Febrero-2014.pdf

CarlosGarSil98

Algorítmica y Modelos de Computación

3º Grado en Ingeniería Informática

Escuela Técnica Superior de Ingeniería
Universidad de Huelva

APELLIDOS, NOMBRE García Silva, Carlos ____ NOTA ____

Ejercicio_1. (2 puntos)

El algoritmo de ordenación MergeSort puede ser implementado por:

```
MergeSort (A, p, r) /* Ordena un vector A desde p hasta r */
    if p<r { /* Dividir en dos trozos de tamaño igual (o lo más parecido posible), es decir ⌈n/2⌉ y ⌊n/2⌋ */
        q = ⌊(p+r)/2⌋ ;            /* Divide */
        /* Resolver recursivamente los subproblemas */
        MergeSort (A,p,q);         /* Resuelve */
        MergeSort (A,q+1,r) ;      /* Resuelve */
        /* Combinar: mezcla dos listas ordenadas en O(n) */
        Merge (A,p,q,r);           /* Combina*/
    }
```

- El algoritmo utiliza para su implementación la función Merge que tiene un coste $\Theta(n)$.

Se pide:

a. (0,75 puntos). Calcular la complejidad del algoritmo propuesto por el método de la **ecuación característica**.
b. (0,5 puntos). Calcular la complejidad del algoritmo propuesto por el Teorema maestro.
c. (0,75 puntos). Comparar el algoritmo propuesto con el siguiente (Calcular la complejidad y compararlas):

```
MergeSort_bad (A, p, r) /* Ordena un vector A desde p hasta r */
    if p<r {
        MergeSort_bad (A,p,p);
        MergeSort_bad (A,p+1,r) ;
        Merge (A,p,p,r);
    }
```

NOTA: El Teorema maestro es:

$$T(n) \in \begin{cases} O(n^{\log_b a}) & \text{si } a > b^k \\ O(n^k \cdot \log^{p+1} n) & \text{si } a = b^k \\ O(n^k \cdot \log^{p} n) & \text{si } a < b^k \end{cases}$$

Apartado a:

Observando el código podemos ver cómo, en función de si se cumple la condición del if, tendremos dos casos:

$p<r \longrightarrow n>1: T(n) = 1 + 3 + 2T(n/2) + 2 + \underbrace{n}_{\text{coste Merge} = n}$

$p \geq r \longrightarrow n \leq 1: T(n) = 1$

$$T(n) \begin{cases} 1 & \text{si } n \leq 1 \\ 2T(n/2) + n + 6 & \text{si } n > 1 \end{cases}$$

Obviamos el 6 ya que siempre será de orden O(1), y siempre se realizará.

$T(n) - 2T(n/2) = n \begin{bmatrix} \text{cambio base} \\ n = 2^k \end{bmatrix} T(2^k) - 2T(2^{k-1}) = 2^k$ (No homogénea)

$T(2^k) - 2T(2^{k-1}) \longrightarrow (x-2)$
$b^k \cdot p(k)^d = 2^k \longrightarrow (x-2)^{0+1}$

$p(x) = (x-2)(x-2) = 0$; Raices: $r_1 = 2$ doble

$T(2^k) = c_0 \cdot 2^k \cdot k^0 + c_1 \cdot 2^k \cdot k \begin{bmatrix} \text{cambio base} \\ 2^k = n \end{bmatrix} T(n) = n c_0 + n \log_2(n) \cdot c_1 \in O(n \log(n))$

Apartado b:

según la estructura del teorema maestro:
$T(n) = a T(n/b) + O(n^k \log^p(n))$

En nuestro caso:
$T(n) = 2T(n/2) + n \longrightarrow a = 2,\ b = 2,\ k = 1,\ p = 0$

$a > b^k$; $2 > 2^1$; No se cumple
$a = b^k$; $2 = 2^1$; Sí se cumple

Por tanto, a partir del teorema maestro: $T(n) \in O(n \log(n))$

Apartado c:

$$T(n) \begin{cases} 1 & \text{si } n \leq 1 \\ T(n-1) + n & \text{si } n > 1 \end{cases}$$

$T(n) - T(n-1) = n$ (No homogénea)
$T(n) - T(n-1) \longrightarrow (x-1)$
$b^k \cdot p(n)^d = 1^n \cdot n$; $b = 1,\ d = 1 \longrightarrow (x-1)^{1+1}$

$p(x) = (x-1)^3$; Raices: $r_1 = 1$ (triple)

$T(n) = c_0 \cdot 1^n + c_1 \cdot 1^n \cdot n^1 + c_2 \cdot 1^n \cdot n^2 \in O(n^2)$

Para comparar usamos el método de cálculo de límites:

$$\lim_{n \to \infty} \frac{T_a}{T_b} = \frac{n^2}{n \log(n)} = +\infty$$

Por tanto, el segundo algoritmo crece más rápido, lo que quiere decir:

$T_b \in O(T_a)$
$T_a \in \Omega(T_b)$

Es el momento de asegurar tu futuro.

Sueldo+Pagas

Conciliación

Estabilidad

Horario intensivo

Tiempo libre

¡El esfuerzo para ser funcionario merece la pena!

Más info sobre tus oposiciones.

Jesús Ayala
Academia de Oposiciones

Ejercicio_2. (3 puntos)

Para aprobar una asignatura el/la estudiante tiene que hacer en total **n** tareas (exámenes, prácticas, trabajos, etc). Para cada una de ellas estima que le llevará cierto tiempo, t_i. Como puede realizarlas a lo largo del curso, las quiere repartir entre las convocatorias de junio, septiembre y diciembre. En cada convocatoria puede sacar **M** unidades de tiempo como **máximo**. Se supone que todas las tareas se deben hacer y que no se pueden fraccionar (cada tarea va a una sola convocatoria). El **objetivo** es conseguir un reparto haciendo las tareas cuanto antes, no dejarlas para el final, es decir, minimizar el tiempo dedicado en la convocatoria de diciembre. En caso de empate en diciembre, minimizar el tiempo en la de septiembre.

a. (1 punto). Diseñar un algoritmo **voraz** para resolver el problema aunque no se garantice siempre la solución óptima. Proponer y contrastar dos criterios de selección.
   Aplicar el algoritmo al caso: **n** = 5, **M** = 16, **t** = {7, 5, 3, 5, 6}.
b. (1 punto). Resolver el problema mediante **programación dinámica**. Definir la ecuación recurrente, los casos base, las tablas y el algoritmo para rellenarlas. No hay que aplicar el ejemplo.
c. (1 punto). Resolver el problema por **backtracking** usando el esquema iterativo. Indicar cómo es la representación de la solución, la forma del árbol y programar las funciones genéricas del esquema correspondiente.

NOTA: El esquema iterativo de backtracking puede ser implementado por:

```
Backtracking (var s: TuplaSolución)
    nivel:= 1
    s:= sINICIAL
    fin:= false
    repetir
        Generar (nivel, s)
        si Solución (nivel, s) entonces
            fin:= true
        sino si Criterio (nivel, s) entonces
            nivel:= nivel + 1
        sino mientras NOT MasHermanos (nivel, s) hacer
            Retroceder (nivel, s)
    hasta fin
```

Apartado a:

Nos planteamos dos criterios:
- Elegir primero las que menos tiempo requieren
- Elegir primero las que más tiempo requieren

Como el objetivo es minimizar el tiempo de la convocatoria de diciembre, usaremos el primer criterio, para hacer antes el mayor número de actividades.

$N = 5$; $M = 16$; $t = (7, 5, 3, 5, 6)$
$s = ((0,0,0,0,0), (0,0,0,0,0), (0,0,0,0,0))$
tiempoEmpleado = $(0, 0, 0)$
tOrdenado = $(3, 2, 4, 5, 1)$ Representa los índices de t

```
funcion convocatorias(M, N : enteros, t : vector) : vector
    S = matriz de 0's de tamaño 3·N
    tiempoEmpleado = (0, 0, 0)
    convocatoria = 1;
    i = 1;
    tOrdenada = ordenar(t);  Ordena de menor a mayor y devuelve los índices
    C = tOrdenada(i);
    mientras convocatoria ≤ 3 AND i ≤ N hacer
        si tiempoEmpleado(convocatoria) + t(C) ≤ M entonces
            tiempoEmpleado(convocatoria) = tiempoEmpleado(convocatoria) + t(C);
            S(convocatoria, C) = 1;
            i = i + 1;
            si i ≤ N;
                C = tOrdenada(i);
            fsi
        sino
            convocatoria = convocatoria + 1;
        fsi
    fmientras
    si i > N
        devuelve S;
    sino
        devuelve error;
    fsi
ffuncion
```

Aplicamos el algoritmo para el caso:

c=3; t(c)=3; tiempoEmpleado(1) + 3 == 3 ≤ M ⟶ True;
s = ((0,0,1,0,0), (0,0,0,0,0), (0,0,0 0 0))

c=2; t(c)= 5; tiempoEmpleado(1) + 5 == 8 ≤ M ⟶ True;
s = ((0,1,1,0,0), (0,0,0,0,0), (0,0,0 0 0))

c=4; t(c)= 5; tiempoEmpleado(1) + 5 == 13 ≤ M ⟶ True;
s = ((0,1,1,0,0), (0,0,0,0,0), (0,0,0 0 0))

c=5; t(c)= 6; tiempoEmpleado(1) + 6 == 15 ≤ M ⟶ False;

c=5; t(c)= 6; tiempoEmpleado(2) + 6 == 6 ≤ M ⟶ True;
s = ((0,1,1,0,0), (0,0,0,0,1), (0,0,0 0 0))

c=1; t(c)= 7; tiempoEmpleado(2) + 7 == 13 ≤ M ⟶ True;
s = ((0,1,1,0,0), (1,0,0,0,1), (0,0,0 0 0))

Apartado b·

Tomamos como base la decisión de añadir la tarea a la convocatoria actual o a una anterior. Para minimizar el número de tareas (horas) en la última convocatoria

Ecuación recurrente:

$$\text{convocatorias}(M,N,C) = \begin{cases} 0 & \text{si } N=0 \\ +\infty & \text{si } C=0 \\ \min(\text{convoc}(M,N-1,c)+t_N\ ,\ \text{convoc}(M,N,c-1)) & \text{si } N>1 \end{cases}$$

caso base:

si no hay actividades, el número de horas dedicadas en la última convocatoria será 0.
si no hay convocatorias, será una situación de error, expresada con $+\infty$

Tabla a generar:

sería una tabla de tantas filas como tareas haya añadiendo una para contemplar la posibilidad de no haber tareas
Habrá tantas columnas como convocatorias, con una más para representar que hay convocatorias.

	0	1	2	3
0	$+\infty$	0	0	0
1	$+\infty$			
2	$+\infty$			
3	$+\infty$			
4	$+\infty$			
5	$+\infty$			

Algoritmo para rellenar tabla:

```
funcion convocatoria (T(1...N))
    actual = 0;
    anterior = 0;
    para i=0 hasta N  Tabla(i)(0) = infinito  fpara
    para i=0 hasta C  Tabla(0)(i) = 0         fpara
    para i=1 hasta N hacer
        para j=1 hasta C hacer
            actual = Tabla(i-1)(j);
            si actual > M entonces
                anterior = 0;
            fsi
            Tabla(i)(j) = minimo(actual, anterior);
        fpara
    fpara
ffuncion
```

Apartado c:

Representación de la solución:

Para cada tarea, en principio, podría ir en cada una de las convocatorias.
Los nodos del árbol definirán internamente las asignaciones, de tantos elementos como convocatorias haya; y los valores que tomarán serán las convocatorias asignadas.

La solución viene en forma de lista:
$S = (O_1, \dots, O_N)$ solución
M capacidad máxima de tiempo por convocatoria
N Número de tareas
C Número de convocatorias
convocatorias $(O_1, \dots, O_C)$ tiempo empleado en cada convocatoria
tiempos $(1 \dots N)$ tiempo de cada tarea

Forma del árbol:

El árbol que se expandiría estaría formado a su vez por tres hijos. Habrá tantos niveles como tareas. Podría darse que si una convocatoria está llena en el momento de asignar una tarea, esa convocatoria quedaría descartada automáticamente.

Funciones genéricas:

```
procedimiento Backtracking (Tiempos (1... N))
    nivel = 1;
    inicialización de S
    inicialización de convocatorias
    fin = false;
    repetir
        Generar (nivel, S);
        si solucion (nivel, S) entonces
            fin = true;
        sino
            si criterio (nivel, S) entonces
                convocatorias (S (nivel)) = convocatorias (S (nivel)) + tiempos (nivel);
                nivel = nivel + 1;
            sino
                mientras NOT MasHermanos (nivel, S) hacer
                    Retroceder (nivel, S);
                fmientras
            fsi
        fsi
    hasta fin
fprocedimiento
```

```
funcion Generar(nivel, s)
    s(nivel) = s(nivel) + 1;
ffuncion

funcion solucion(nivel, s)
    devuelve nivel == N AND s(nivel) != 0;
ffuncion

funcion criterio(nivel, s)
    devuelve nivel ≤ N AND convocatorias(s(nivel)) + tiempos(nivel) ≤ M;
ffuncion

funcion MasHermanos(nivel, s)
    devuelve s(nivel) ≤ C;
ffuncion

funcion Retroceder(nivel, s)
    s(nivel) = 0;
    nivel = nivel - 1;
ffuncion
```

APELLIDOS, NOMBRE García Silva, Carlos NOTA

Ejercicio_3. (1,5 puntos)

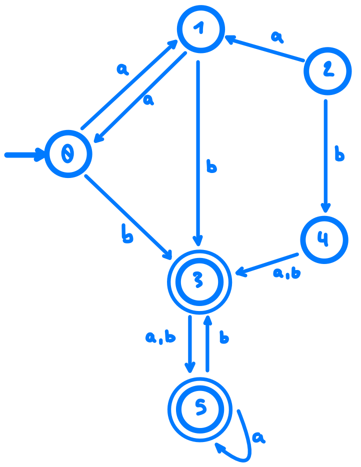
Dado el autómata siguiente,

f	a	b
→0	1	3
1	0	3
2	1	4
*3	5	5
4	3	3
*5	5	3

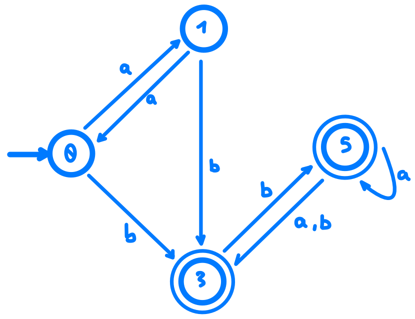
Obtener:

a. (0.5 puntos). El A.F.D. mínimo equivalente.
b. (0.5 puntos). La expresión regular del lenguaje reconocido por el autómata del apartado anterior.
c. (0.5 puntos). La gramática de tipo 3 para el lenguaje.

Apartado a:



Como podemos observar, los estados 2 y 4 no son accesibles desde otro estado, por tanto, pueden quitarse:



Agrupamos en estados no finales y finales

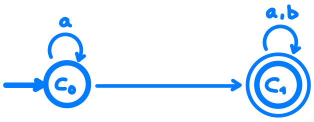
$Q/E_0 = (C_0 = \{0,1\}, C_1 = \{3,5\})$

$f(0,a) = C_0 \quad f(0,b) = C_1$
$f(1,a) = C_0 \quad f(1,b) = C_1$ — coinciden, no divisible

$f(3,a) = C_1 \quad f(3,b) = C_1$
$f(5,a) = C_1 \quad f(5,b) = C_1$ — coinciden, no divisible

El AFD mínimo sería:

f	a	b
→ C_0	C_0	C_1
* C_1	C_1	C_1



Apartado b:

Ecuación característica
$$\begin{cases} X_0 = aX_0 + bX_1 + b \\ X_1 = aX_1 + bX_1 + a + b \end{cases}$$

Hallamos valor de X_1:

$X_1 = aX_1 + bX_1 + a + b$;

$X_1 = (a+b)^* (a+b)$

sustituimos en la otra ecuacion:

$X_0 = aX_0 + bX_1 + b$;

$X_0 = aX_0 + b((a+b)^* (a+b)) + b$

$X_0 = a^*(b((a+b)^* (a+b)) + b)$

Apartado c:

$G = \langle \Sigma_T, \Sigma_N, S, P \rangle$

$G = \langle \{C_0, C_1\}, \{a, b\}, C_0, P \rangle$

$$P = \begin{cases} C_0 ::= aC_0 \mid bC_1 \mid b \\ C_1 ::= aC_1 \mid bC_1 \mid a \mid b \end{cases}$$

APELLIDOS, NOMBRE García Silva, Carlos NOTA

Ejercicio_4. (1,5 puntos)

Dado el lenguaje $(01)^n$ con $n \geq 0$.

1) (0,75 puntos). Seleccionar, **justificando la respuesta**, el autómata que reconoce el lenguaje indicado.
   a. AF=[{0,1}, {A,B,C,F}, f, A, {F}] con f(A,0)=B, f(A, λ)= λ, f(C,0)=B, f(B,1)=C, f(B,1)= λ
   b. AF=[{0,1}, {A,B,C,F}, f, A, {F}] con f(A,0)=B, f(A, λ)=F, f(C,0)=B, f(B,1)=C, f(B,1)=F
   c. AF=[{0,1}, {A,B,C,F}, f, A,{F}] con f(A, B)=0, f(A,F)= λ, f(C,B)=0, f(B,C)=1, f(B,F)=1
   d. AF=[{0,1}, {A,B,C,F}, f, A, {F}] con f(B,0)=A, f(F, λ)=A, f(B,0)=C, f(C,1)=B, f(F,1)=B
2) (0,75 puntos). Obtener el AFD mínimo equivalente del autómata seleccionado en el apartado anterior.

Apartado 1:

a) No reconoce el lenguaje ni ningún otro, ya que no posee transición al estado final

b) Si presenta transiciones al estado final, vamos a ver si son accesibles:

$f(A,\lambda) = F$, la cadena vacía la acepta

$f(A,0) = B \rightarrow f(B,1) = F$; Acepta la cadena 01
$f(A,0) = B \rightarrow f(B,1) = C \rightarrow f(C,0) = B \rightarrow f(B,1) = F$
Acepta las cadenas 0101...
Podemos decir que el automata reconoce el lenguaje

c) No reconoce el lenguaje, tiene mal definidas las transiciones. De un símbolo no terminal con otro no terminal no se puede hacer una transición a otro símbolo terminal.

d) No reconoce el lenguaje, no hay transiciones que partan del estado inicial

Apartado 2:

$\xrightarrow{*} Q_0 = A, F$
$f'(Q_0, 0) = B$ Q_1 estado Normal
$f'(Q_0, 1) = \emptyset$

$Q_1 = B$
$f'(Q_1, 0) = \emptyset$
$f'(Q_1, 1) = C, F$ Q_2 estado final

$* Q_2 = C, F$
$f'(Q_2, 0) = B \rightarrow Q_1$
$f'(Q_2, 1) = \emptyset$

	0	1
$\xrightarrow{*} Q_0$	Q_1	M
Q_1	M	Q_2
$* Q_2$	Q_1	M
M	M	M

Agrupamos entre estados no finales y finales:

$Q/E_0 = \left(C_0 = (Q_1), C_1 = (Q_0, Q_2), C_2 = (M) \right)$

$$\left.\begin{array}{ll} f'(Q_0, 0) = C_0 & f'(Q_0, 1) = C_2 \\ f'(Q_2, 0) = C_0 & f'(Q_2, 1) = C_2 \end{array}\right\} \text{coinciden, se mantienen juntos}$$

Ya hemos obtenido el autómata mínimo

	0	1
$\rightarrow * C_1$	C_0	M
C_0	M	C_1
M	M	M

Ejercicio_5. (2 puntos)

Dada la gramática:

$$S \to S = A \mid A$$
$$A \to A = B \mid B$$
$$B \to (S) \mid a \mid b$$

a. (0.5 puntos). Comprobar si es LL(1), eliminar la recursividad a la izquierda y obtener la gramática LL(1) equivalente.

b. (0.5 puntos). Convertir la gramática del apartado anterior en un autómata con pila que acepte el mismo lenguaje por pila vacía.

c. (0.5 puntos). Analizar, teniendo en cuenta el principio de preanálisis (lectura de un símbolo de la entrada con anticipación) la entrada **"(a)"**.

d. (0.5 puntos). Implementar el seudocódigo de análisis descendente dirigido por la sintaxis para la gramática obtenida LL(1)

Apartado a:

$S \to S{=}A \mid A$ | $S \to AS'$, $S' \to {=}AS' \mid \lambda$

$A \to A{:=}B \mid B$ | $A \to BA'$, $A' \to {:=}BA' \mid \lambda$

Gramática equivalente resultante:

1. $S \to AS'$
2. $S' \to {=}AS'$
3. $\quad \mid \lambda$
4. $A \to BA'$
5. $A' \to {:=}BA'$
6. $\quad \mid \lambda$
7. $B \to (S)$
8. $\quad \mid a$
9. $\quad \mid b$

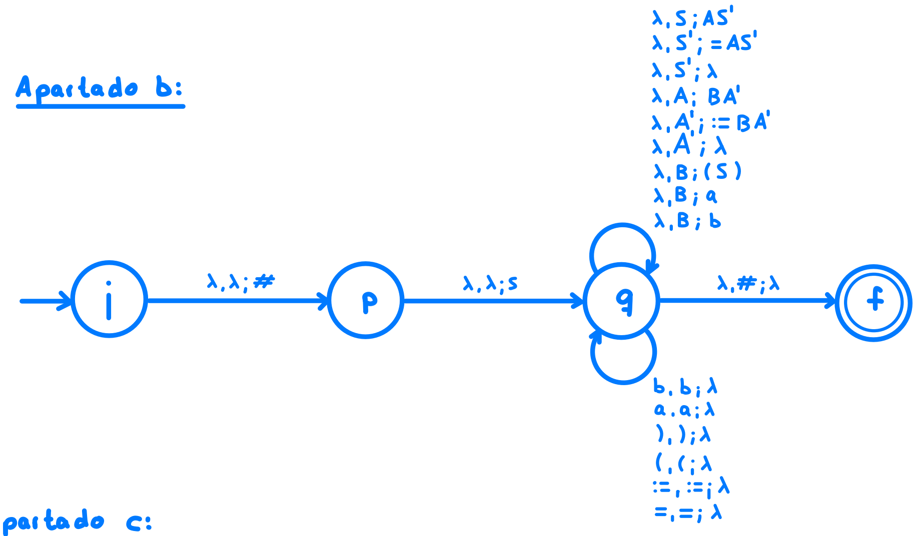
Para comprobar vamos a hacer uso de la tabla: primeros, siguientes, predicción:

	Primeros	Siguientes	Predicción
S	(, a , b	) , $	(, a , b
S'	=	) , $	=
	λ		) , $
A	(, a , b	= ,) , $	(, a , b
A'	:=	= ,) , $	:=
	λ		= ,) , $
B	(	:=	(
	a		a
	b		b

intersección vacía (S')
intersección vacía (A')
intersección vacía (B)

Como todas las intersecciones son vacías,
podemos decir que nos encontramos con la gramática equivalente LL(1).

Apartado b:



Apartado c:

Estado	Pila	Entrada	Acción	indeterminación	Acción
i	λ	(a) $	i λ λ;p #		
p	#	(a) $	p λ λ;q S		
q	S #	(a) $	q λ S;q A S'		S::= AS'
q	A S' #	(a) $	q λ A;q B A'		A::= BA'
q	B A' S' #	(a) $	q λ B;q (S)		B::= (S)
q	(S) A' S' #	(a) $	q ((;q λ		Reconoce (()
q	S) A' S' #	a) $	q λ S;q AS'		S::= AS'
q	A S') A' S' #	a) $	q λ A;q BA'		A::= BA'
q	B A' S') A' S' #	a) $	q λ B;q a		B::= a
q	a A' S') A' S' #	a) $	q a a;q λ		Reconoce (a)
q	A' S') A' S' #	) $	q λ A';q := BA'	q λ A';q λ	A' ::= λ
q	S') A' S' #	) $	q λ S';q = AS'	q λ S';q λ	S'::= λ
q	) A' S' #	) $	q));q λ		Reconoce ())
q	A' S' #	$	q λ A';q := BA'	q λ A';q λ	A' ::= λ
q	S' #	$	q λ S';q = AS'	q λ S';q λ	S'::= λ
q	#	$	q λ #;f λ		
f	λ	λ	Aceptar		

Apartado d:

Vamos a definir el símbolo leído del preanálisis como: símbolo SLA

```
programa_Principal ()
    SLA = leer_simbolo();
    S();
    si SLA != $ entonces
        Error ();
    fsi
fprograma
```

```
procedimiento S()
    A();
    S'();
fprocedimiento
```

```
procedimiento S'()
    switch SLA
        case =
            Reconocer (=);
            A();
            S'();
        case ), $

        default
            error_sintactico();
    fswitch
fprocedimiento
```

```
procedimiento Reconocer(simbolo T)
    si SLA == T entonces
        leer_simbollo();
    sino
        error_sintactico();
    fsi
fprocedimiento
```

```
procedimiento A()
    B();
    A'();
fprocedimiento
```

```
procedimiento A'()
    switch SLA
        case :=
            Reconocer(:=);
            B();
            A'();
        case =, ), $

        default
            error_sintactico();
    fswitch
fprocedimiento
```

```
procedimiento B()
    switch SLA
        case (
            Reconocer (();
            S();
            Reconocer ());
        case a
            Reconocer (a);
        case b
            Reconocer (b);
        default
            error_sintactico();
    fswitch
fprocedimiento
```