id^2.n, id^2.t, id^2.d)] { E.t = terror; MenError(.); } SINO E.t = id^2.t; desp = id^1.d + id^2.d; E.d = **CreaVarTemp**(E.t); **Emite**(E.d = desp);
S ⇒ id . id = E	SI ¬[**obtTdS**(id^1.n, id^1.t, id^1.d) ∧ (id^1.t = tregistro(id^1.lc)) ∧ **obtCampo**(id^1.lc, id^2.n, id^2.t, id^2.d) ∧ (id^2.t = E.t)] { MenError(.); } SINO desp = id^1.d + id^2.d; **Emite**(desp = E.d);

obtCampo: 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 ¬[**obtTdS**(id.n, id.t, id.d) ∧ (id.t = tvector(id.nel, id.tel)) ∧ (E^1.t = tentero)] { E.t = terror; MenError(.); } SINO E.t = id.tel; **Emite**(E^1.d = E^1.d ∗ **talla**(E.t)); E.d = **CreaVarTemp**(E.t); **Emite**(E.d = id.d [E^1.d]);
S ⇒ id [E] = E	SI ¬[**obtTdS**(id.n, id.t, id.d) ∧ (id.t = tvector(id.nel, id.tel)) ∧ (E^1.t = tentero) ∧ (id.tel = E^2.t)] { MenError(.); } SINO **Emite**(E^1.d = E^1.d ∗ **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 ¬[(E^1, E^2).t = tlógico] { E.t = terror; MenError(.); } SINO E.t = tlógico; E.d = **CreaVarTemp**(E.t); **Emite**(E.d = E^1.d ∗ E^2.d);
⇒ E or E	SI ¬[(E^1, E^2).t = tlógico] { E.t = terror; MenError(.); } SINO E.t = tlógico; E.d = **CreaVarTemp**(E.t); **Emite**(E.d = E^1.d + E^2.d); **Emite**(if E.d ≤ 1 goto Ω + 2); **Emite**(E.d = 1);
⇒ not E	SI ¬[(E^1, E^2).t = tlógico] { E.t = terror; MenError(.); } SINO E.t = tlógico; E.d = **CreaVarTemp**(E.t); **Emite**(E.d = 1 − E^1.d);

Generación de código intermedio

Expresiones lógicas

E	⇒ true	E.t = tlógico; E.d = **CreaVarTemp**(E.t); **Emite**(E.d = '1');
	⇒ false	E.t = tlógico; E.d = **CreaVarTemp**(E.t); **Emite**(E.d = '0');
	⇒ E oprel E	SI ¬[(E¹, E²).t ∈ {tentero, treal }] { E.t = terror; MenError(.); } SINO E.t = tlógico; E.d = **CreaVarTemp**(E.t); **Emite**(E.d = '1'); **Emite**(if E¹.d oprel E².d goto Ω + 2); **Emite**(E.d = '0');

Listas de referencias no satisfechas

Segmento de código			L.A.N.S
	⋮		...
i:	goto ⊗	⟸ A = CreaLans (i)	A [i]
	⋮		...
j:	if ··· goto ⊗	⟸ B = CreaLans (i)	B [j]
	⋮		...
k:	···	⟸ C = Fusionalans (A, B)	

Listas de referencias no satisfechas

Segmento de código			L.A.N.S
	⋮		...
i:	goto ⊗	⟸ A = CreaLans (i)	C [i, j]
	⋮		...
j:	if ··· goto ⊗	⟸ B = CreaLans (i)	
	⋮		
k:	···	⟸ C = Fusionalans (A, B)	
	⋮		
n:	···	⟸ CompletaLans (C, n)	

Generación de código intermedio

Instrucciones que implican rotura del flujo de control

S	⇒ if (E)	SI (E.t ≠ tlógico)) { MenError(.); } S.lf = **CreaLans**(Ω); **Emite**(if E.d = '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.d = '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.d = '1' goto S.ini);
⇒ for (E ;	S.ini = Ω;
E ;	SI (E$_2$.t ≠ tlógico) { MenError(.); }
	S.lv = **CreaLans**(Ω); **Emite**(if E$_2$.d = '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;
T id (PF)	**insTdS**(id.nom, "función", tfunción(PF.t, T.t, PF.talla), 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 ⇒ ϵ	PF.t = tvacio; PF.talla = 0;
⇒ LF	PF.t = LF.t; PF.talla = LF.talla − TallaSegEnlaces;
LF ⇒ DV , LF	LF.t = DV.t ⊗ LF'.t; LF.talla = LF'.talla + DV.talla;
	insTdS(DV.nom, "parámetro", DV.t, n, -LF.talla);
⇒ DV	LF.t = DV.t; LF.talla = TallaSegEnlaces + DV.talla;
	insTdS(DV.nom, "parámetro", DV.t, n, -LF'.talla);

Llamadas a funciones

E ⇒ id (	SI ¬[**obtTdS**(id.nom, id.t, id.dpi)
	∧ (id.t = tfunción(id.td, id.tr, id.tasp))]
	{ E.t = terror; MenError(.); }
	SINO E.t = id.tr;
	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.d = **CreaVarTemp**(E.t); **Emite**(E.d = pop);
A ⇒ ϵ	A.t = tvacio;
⇒ LA	A.t = LA.t;
LA ⇒ E	LA.t = E.t; **Emite**(push(E.d));
⇒ E ,	**Emite**(push(E.ds));
LA	LA.t = E.t ⊗ LA'.t;

S ⇒ S ; S	S.b = **FusionaLans**(S^1.b, S^2.b);
⇒ swich (E) {	SI (E.t ≠ tentero) MenError(.);
	L.d = E.d; 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.d goto ⊗);
	CompletaLans(L.h, Ω);
S	L^1.h = **CreaLans**(Ω); **Emite**(goto ⊗);
	L^1.d = L.d; **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;