Tema 8 | Lenguajes Intermedios

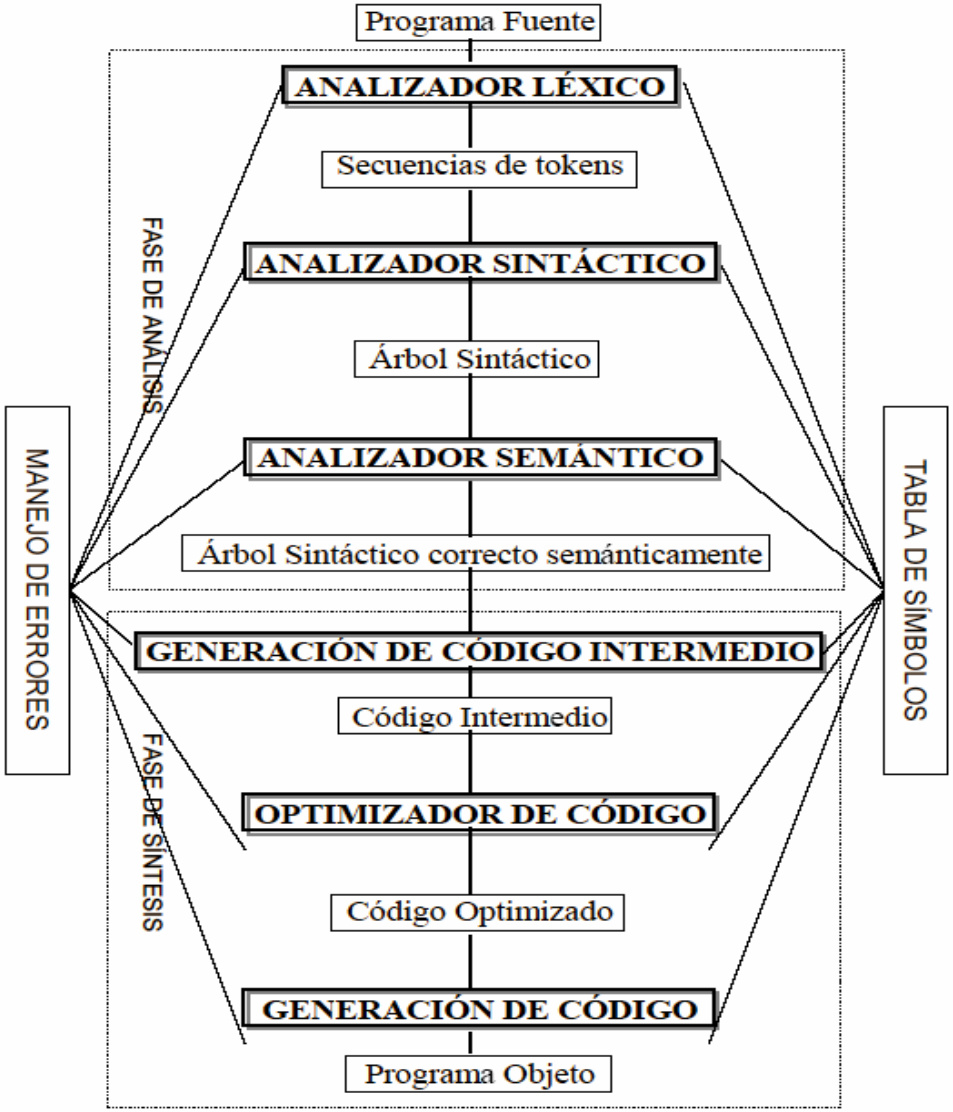
Contenidos

Bibliografía básica

20-Feb-2011

[Aho90] Alfred V. Aho, Ravi Sethi, Jffrey D. Ullman
Compiladores. Principios, técnicas y herramientas. Addison-Wesley Iberoamericana 1990.

[Trem85] J. Tremblay, P.G. Sorenson
The theory and practice of compiler writing. Mc-Graw-Hill 1985.


Programa Fuente
ANALIZADOR LÉXICO
Secuencias de tokens
ANALIZADOR SINTÁCTICO
Árbol Sintáctico
ANALIZADOR SEMÁNTICO
Árbol Sintáctico correcto semánticamente
GENERACIÓN DE CÓDIGO INTERMEDIO
Código Intermedio
OPTIMIZADOR DE CÓDIGO
Código Optimizado
GENERACIÓN DE CÓDIGO
Programa Objeto
FASE DE ANÁLISIS
FASE DE SÍNTESIS
MANEJO DE ERRORES
TABLA DE SÍMBOLOS

8.1 Lenguaje intermedio. Tipos

Es una representación **más abstracta y uniforme** que un **lenguaje máquina concreto**.

Su misión es descomponer las expresiones complejas en binarias y las sentencias complejas en sentencias simples.

Ventajas:

- Permite una fase de análisis semántico independiente de la máquina.
- Se pueden realizar optimizaciones sobre el código intermedio (las complejas rutinas de optimización son independientes de la máquina).

Desventajas:

- Introduce en el compilador una nueva fase de traducción.

8.1 Lenguaje Intermedio. Tipos

Tipos de lenguajes intermedios

- **Árbol sintáctico**
- **Árbol sintáctico abstracto**
  - Todos los nodos del árbol representan **símbolos terminales**
  - Los nodos hijos son operandos y los nodos internos son **operadores**

- **Grafo dirigido acíclico** (GDA)
- **Notación posfija**
- **Usados en máquinas abstractas**
- **N-tuplas**
  - Cada sentencia del lenguaje intermedio consta de **N elementos**: (Operador, Operando1, Operando2, ... , OperandoN-1)
  - Los más usuales son los **tercetos (tripletas)** y los **cuartetos (cuádruplas)**, llamados también **código de tres direcciones**.

8.1 Lenguaje intermedio. Tipos

Tercetos:

```
<operador>, <operando_1>, <operando_2>
```

Ejemplo: `d = a + b * c`

```
[1] (*, b, c)
[2] (+, a, [1])
[3] (=, d, [2])
```

Cuartetos:

```
<operador>, <operando_1>, <operando_2>,
                          <resultado>
```

Ejemplo: `d = a + b * c`

```
(*, b, c, temp1)
(+, a, temp1, temp2)
(=, temp2, --, d)
```

8.2 Lenguaje intermedio de cuartetos. Proposiciones

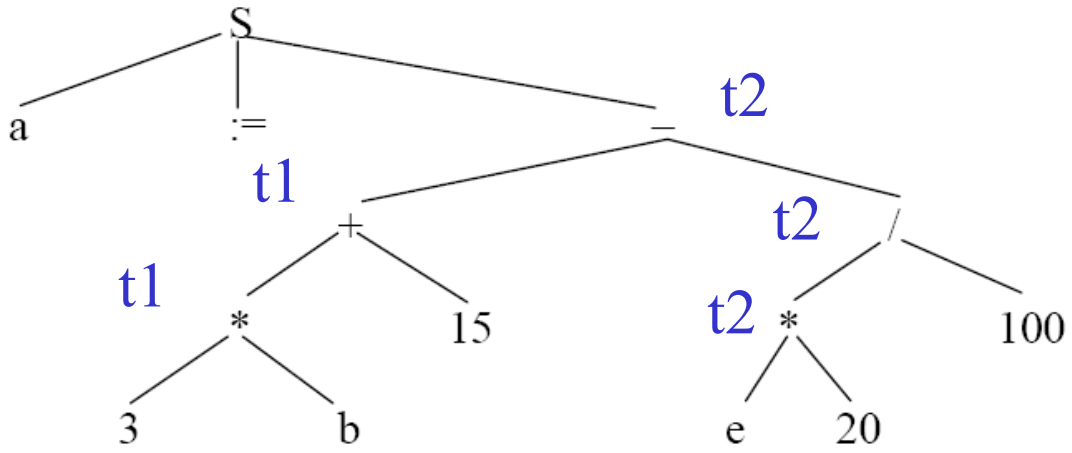
Tipos de Instrucciones	
Asignación	`x := y op z` `x := op z`
Copia	`x := y`
Salto incondicional	`goto etiqueta`
Salto condicional	`if x op_relacional y goto etiqueta`
Asignación de índices	`x := y[i]` `x[i] := y` `&y+i` `&x+i`
Asignación de direcciones y apuntadores	`x := &y`
Procedimientos	
Parámetros	`param x`
Llamada	`call p, N` `/* N: nº parámetros */`
Vuelta	`return y`

8.2 Lenguaje intermedio de cuartetos. Proposiciones

ESQUEMA DE TRADUCCIÓN

- **Asignación**: $S \rightarrow \text{id} := exp$
  (Descomposición de *exp* en operaciones binarias)

Ejemplo 8.1: Descomposición en operaciones binarias de la expresión siguiente: `a:=3*b+15-(e*20)/100`




```
s_asigna_1:
            t1 := 3 * b
            t1 := t1 + 15
            t2 := e * 20
            t2 := t2 / 100
            t2 := t1 - t2
            a := t2
```

8.2 Lenguaje intermedio de cuartetos. Proposiciones

ESQUEMA DE TRADUCCIÓN

- **Lazo**: $S \rightarrow$ while E do S_1

```
s_comienzo_1:
              código de asignación de expresión E
              if E = 0 goto s_despues_1
              código de sentencias de S1
              goto s_comienzo_1
s_despues_1:
```

8.2 Lenguaje intermedio de cuartetos. Proposiciones

ESQUEMA DE TRADUCCIÓN

- **if-then-else**: $S \rightarrow \text{if } E \text{ then } S_1 \text{ else } S_2$

```
s_comienzo_1:
                código de asignación de expresión E
                if E goto s_then_1
                goto s_else_1
s_then_1:
                código de sentencias de S1
                goto s_despues_1
s_else_1:
                código de sentencias de S2
s_despues_1:
```

8.2 Lenguaje intermedio de cuartetos. Proposiciones

ESQUEMA DE TRADUCCIÓN

- **Case**: switch E
  begin
  case $V_1 : S_1$;
  case $V_2 : S_2$;
  …
  case $V_{n-1} : S_{n-1}$;
  default : S_n
  end;

```
s_inicio_case:
                    código evaluación de E en t
                    if t ≠ V1 goto L1
                    código S1
                    goto s_despues_1
L1:                 if t ≠ V2 goto L2
                    código S2
                    goto s_despues_1
                          ...
Ln-2:               if t ≠ Vn-1 goto Ln-1
                    código Sn-1
                    goto s_despues_1
Ln-1:               código Sn
s_despues_1:
```

8.2 Lenguaje intermedio de cuartetos. Proposiciones

ESQUEMA DE TRADUCCIÓN

- **Case**: switch E
  begin
  case V_1 : S_1 ;
  case V_2 : S_2 ;
  …
  case V_{n-1} : S_{n-1} ;
  default : S_n
  end;

Otra forma de implantación:

```
s_inicio_case:
                    código evaluación de E en t
                    goto prueba
L1:                 código S1
                    goto s_despues_1
L2:                 código S2
                    goto s_despues_1
                              ...
Ln-1:               código Sn-1
                    goto s_despues_1
Ln:                 código Sn
                    goto s_despues_1
prueba:
                    if t = V1 goto L1
                    if t = V2 goto L2
                              ...
                    if t = Vn-1 goto Ln-1
                    goto Ln
s_despues_1:
```

8.3 Generación de código en cuartetos

Cuando se genera código de tres direcciones, se construyen **nombres temporales** para los nodos interiores de un árbol sintáctico. Inicialmente, se creará un nuevo nombre cada vez que se necesite una variable temporal.

- Los atributos sintetizados empleados son, entre otros, los siguientes:
  - *lugar*: contendrá el valor del atributo.
  - *codigo*: contendrá la secuencia de proposiciones de tres direcciones.
- La función *tempnuevo* devuelve una secuencia de nombres distintos $t_1, t_2, \ldots, t_n$ en respuesta a sucesivas llamadas.
- La función *etiqnueva* devuelve una secuencia de nombres distintos $e_1, e_2, \ldots, e_n$ en respuesta a sucesivas llamadas.
- Por notación, se utiliza *gen* $(\cdots)$ para representar una proposición de tres direcciones concreta dada como conjunto de argumentos.

8.3.1 Expresiones aritméticas

Expresiones aritméticas y sentencia de asignación

S : **id** := E	{ S.*codigo* = E.*codigo* || *gen* (id.*lugar* ' := ' E.*lugar*) ; }
E : E_1 + E_2	{ E.*lugar* = *tempnuevo* ; E.*codigo* = E_1.*codigo* || E_2.*codigo* || *gen* (E.*lugar* ' := ' E_1.*lugar* '+' E_2.*lugar*) ; }
E : E_1 * E_2	{ E.*lugar* = *tempnuevo* ; E.*codigo* = E_1.*codigo* || E_2.*codigo* || *gen* (E.*lugar* ' := ' E_1.*lugar* '*' E_2.*lugar*) ; }
E : - E_1	{ E.*lugar* = *tempnuevo* ; E.*codigo* = E_1.*codigo* || *gen* (E.*lugar* ' := - ' E_1.*lugar*) ; }
E : (E_1)	{ E.*lugar* = E_1.*lugar* ; E.*codigo* = E_1.*codigo* ; }
E : **id**	{ E.*lugar* = id.*lugar* ; E.*codigo* = ' ' ; }

8.3.2 Expresiones lógicas

Existen dos métodos para evaluar las expresiones lógicas:

- Mediante **representación numérica**: Asociar valores numéricos a true y false (ejemplo: En C true es cualquier valor distinto de cero y false es cero).

- Mediante **flujo de control**: Representar el valor de una expresión lógica dependiendo de la posición alcanzada en el programa.

8.3.2 Expresiones lógicas

Ejemplo 8.1: Generación de código en cuartetos para expresiones lógicas representadas de forma numérica.

A and not B or true	A > B or C = D
temp1 := not B	0: if A > B goto 3
temp2 := A and temp1	1: temp1 := false
temp3 := true	2: goto 4
temp4 := temp2 or temp3	3: temp1 := true
	4: if C = D goto 7
	5: temp2 := false
	6: goto 8
	7: temp2 := true
	8: temp3 := temp1 or temp2

8.3.2 Expresiones lógicas

Representación numérica	
E : E_1 **or** E_2	{ E.*lugar* = *tempnuevo* ; *gen* (E.*lugar* ‘`:=`‘ E_1.*lugar* ‘`or`’ E_2.*lugar*) ; }
E : E_1 **and** E_2	{ E.*lugar* = *tempnuevo* ; *gen* (E.*lugar* ‘`:=`‘ E_1.*lugar* ‘`and`’ E_2.*lugar*) ; }
E : **not** E_1	{ E.*lugar* = *tempnuevo* ; *gen* (E.*lugar* ‘`:= not`’ E_1.*lugar*) ; }
E : **(** E_1 **)**	{ E.*lugar*= E_1.*lugar* ; }
E : id **oprel** id	{ E.*lugar*= *tempnuevo* ; *gen* (‘`if`’ id_1.*lugar* **oprel.*op*** id_2.*lugar* ‘`goto`‘ **sigt prop+3**) ; *gen* (E.*lugar* ‘`= 0`’) ; *gen* (‘`goto`’ **sigt prop+2**) ; *gen* (E.*lugar* ‘`:= 1`’) ; }
E : **true**	{ E.*lugar*= *tempnuevo* ; *gen* (E.*lugar* ‘`:= 1`’) ; }
E : **false**	{ E.*lugar*= *tempnuevo* ; *gen* (E.*lugar* ‘`:= 0`’) ; }

8.3.2 Expresiones lógicas

Ejemplo 8.2: Generación de código en cuartetos para expresiones lógicas representadas mediante flujo de control.

E := A > B **or** C = D

```
            if A > B goto Everdadera
            goto L1
        L1: if C = D goto Everdadera
            goto Efalsa
Everdadera: E = verdadera
            goto L2
    Efalsa: E = falsa
        L2:
```

8.3.2 Expresiones lógicas

Flujo de control	
E : E_1 **or** E_2	{ $E_1.verdadera$ = E.*verdadera* ; $E_1.falsa$ = *etiqnueva* ; $E_2.verdadera$ = E.*verdadera* ; $E_2.falsa$ = E.*falsa* ; E.*codigo* = $E_1.codigo$ || *gen* ($E_1.falsa$ ' : ') || $E_2.codigo$; }
E : E_1 **and** E_2	{ $E_1.verdadera$ = *etiqnueva* ; $E_1.falsa$ = E.*falsa* ; $E_2.verdadera$ = E.*verdadera* ; $E_2.falsa$ = E.*falsa* ; E.*codigo* = $E_1.codigo$ || *gen* ($E_1.verdadera$ ' : ') || $E_2.codigo$; }
E : **not** E_1	{ $E_1.verdadera$ = E.*falsa* ; $E_1.falsa$ = E.*verdadera* ; E.*codigo* = $E_1.codigo$; }

8.3.2 Expresiones lógicas

Flujo de control	
E : **(** E_1 **)**	{ $E_1.verdadera = E.verdadera$; $E_1.falsa = E.falsa$; $E.codigo = E_1.codigo$; }
E : id **oprel** id	{ $E.codigo = gen$ (‘`if`’ $id_1.lugar$ **oprel.*op*** $id_2.lugar$ ‘`goto`‘ $E.verdadera$) || gen (‘`goto`‘ $E.falsa$) ; }
E : **true**	{ $E.codigo = gen$ (‘`goto`‘ $E.verdadera$) ; }
E : **false**	{ $E.codigo = gen$ (‘`goto`‘ $E.falsa$) ; }

8.3.3 Estructura de control: if-then

S : **if** E **then** S_1	{ E.*verdadera* = *etiqnueva* ; E.*falsa* = S.*siguiente* ; S_1.*siguiente* = S.*siguiente* ; S.*codigo* = E.*codigo* || *gen* (‘**if**’ !E.lugar goto E.*falsa*) || *gen* (E.*verdadera* ‘**:**’) || S_1.*codigo* || *gen* (‘**goto**‘ S.*siguiente*) ; }

```
comienzo:
                código de asignación de expresión E
                if !E.lugar goto E.falsa
etiqnueva:
                código de sentencias de S1
                goto S.siguiente
```

8.3.3 Estructura de control: if-then-else

S : **if** E **then** S_1 **else** S_2	{ E.*verdadera* = *etiqnueva1* ; E.*falsa* = *etiqnueva2* ; S_1.*siguiente* = S.*siguiente* ; S_2.*siguiente* = S.*siguiente* ; S.*codigo*= E.*codigo* || *gen* (‘**if**’ !E.lugar goto E.*falsa*) || *gen* (E.*verdadera* ‘**:**’) || S_1.*codigo* || *gen* (‘**goto**‘ S.*siguiente*) ; || *gen* (E.*falsa* ‘**:**’) || S_2.*codigo* ; }

```
comienzo:
                código de asignación de expresión E
                if !E.lugar goto E.falsa
etiqnueva1:
                código de sentencias de S1
                goto S.siguiente
etiqnueva2:
                código de sentencias de S2
```

8.3.3 Estructura de control: while

S : **while** E **do** S_1	{ *S.comienzo* = *etiqnueva1* ; *S.despues* = *etiqnueva2* ; *S.codigo* = *gen* (*S.comienzo* ':') ‖ *E.codigo* ‖ *gen* ('**if**' *E.lugar* '**= 0 goto**' *S.despues*) ‖ S_1*.codigo* ‖ *gen* ('**goto**' *S.comienzo*) ‖ *gen* (*S.despues* ':') ; }

```
etiqnueva1:
                código de asignación de expresión E
                if E.lugar = 0 goto etiqnueva2
                código de sentencias de S1
                goto etiqnueva1
etiqnueva2:
```

8.4 Ejemplo de lenguaje intermedio de máquina abstracta: Código P

El código P (P-Code) comenzó como un código ensamblador objeto estándar producido por varios compiladores de Pascal en la década de los 70 y principios de los 80.

Diseñado para ser el código real de una máquina abstracta denominada **máquina P** para la cual se escribieron intérpretes para varias arquitecturas.

La idea general era que los compiladores de Pascal se transportaran fácilmente a diferentes arquitecturas. Se trata de un código más cercano al código máquina que al de tres direcciones.

8.4 Ejemplo de lenguaje intermedio de máquina abstracta: Código P

La máquina P está compuesta por:

- Conjunto de instrucciones.
- Conjunto de datos.
- Memoria organizada mediante pila y montículo (heap).
- Registros de control de la memoria.

Ejemplo 8.3: Expresión traducida a código P: **`2*a+(b-3)`**

```
ldc 2           ; carga la constante 2
lod a           ; carga el valor de la variable a
mpi             ; multiplicación entera
lod b           ; carga el valor de la variable b
ldc 3           ; carga la constante 3
sbi             ; resta entera
adi             ; suma de enteros
```