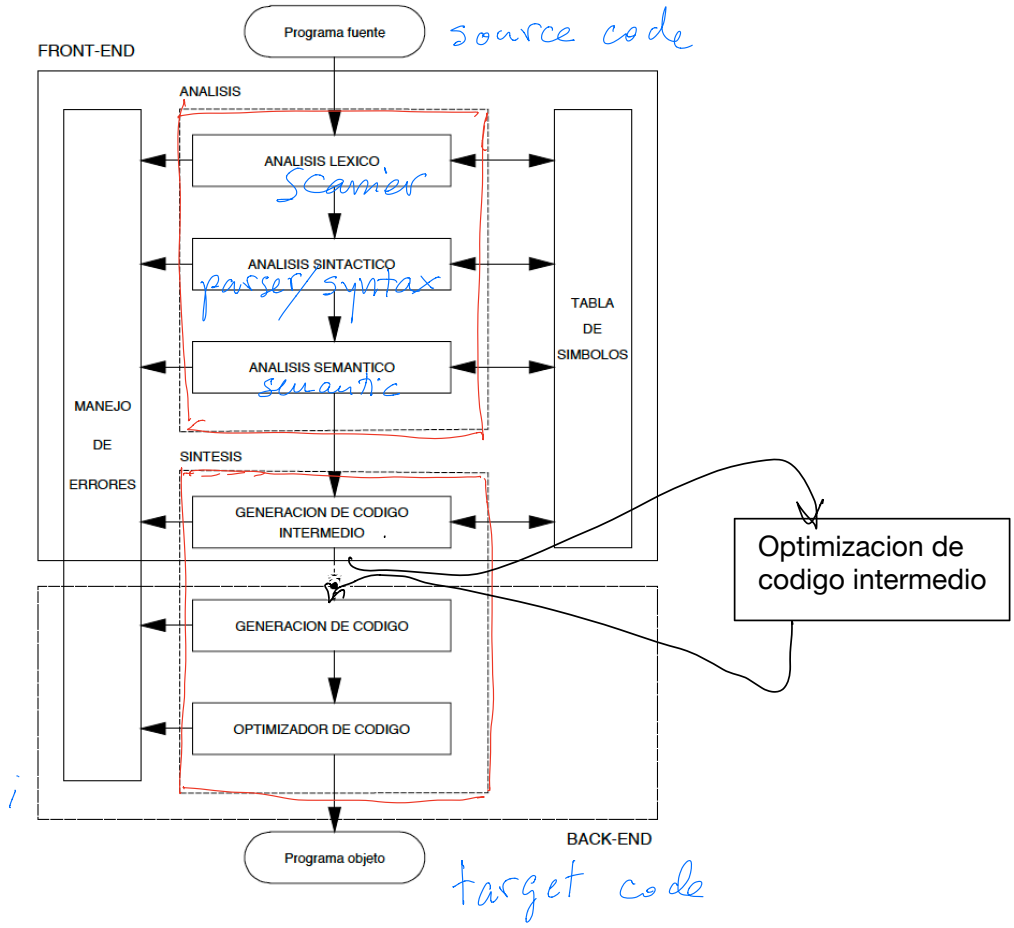
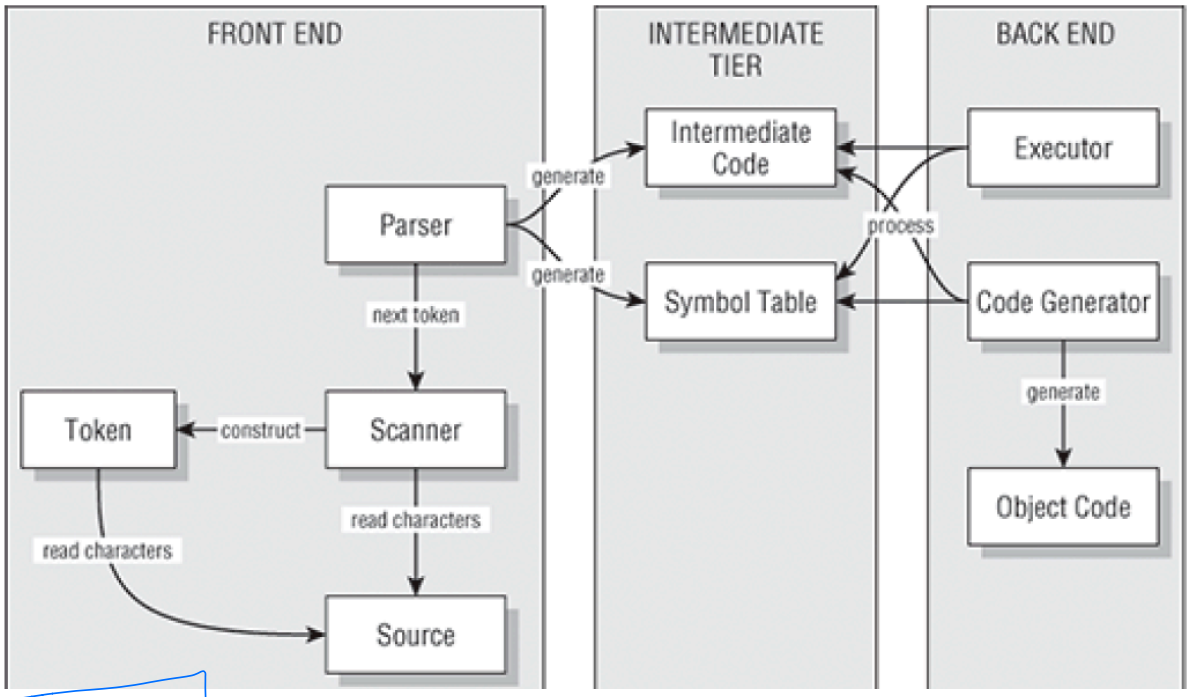
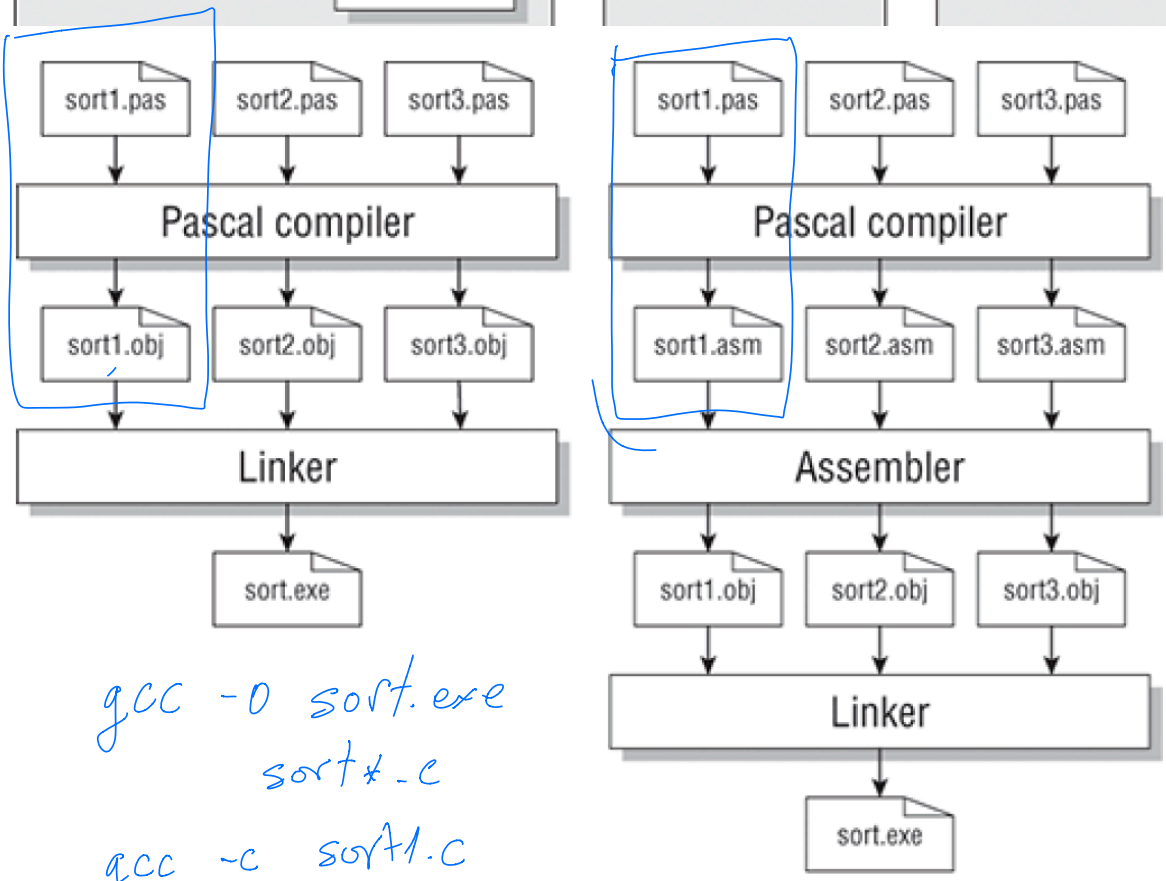

Programa fuente
source code
FRONT-END
ANALISIS
ANALISIS LEXICO
Scanner
ANALISIS SINTACTICO
parser/ syntax
ANALISIS SEMANTICO
Semantic
TABLA
DE
SIMBOLOS
MANEJO
DE
ERRORES
SINTESIS
GENERACION DE CODIGO
INTERMEDIO
Optimizacion de
codigo intermedio
GENERACION DE CODIGO
OPTIMIZADOR DE CODIGO
BACK-END
Programa objeto
target code

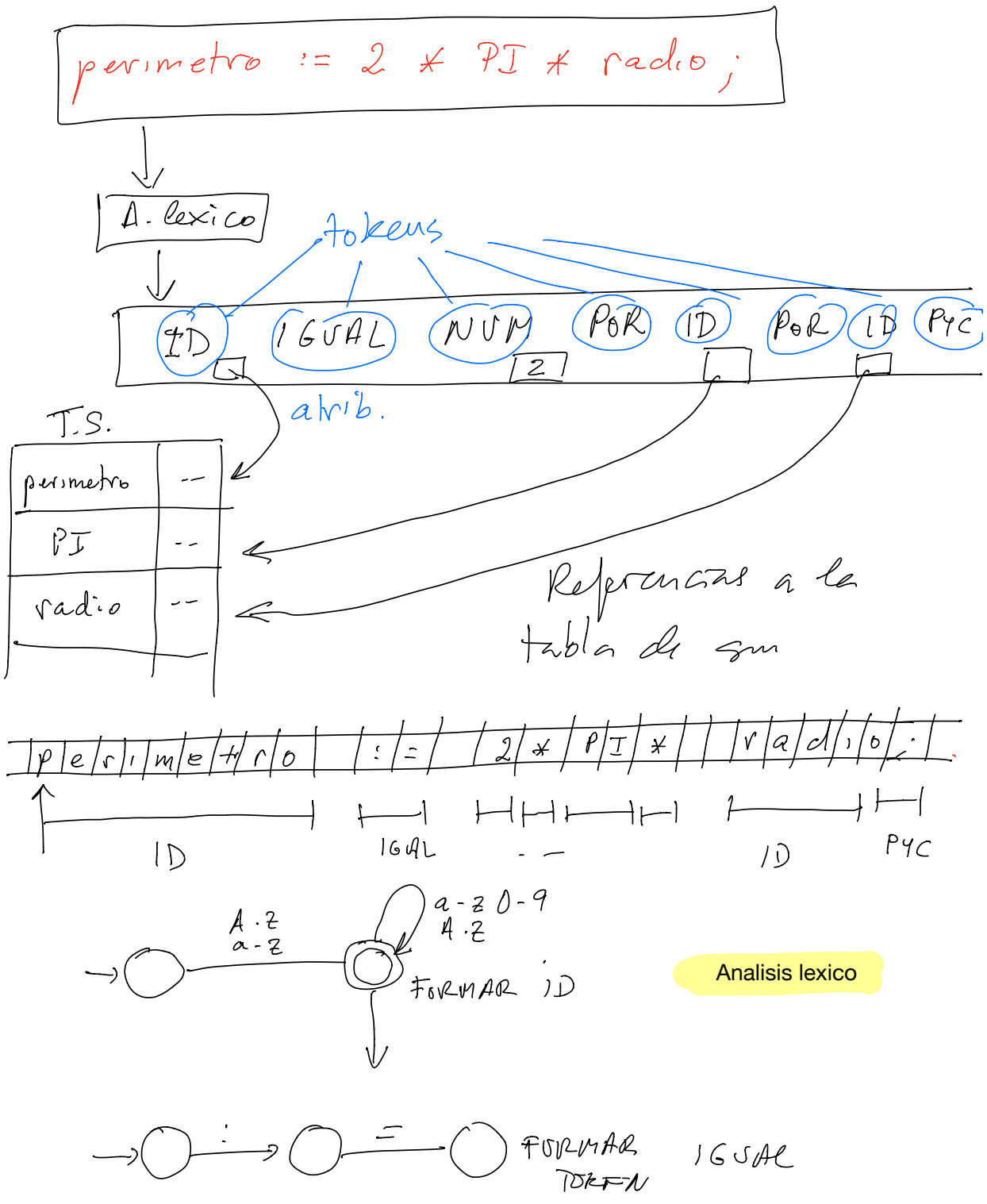




gcc -o sort.exe sort*.c

gcc -c sort1.c
↳ sort1.obj

ar sort*.obj → lib.a / lib.so

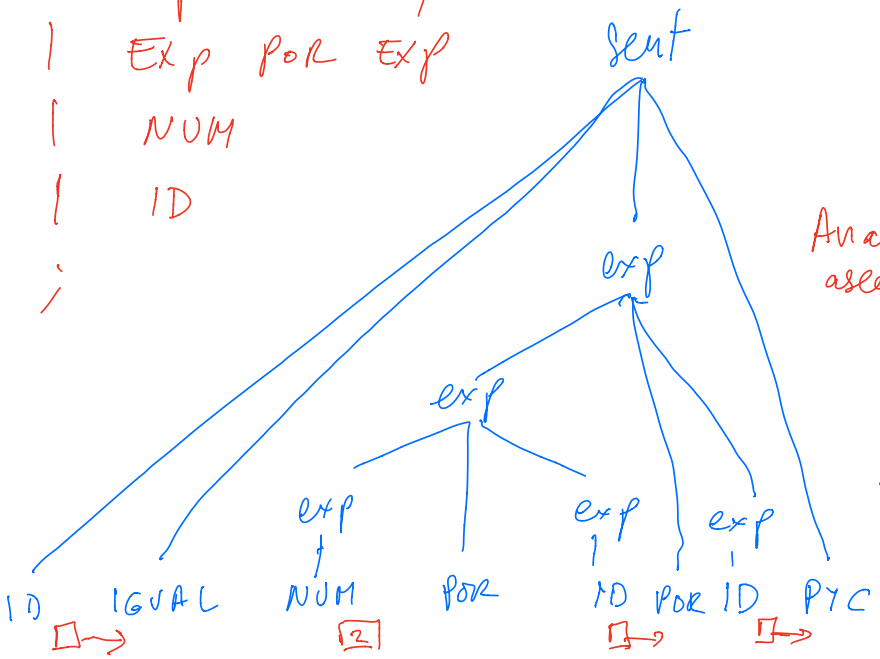


Analisis lexico

Analisis sintactico

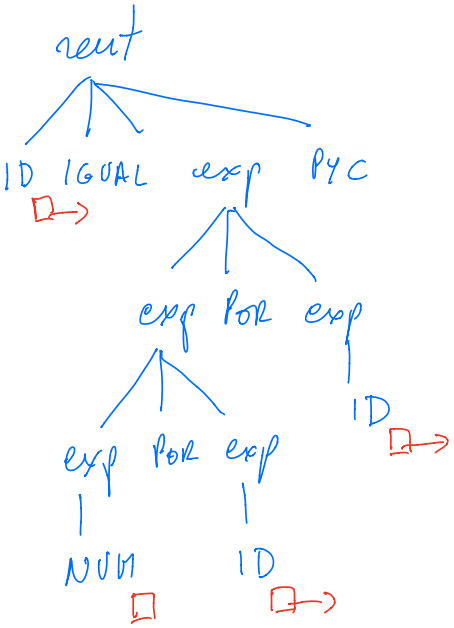
Sent → ID IGUA Exp PyC

Exp → Exp MAS Exp
| Exp POR Exp
| NUM
| ID
;



Analisis ascendente

SLR(1)



Analisis descendente

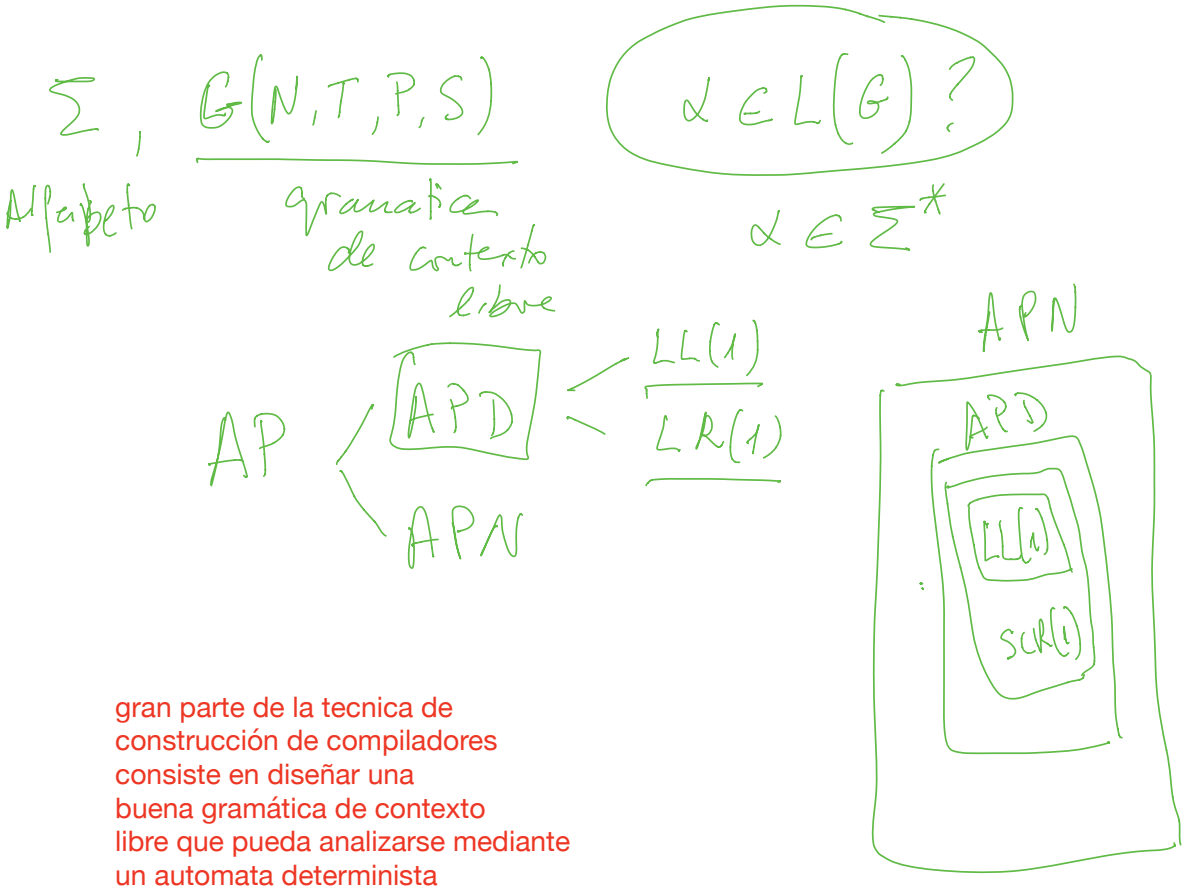
LL(1)

Sent → ID IGUAL Exp PyC

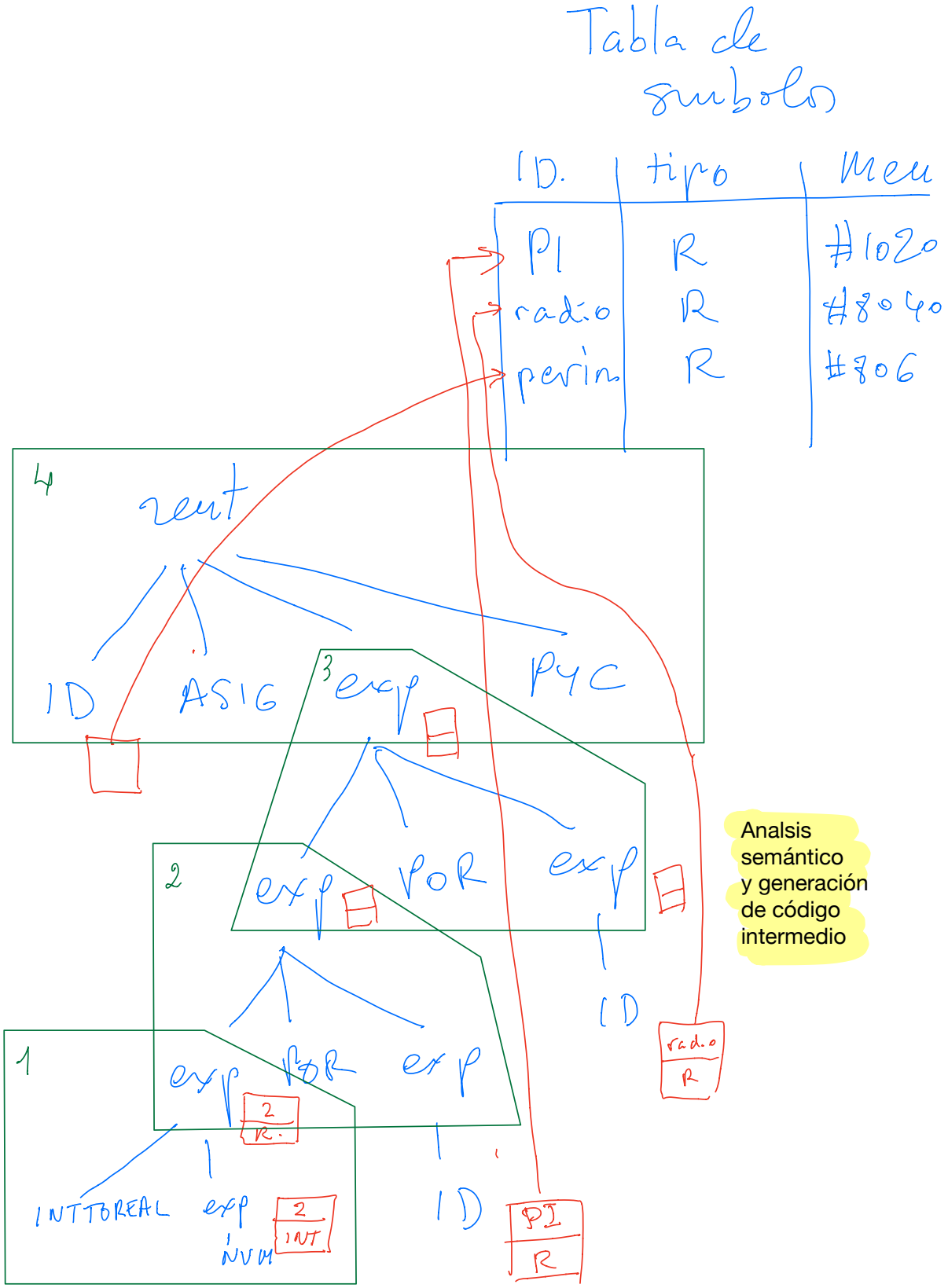
Exp → Exp MAS Termino
| Termino

Termino → Termino Por Factor
| Factor

Factor → NUM
| ID



gran parte de la tecnica de construcción de compiladores consiste en diseñar una buena gramática de contexto libre que pueda analizarse mediante un automata determinista



Analsis semántico y generación de código intermedio

Representación interna = Representación legible

	op	arg1	arg2	res
1	INTTOR	2		t1
2	POR	t1		t2
3	POR	t2		t3
4	ASIG	t3		

punteros

```
t1 = inttoreal(2)
t2 = t1 * PI
t3 = t2 * radio
perimetro = t3
```

Tabla de símbolos

ID	Posición en memoria
PI	#1020
radio	#8040
perimetro	#806

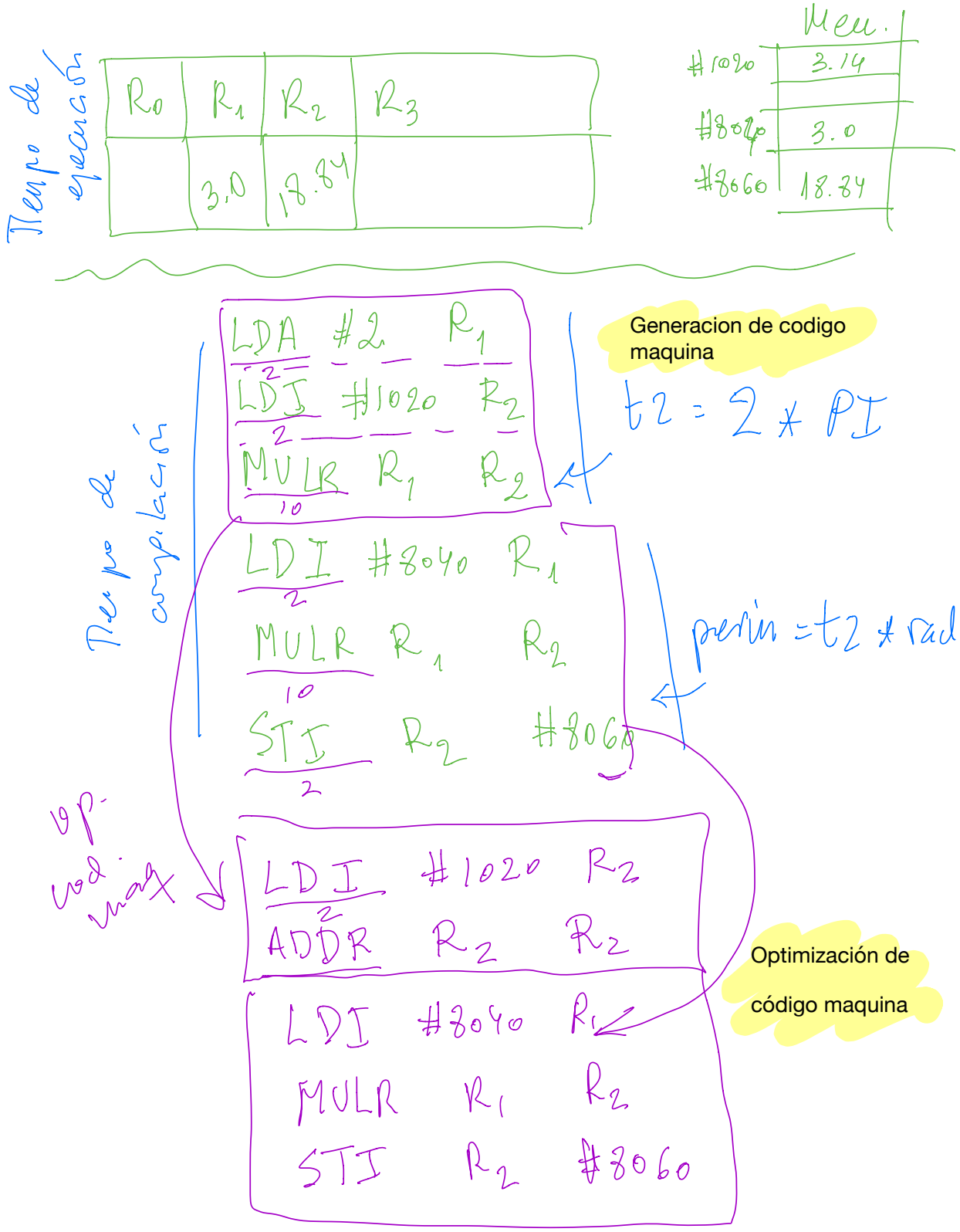
```
t1 = 2.0
t2 = t1 * PI
```

→ `t2 = 2 * PI`

```
t3 = t2 * radio
perimetro = t3
```

Optimización de código intermedio

```
t2 = 2 * PI
perimetro = t2 * radio
```


Tiempo de ejecución
R0 R1 R2 R3
3.0 18.84
Mem.
#1020 3.14
#8040 3.0
#8060 18.84
LDA #2 R1
2
LDI #1020 R2
2
MULR R1 R2
10
Generacion de codigo maquina
t2 = 2 * PI
Tiempo de compilación
LDI #8040 R1
2
MULR R1 R2
10
STI R2 #8060
2
perim = t2 * radio
Op. cod. maq
LDI #1020 R2
2
ADDR R2 R2
LDI #8040 R1
MULR R1 R2
STI R2 #8060
Optimización de código maquina