Lenguajes de Programación
y
Procesadores de Lenguajes

Ejercicios curso 2020 - 2021

José Miguel Benedí
<jbenedi@dsic.upv.es>

28 de noviembre de 2020

Índice

2. Análisis Léxico

1. Dado el alfabeto de los dígitos proporcionad una expresión regular que defina el lenguaje de todas as cadenas de dígitos que empiecen y terminen por 2.

2. * Dado el alfabeto de los dígitos proporcionad una expresión regular que defina el lenguaje de todas las cadenas de dígitos que representen números pares.

3. * Dado el alfabeto $\{a, b, c\}$, proporcionad una expresión regular que defina el lenguaje de todas las cadenas que contengan exactamente una a.

4. * Dado el alfabeto $\{a, b, c\}$, proporcionad una expresión regular que defina el lenguaje de todas las cadenas que no contengan dos a consecutivas.

5. * Dado el alfabeto $\{a, b, c\}$, proporcionad una expresión regular en la que la primera aparición de b es siempre precedida por al menos una aparición de a.

6. * Dado el alfabeto $\{a, b, c\}$, proporcionad una descripción concisa en Castellano del lenguaje que genera la expresión regular: $((a|b)^*\ c\ (a|b)^*\ c)^*\ (a|b)^*$.

7. * Proporcionar una expresión regular en la que todas las cadenas de letras minúsculas deben contener las cinco vocales en orden (cada vocal solo debe aparecer una vez); por ejemplo: `k j a h l e i b o k j u m`

8. Dibujad un AEF para las expresiones regulares de los ejercicios 2 a 8.

9. * Dada la siguiente cadena de entrada `babcaababccbcabb` vuestro AL detecta los siguientes tokens: `b abc aab abcc b c abb`

   Proporcionad una especificación léxica en términos de expresiones regulares. Emplead solo las operaciones de concatenación y cierre transitivo y reflexivo.

10. Diseñad una expresión regular para las siguientes especificaciones en lenguaje natural:

    a) La dirección de un correo electrónico.

    b) La dirección de una página web.

11. * Describid brevemente las principales funciones de un Analizador Léxico e indicad cómo se puede tratar el problema de la detección de las palabras reservadas.

12. * Construid una *expresión regular* (o en formato FLEX, si se quiere) para las constantes numéricas, que permita, por ejemplo: `3.14  3.  3  0.14`

13. * Construid una *expresión regular* en formato FLEX para los números sin signo del lenguaje PASCAL.

14. Escribid un programa FLEX que sustituya las palabras reservadas `while, do, if, else` a mayúsculas. Debe generar un fichero igual al de la entrada con la única diferencia del cambio de dichas palabras reservadas.

15. Construid una expresión regular para las constantes numéricas decimales; por ejemplo: `2.27  -4e7  +14.0e-4  .127e+7`

16. Diseñad las expresiones regulares para las constantes numéricas de tu lenguaje de programación favorito.

17. Construid un programa FLEX para la especificación léxica de tu lenguaje de programación favorito.

18. Construid un programa FLEX para la especificación léxica del lenguaje `MenosC.18` (consultad el enunciado del proyecto).

2.1. Ejercicios resueltos

Ejercicio 2 $digito \quad [0-9]$

$pares \quad [02468]$

$\{digito\}* \ \{pares\}$

Ejercicio 3 $[bc]* \ a \ [bc]*$

Ejercicio 4 $[bc]* \ (\ (a \ [bc]+)* \ | \ (a \ [bc]*) \ )$

Ejercicio 5 $[ac]* \ ab \, [abc]* \ | \ [ac]*$

Ejercicio 6 Cadenas con un número par de c.

Ejercicio 7 $noV \quad [\wedge aeiou]$

$\{noV\}* \ a \ \{noV\}* \ e \ \{noV\}* \ i \ \{noV\}* \ o \ \{noV\}* \ u \ \{noV\}*$

Ejercicio 9 Dos posibles soluciones correctas

1	2
c	$aa^*b^*c^*$
b	c^*b^*
$aa^*b^*c^*$	

Ejercicio 11 Las principales funciones de un Analizador Léxico (AL) son la detección de los símbolos del lenguaje y la realización de las acciones semánticas asociadas con la detección de un símbolo, así como la emisión de las componentes léxicas ("tokens") detectadas.

Además el AL también realiza otras funciones: 1) Tratamiento de errores léxicos; 2) eliminación de cadenas inútiles (comentarios, tabuladores, saltos de línea, etc.); 3) lectura eficiente del fichero de entrada; 4) relación de los mensajes de error con las líneas del programa fuente; y 4) reconocimiento y ejecución de las directivas de compilación.

Las palabras reservadas pueden tratarse, o bien con expresiones regulares independientes, o bien, como identificadores especiales (tabla de palabras reservadas).

Ejercicio 12 $digito \quad [0-9]$

$\{digito\}+ \ (\text{“.”} \ \{digito\}*)?$

Ejercicio 13

$$
\begin{array}{ll}
digito & [0-9] \\
digitos & \{digitos\}+ \\
fracOpc & (\text{“.”}\{digitos\})? \\
expoOpc & (\text{“}E\text{”}([+-])?\{digitos\})? \\
numeros & \{digitos\}\{fracOpc\}\{expoOpc\}
\end{array}
$$

3. Análisis Sintáctico

3.1. Introducción

19. Dado el alfabeto $\Sigma = \{a, b\}$, proporcionad una expresión regular y una gramática incontextual para el lenguaje formado por todas las cadenas que empiecen por a.

20. Proporcionad un ejemplo simple de lenguaje que sea incontextual y no sea regular.

21. * Dada la gramática que se muestra, calculad una secuencia de derivación a izquierdas para cada una de las siguientes tres cadenas: $\{\{i\}, \{i, i\}\}$ $\{i, i, i\}$ $\{i, \{i\}\}$

$$B \Rightarrow \{\ \mathrm{I}\ \} \mid \mathrm{i} \qquad\qquad I \Rightarrow \mathrm{I}\ ,\ \mathrm{B} \mid \mathrm{B}$$

22. * Para la misma gramática del ejercicio 21, calculad una secuencia de derivación a derechas para cada una de las mismas tres cadenas.

23. Construid el árbol de análisis sintáctico asociado a la primera cadena de los ejercicios 21 y 22. ¿Qué se puede decir de ambos y porqué?

24. Dada la siguiente gramática, calculad la secuencia de derivación a izquierdas y a secuencia de derivación a derechas para la cadena: $a\ a\ +\ a\ *$

$$S \Rightarrow \mathrm{S\ S}\ +\ \mid \mathrm{S\ S}\ *\ \mid a$$

Obtened un árbol de análisis para la cadena. ¿Es la gramática ambigua? justificad la respuesta.

25. Dada la siguiente gramática, calculad la secuencia de derivación a izquierdas y la secuencia de derivación a derechas para la cadena: $b\ c\ d\ c$

$$S \Rightarrow \mathrm{S\ B}\ \mid\ b \qquad\qquad B \Rightarrow \mathrm{B}\ d\ \mid\ c$$

26. `[A.Aho, R.Sethi, J.Ullman, 1990]` Dada la gramática que se muestra, demostrad que esta gramática es ambigua construyendo dos árboles de análisis sintáctico distintos para la cadena: *abab*

$$S \Rightarrow \mathrm{a\ S\ b\ S} \mid \mathrm{b\ S\ a\ S} \mid \epsilon$$

27. * Escribid una gramática no ambigua que genere el lenguaje de los palíndromos sobre el alfabeto $\{a, b\}$. Considerando esta gramática, obtened la secuencia de derivaciones a izquierdas para la cadena: $a\ b\ b\ a$

28. * Escribid una gramática no ambigua que genere el lenguaje de las expresiones booleanas que incluyan: operadores **and**, **or** y **not**; constantes **true** y **false**; y paréntesis (y). Considerad la precedencia: **or** menos precedencia que **and** y **and** menos que **not**. Considerando esta gramática, obtened la secuencia de derivaciones a derechas para la cadena: $not\ (true\ or\ false)$

29. * Dado el alfabeto $\Sigma = \{a, b\}$, escribid una gramática incontextual para cada uno de los siguientes lenguajes:

    a) Cadenas del lenguaje L:$\{a^{2n}b^m c^{3n} \mid n, m \geq 0\}$

b) Conjunto de todas las cadenas con más a que b.

30. Escribid una gramática para definir los números romanos menores que 100.

31. * Dado el autómata a pila que se adjunta, obtened la traza asociada (con el criterio: pila vacía) para la cadena $a+a*a$ tal que: $(q,\ a+a*a,\ E) \overset{*}{\vdash} (q,\ ,\ )$

$A_p = (Q, T, \Gamma, \delta, q, E, \emptyset)$ con $Q = \{q\}$; $T = \{a, +, *\}$; $\Gamma = \{E, T, F, a, +, *\}$

$\delta(q, a, a) = (q, \epsilon)$; $\delta(q, +, +) = (q, \epsilon)$; $\delta(q, *, *) = (q, \epsilon)$;

$\delta(q, \epsilon, E) = \{(q, E+T), (q, T)\}$; $\delta(q, \epsilon, T) = \{(q, T*F), (q, F)\}$; $\delta(q, \epsilon, F) = (q, a)$;

32. * Dado el autómata a pila que se adjunta, obtened la traza asociada (con el criterio: pila vacía) para la cadena $a+a*a$ tal que: $(q,\ a+a*a,\ \$) \overset{*}{\vdash} (q,\ ,\ )$

$A_p = (Q, T, \Gamma, \delta, q, \$, \emptyset)$ con $Q = \{q\}$; $T = \{a, +, *\}$; $\Gamma = \{E, T, F, a, +, *\} \cup \{\$\}$

$\delta(q, a, \epsilon) = (q, a)$; $\delta(q, +, \epsilon) = (q, +)$; $\delta(q, *, \epsilon) = (q, *)$;

$\delta(q, \epsilon, \$E) = (q, \epsilon)$; $\delta(q, \epsilon, E+T) = (q, E)$; $\delta(q, \epsilon, T) = (q, E)$;

$\delta(q, \epsilon, T*F) = (q, T)$; $\delta(q, \epsilon, F) = (q, T)$; $\delta(q, \epsilon, a) = (q, F)$;

33. Comparar el ejercicio 21 y el 31 y relacionarlo con el Análisis Sintáctico Descendente.

34. Análogamente, comparar el ejercicio 22 y el 32 y relacionarlo con el Análisis Sintáctico Ascendente.

3.1.1. Ejercicios resueltos

Ejercicio 21 Solo para la primera cadena:

$$B \Rightarrow \{I\} \Rightarrow \{I, B\} \Rightarrow \{B, B\} \Rightarrow \{\{I\}, B\} \Rightarrow \{\{B\}, B\} \Rightarrow \{\{i\}, B\} \Rightarrow \{\{i\}, \{I\}\}$$
$$\Rightarrow \{\{i\}, \{I, B\}\} \Rightarrow \{\{i\}, \{B, B\}\} \Rightarrow \{\{i\}, \{i, B\}\} \Rightarrow \{\{i\}, \{i, i\}\}$$

Ejercicio 22 Solo para la primera cadena:

$$B \Rightarrow \{I\} \Rightarrow \{I, B\} \Rightarrow \{I, \{I\}\} \Rightarrow \{I, \{I, B\}\} \Rightarrow \{I, \{I, i\}\} \Rightarrow \{I, \{B, i\}\} \Rightarrow \{I, \{i, i\}\}$$
$$\Rightarrow \{B, \{i, i\}\} \Rightarrow \{\{I\}, \{i, i\}\} \Rightarrow \{\{B\}, \{i, i\}\} \Rightarrow \{\{i\}, \{i, i\}\}$$

Ejercicio 27 Una posible gramática:

$$S \Rightarrow a\ \mathrm{S}\ a \mid b\ \mathrm{S}\ b \mid \epsilon$$

Y la secuencia de derivación a izquierdas es:

$$S \Rightarrow a\ S\ a \Rightarrow a\ b\ S\ b\ a \Rightarrow a\ b\ b\ a$$

Ejercicio 28 Una posible gramática:

$$\begin{array}{lcl} E & \Rightarrow & \mathrm{E}\ or\ \mathrm{T} \mid \mathrm{T} \\ T & \Rightarrow & \mathrm{T}\ and\ \mathrm{F} \mid \mathrm{F} \\ F & \Rightarrow & not\ \mathrm{I} \mid \mathrm{I} \\ I & \Rightarrow & true \mid false \mid (\ \mathrm{E}\ ) \end{array}$$

Y la secuencia de derivación a derechas es:

$$E \Rightarrow T \Rightarrow F \Rightarrow not\ I \Rightarrow not\ (\ E\ ) \Rightarrow not\ (\ E\ or\ T\ ) \Rightarrow not\ (\ E\ or\ F\ )$$
$$\Rightarrow not\ (\ E\ or\ I\ ) \Rightarrow not\ (\ E\ or\ false\ ) \Rightarrow not\ (\ T\ or\ false\ )$$
$$\Rightarrow not\ (\ F\ or\ false\ ) \Rightarrow not\ (\ I\ or\ false\ ) \Rightarrow not\ (\ true\ or\ false\ )$$

Ejercicio 29

1. Cadenas del lenguaje L:$\{a^{2n}b^mc^{3n} \mid n, m \geq 0\}$

$$S \Rightarrow \text{a a S c c c} \mid \text{B} \qquad\qquad B \Rightarrow \text{b B} \mid \epsilon$$

2. Conjunto de todas las cadenas con más a que b.

$$\begin{array}{lll} S & \Rightarrow & \text{A a} \mid \text{M S} \mid \text{S M A} \\ A & \Rightarrow & \text{A a} \mid \epsilon \\ M & \Rightarrow & \text{M M} \mid \text{b M a} \mid \text{a M b} \mid \epsilon \end{array}$$

Ejercicio 31

$(q,\ a+a*a,\ E) \vdash (q,\ a+a*a,\ E+T) \vdash (q,\ a+a*a,\ T+T) \vdash (q,\ a+a*a,\ F+T) \vdash$
$(q,\ a+a*a,\ a+T) \vdash (q,\ +a*a,\ +T) \vdash (q,\ a*a,\ T) \vdash (q,\ a*a,\ T*F) \vdash$
$(q,\ a*a,\ F*F) \vdash (q,\ a*a,\ a*F) \vdash (q,\ *a,\ *F) \vdash (q,\ a,\ F) \vdash (q,\ a,\ a) \vdash (q,\ ,\ )$

Ejercicio 32

$(q,\ a+a*a,\ \$) \vdash (q,\ +a*a,\ \$a) \vdash (q,\ +a*a,\ \$F) \vdash (q,\ +a*a,\ \$T) \vdash$
$(q,\ +a*a,\ \$E) \vdash (q,\ a*a,\ \$E+) \vdash (q,\ *a,\ \$E+a) \vdash (q,\ *a,\ \$E+F) \vdash$
$(q,\ *a,\ \$E+T) \vdash (q,\ a,\ \$E+T*) \vdash (q,\ ,\ \$E+T*a) \vdash (q,\ ,\ \$E+T*F) \vdash$
$(q,\ ,\ \$E+T) \vdash (q,\ ,\ \$E) \vdash (q,\ ,\ )$

3.2. Análisis Sintáctico Descendente

35. * Dada una gramática $\mathcal{G}$ bien definida (todos los no-terminales son accesibles y no hay símbolos inútiles), demostrad que: $\forall A \in N$, SIGUIENTE$(A) \neq \emptyset$.

36. * Calculad PRIMEROS y los SIGUIENTES de todos los símbolos no terminales de la gramática. Comprobar igualmente si cumple la condición LL(1).

$$\begin{array}{lcl} \text{S} & \Rightarrow & \text{a A B C} \\ \text{A} & \Rightarrow & \text{b b D} \mid \text{a} \end{array} \qquad \begin{array}{lcl} \text{B} & \Rightarrow & \text{a} \mid \epsilon \\ \text{C} & \Rightarrow & \text{b} \mid \epsilon \end{array} \qquad \begin{array}{lcl} \text{D} & \Rightarrow & \text{c} \mid \epsilon \end{array}$$

37. * Ídem para la siguiente gramática:

$$\begin{array}{lcl} \text{S} & \Rightarrow & \text{E D C} \\ \text{A} & \Rightarrow & \text{a} \mid \epsilon \end{array} \qquad \begin{array}{lcl} \text{B} & \Rightarrow & \epsilon \\ \text{C} & \Rightarrow & \text{c} \mid \epsilon \end{array} \qquad \begin{array}{lcl} \text{D} & \Rightarrow & \text{B} \mid \text{b b} \\ \text{E} & \Rightarrow & \text{A e} \mid \epsilon \end{array}$$

38. * Ídem para la siguiente gramática:

$$\begin{array}{lcl} \text{S} & \Rightarrow & \text{B C D} \\ \text{C} & \Rightarrow & \text{c C} \mid \epsilon \end{array} \qquad \begin{array}{lcl} \text{B} & \Rightarrow & \text{C D} \mid \text{D C} \\ \text{D} & \Rightarrow & \text{D d} \mid \text{a} \mid \epsilon \end{array}$$

39. Calculad los PRIMEROS y los SIGUIENTES de todos los símbolos no terminales de la gramática:

$$\begin{array}{lcl} \text{S} & \Rightarrow & \text{a S A} \mid \epsilon \\ \text{B} & \Rightarrow & \text{A c} \mid \epsilon \end{array} \qquad \begin{array}{lcl} \text{A} & \Rightarrow & \text{B b} \end{array}$$

40. Dada una gramática LL(1), demostrad las siguientes proposiciones:

    a) * Si una gramática es recursiva a izquierdas, entonces no es LL(1).

    b) * Si una gramática es LL(1), entonces no es ambigua.

41. * Dada una *derivación a izquierdas* $S \overset{*}{\Rightarrow} wA\alpha$ y siendo a el símbolo actual de la cadena de entrada. ¿Qué parte de la forma sentencial $wA\alpha$ puede estar en la pila de un analizador LL(1) y qué condición se debe cumplir para que la siguiente acción sea derivar $A \to \beta$?

42. * Dada la siguiente gramática: a) Construid la tabla de análisis LL(1), y b) Proporcionad la traza LL(1) para la cadena: `x z x`

$$\begin{array}{lcl} \text{A} & \to & \text{B C} \\ \text{B} & \to & \text{w D} \mid \text{E D} \end{array} \qquad \begin{array}{lcl} \text{C} & \to & \text{z D} \\ \text{D} & \to & \text{x D} \mid \epsilon \end{array} \qquad \begin{array}{lcl} \text{E} & \to & \text{y} \mid \epsilon \end{array}$$

43. * Dada la siguiente gramática, obtened una gramática LL(1) equivalente y comprobad que la nueva gramática es LL(1).

$$\text{P} \to \text{P x} \mid \text{y P y} \mid \text{y Q} \qquad \text{Q} \to \text{x} \mid \text{z}$$

44. * Para la gramática del ejercicio 21, comprobad si cumple la condición LL(1); en caso negativo, transformad la gramática y construid su tabla de análisis LL(1) ¿Es una gramática LL(1)? En caso afirmativo, proporcionad la traza de análisis para la cadena: $\{\{i\}, \{i, i\}\}$

45. Ídem para la gramática del ejercicio 37 y la siguiente cadena: *aabbcb*

46. * Ídem para la siguiente gramática y las cadenas: *aab* *d*

$$\begin{array}{lcl} \text{A} & \Rightarrow & \text{B b} \mid \text{C d} \\ \text{C} & \Rightarrow & \text{c C} \mid \epsilon \end{array} \qquad \begin{array}{lcl} \text{B} & \Rightarrow & \text{a B} \mid \epsilon \end{array}$$

47. Ídem para la siguiente gramática y las dos cadenas siguientes: *c* ϵ

$$\begin{array}{lcl} \text{A} & \Rightarrow & \text{B C} \\ \text{C} & \Rightarrow & \text{c C} \mid \epsilon \end{array} \qquad \begin{array}{lcl} \text{B} & \Rightarrow & \text{b c B} \mid \epsilon \end{array}$$

48. Ídem para la siguiente gramática y la cadena: *ca*

$$\begin{array}{lcl} \text{S} & \Rightarrow & \text{B} \\ \text{C} & \Rightarrow & \text{c C} \mid \epsilon \end{array} \qquad \begin{array}{lcl} \text{B} & \Rightarrow & \text{C D} \\ \text{D} & \Rightarrow & \text{a} \end{array}$$

49. Ídem para la siguiente gramática y la cadena: 0110

$$\text{S} \Rightarrow \text{0 S 0} \mid \text{1 S 1} \mid \epsilon$$

50. * Ídem para la siguiente gramática y la cadena: *aabb*

$$\begin{array}{lcl} \text{S} & \Rightarrow & \text{A B} \\ \text{A} & \Rightarrow & \text{a A} \mid \epsilon \end{array} \qquad \begin{array}{lcl} \text{B} & \Rightarrow & \text{b B} \mid \epsilon \end{array}$$

51. Demostrad que las siguientes gramáticas no son LL(1).

$$\begin{array}{lcl} \text{S} & \Rightarrow & \text{A B} \mid \text{C D x} \\ \text{A} & \Rightarrow & \text{x y} \mid \text{m} \\ \text{B} & \Rightarrow & \text{z} \\ \text{C} & \Rightarrow & \text{p C} \mid \epsilon \\ \text{D} & \Rightarrow & \text{q D} \mid \epsilon \end{array} \qquad \begin{array}{lcl} \text{S} & \Rightarrow & \text{A b c} \mid \text{a A c b} \\ \text{A} & \Rightarrow & \text{b} \mid \text{c} \mid \epsilon \end{array}$$

$$\begin{array}{lcl} \text{S} & \Rightarrow & \text{a A a} \mid \epsilon \\ \text{A} & \Rightarrow & \text{a b S} \mid \text{C} \\ \text{C} & \Rightarrow & \text{b C} \mid \epsilon \end{array}$$

52. Demostrad que una gramática tiene una tabla de análisis LL(1), sin entradas múltiples, si y solo si la gramática cumple la condición LL(1).

53. Indicad si las siguientes afirmaciones son ciertas o falsas, justificando brevemente la respuesta:

   a) En una tabla de análisis LL(1) todas las filas (asociadas a los no-terminales) tienen al menos una acción `derivar`.

   b) En una tabla de análisis LL(1) todas las columnas (asociadas a los terminales) tienen al menos una acción `derivar`.

54. Dada la siguiente gramática, eliminad la recursividad a izquierdas. ¿La gramática resultante cumple la condición LL(1)?

$$\text{S} \Rightarrow \text{A b} \mid \text{B c} \qquad \text{A} \Rightarrow \text{A a} \mid \epsilon \qquad \text{B} \Rightarrow \text{B a} \mid \epsilon$$

55. * Considerad la siguiente gramática en la que le faltan dos reglas

$$\begin{array}{lcl} \text{S} & \Rightarrow & \text{a S} \mid \ldots(1) \\ \text{A} & \Rightarrow & \ldots(2) \mid \epsilon \end{array} \qquad \begin{array}{lcl} \text{X} & \Rightarrow & \text{c S} \mid \epsilon \\ \text{Y} & \Rightarrow & \text{d S} \mid \epsilon \end{array} \qquad \text{Z} \Rightarrow \text{e S}$$

Afortunadamente disponemos del conjunto de PRIMEROS y SIGUIENTES:

	PRIMEROS	SIGUIENTES
S	$\{a, b, c, d, e\}$	$\{\$\} \cup$ SIGUIENTES(X) $\cup$ SIGUIENTES(Y) $\cup$ SIGUIENTES(Z)
A	$\{c, d, e, \epsilon\}$	$\{b\}$
X	$\{c, \epsilon\}$	(PRIMEROS(Y) $-\epsilon$) $\cup$ PRIMEROS(Z)
Y	$\{d, \epsilon\}$	PRIMEROS(Z)
Z	$\{e\}$	SIGUIENTES(A)

Reconstruir la gramática completando las reglas que faltan.

56. Comprobad si las siguientes gramáticas son LL(1), en caso contrario tratad de obtener una LL(1) equivalente.

$$\begin{array}{lcl} \text{A} & \Rightarrow & \text{a A F} \mid \text{a B F} \mid \text{a} \\ \text{B} & \Rightarrow & \text{B b} \mid \text{c} \\ \text{F} & \Rightarrow & \text{f} \end{array} \qquad \begin{array}{lcl} \text{S} & \Rightarrow & \text{A a} \mid \text{b} \\ \text{A} & \Rightarrow & \text{S B} \\ \text{B} & \Rightarrow & \text{a b} \end{array} \qquad \begin{array}{lcl} \text{E} & \Rightarrow & \text{E} + \text{E} \\ \text{E} & \Rightarrow & \text{E} * \text{E} \\ \text{E} & \Rightarrow & (\ \text{E}\ ) \mid \text{a} \end{array}$$

57. Demostrad que la transformación que elimina la recursividad a izquierdas produce una gramática equivalente.

$$A \to A\ \alpha \mid \beta \qquad \Rightarrow \qquad A \to \beta\ A' \qquad A' \to \alpha\ A' \mid \epsilon$$

58. Demostrad que la transformación que elimina posible factores comunes produce una gramática equivalente.

$$A \to \alpha\ \beta_1 \mid \alpha\ \beta_2 \mid \gamma \qquad \Rightarrow \qquad A \to \alpha\ A' \mid \gamma \qquad A' \to \beta_1 \mid \beta_2$$

59. * Demostrar que la siguiente gramática es LL(1), a pesar de que es factorizable a izquierdas:

$$\text{S} \Rightarrow \text{A x} \mid \text{A} \qquad \text{A} \Rightarrow \epsilon$$

60. * Dada la siguiente gramática: a) calculad los PRIMEROS y SIGUIENTES de todos los no-terminales; b) construid la tabla de análisis LL(1); y c) ¿es LL(1) y porqué?.

$$\begin{array}{lcl} \text{S} & \Rightarrow & \text{U V W} \\ \text{U} & \Rightarrow & \text{u} \mid \text{W v} \mid \epsilon \end{array} \qquad \begin{array}{lcl} \text{V} & \Rightarrow & \text{w} \mid \text{x U} \mid \epsilon \\ \text{W} & \Rightarrow & \text{y} \mid \text{z} \end{array}$$

3.2.1. Ejercicios resueltos

Ejercicio 35

Lo vamos a demostrar por reducción al absurdo: Sea una gramática $\mathcal{G}$ bien definida y $\exists A \in N : \text{SIGUIENTE}(A) = \emptyset$.

En primer lugar, $A \neq S$ ya que $\text{SIGUIENTE}(S) = \{\$\}$.

Y en segundo lugar, si $\text{SIGUIENTE}(A) = \emptyset$ esto implica que $\nexists(B \to \alpha A \beta) \in \mathcal{P}$ en la que A se parte derecha de una regla. Si esto ocurre, A no sería accesible y por tanto $\mathcal{G}$ no estaría bien definida, lo que contradice la suposición y demuestra el enunciado.

Ejercicio 36

SIGUIENTES(S)={$}; SIGUIENTES(A)={a, b, $}; SIGUIENTES(B)={b, $}; SIGUIENTES(C)={$}; SIGUIENTES(D)={a, b, $};

(PRIMEROS(bbD · SIGUIENTES(A))={b}) ∩ (PRIMEROS(a · SIGUIENTES(A))={a}) = ∅
(PRIMEROS(a · SIGUIENTES(B))={a}) ∩ (SIGUIENTES(B)={b, $}) = ∅
(PRIMEROS(b · SIGUIENTES(C))={b}) ∩ (SIGUIENTES(C)={$}) = ∅
(PRIMEROS(c · SIGUIENTES(D))={c}) ∩ (SIGUIENTES(D)={a, b, $}) = ∅

La gramática es LL(1)

Ejercicio 37

SIGUIENTES(S)={$}; SIGUIENTES(A)={e}; SIGUIENTES(B)={c, $}; SIGUIENTES(C)={$}; SIGUIENTES(D)={c, $}; SIGUIENTES(E)={b, c $};

(PRIMEROS(a · SIGUIENTES(A))={a}) ∩ (SIGUIENTES(A)={e}) = ∅
(PRIMEROS(c · SIGUIENTES(C))={c}) ∩ (SIGUIENTES(C)={$}) = ∅
(PRIMEROS(B · SIGUIENTES(D))={c, $}) ∩ (PRIMEROS(bb · SIGUIENTES(D)={b})) = ∅
(PRIMEROS(Ae · SIGUIENTES(E))={a, e}) ∩ (SIGUIENTES(E)={b, c, $}) = ∅

La gramática es LL(1)

Ejercicio 38

SIGUIENTES(S)={$}; SIGUIENTES(B)=SIGUIENTES(C)=SIGUIENTES(D)={a, c, d, $};

(PRIMEROS(CD · SIGUIENTES(B)) = {a, c, d, $}) ∩
(PRIMEROS(DC · SIGUIENTES(B)) = {a, c, d, $}) ≠ ∅

(PRIMEROS(cC · SIGUIENTES(C))={c}) ∩
(SIGUIENTES(C)={a, c, d, $}) ≠ ∅

(PRIMEROS(Dd · SIGUIENTES(D))={a, c, d, $}) ∩
(PRIMEROS(a · SIGUIENTES(D)={a})) ∩
(SIGUIENTES(D)={a, c, d, $}) ≠ ∅

La gramática no es LL(1)

Ejercicio 40*a*

Supongamos que una gramática LL(1) es recursiva a izquierdas. Una gramática recursiva a izquierdas tiene reglas del tipo:

$$A \to A\alpha_1 | \dots | A\alpha_n | \beta_1 | \dots | \beta_m$$

Por ser LL(1) se debe de verificar la condición:

$$(\bigcap_{i=1}^{n} primeros(A\alpha_i \cdot siguientes(A))) \bigcap (\bigcap_{j=1}^{m} primeros(\beta_j \cdot siguientes(A))) = \emptyset$$

– Para todo $\beta_j \neq \epsilon$ con $j : 1 \ldots m$, se tiene:

$$\begin{aligned} primeros(\beta_j) &\subseteq primeros(A) \subseteq primeros(A\alpha_i \cdot siguientes(A)) \quad \forall i : 1 \ldots n. \\ primeros(\beta_j) &\subseteq primeros(\beta_j \cdot siguientes(A)) \end{aligned}$$

y por tanto incumple la hipótesis de partida.

– Para $\beta_k = \epsilon$, se tiene:

$$siguiente(A) \subseteq primeros(\beta_k \cdot siguientes(A))$$

Como $A \stackrel{*}{\Rightarrow} \epsilon$ entonces $primeros(\alpha_i \cdot siguientes(A)) \subseteq primeros(A\alpha_i \cdot siguientes(A))$, y como los $primeros(\alpha_i \cdot siguientes(A))$ son símbolos siguientes de A, se tiene que:

$$siguiente(A) \subseteq primeros(A\alpha_i \cdot siguientes(A))$$

incumpliendo también la hipótesis de partida.

Ejercicio 40*b*

Dada una gramática LL(1), supongamos que es ambigua. Si es ambigua entonces existe al menos dos secuencias de derivaciones a izquierdas posibles para una una cadena dada:

$$d_1 : S \stackrel{*}{\Rightarrow} x \qquad d_2 : S \stackrel{*}{\Rightarrow} x$$

Si se cumple esto, entonces existe una forma sentencial $x_1 \ldots x_{i-1} A\omega$ (la forma sentencial inicial S, también lo cumple) en la que a partir de ella se diferencian las dos secuencias de derivación a izquierdas d_1 y d_2. Para que se cumpla esto deben existir al menos dos reglas ($A \to \beta_1$ y $A \to \beta_2$) que cumplan

$$\begin{aligned} x_i &\in \text{PRIMEROS}(\beta_1 \cdot \text{SIGUIENTES(A)}) \text{ y} \\ x_i &\in \text{PRIMEROS}(\beta_2 \cdot \text{SIGUIENTES(A)}) \end{aligned}$$

Lo cual contradice que la gramática sea LL(1)

Ejercicio 41 $A\alpha$

$a \in$ PRIMEROS (β SIGUIENTES(A)).

Ejercicio 42

a) SIGUIENTES(A)={$}; SIGUIENTES(B)={z};
SIGUIENTES(C)={$}; SIGUIENTES(D)={$, z};
SIGUIENTES(E)={x, z};

(PRIMEROS(B C · SIGUIENTES(A))={w, x, y, z})
(PRIMEROS(w D · SIGUIENTES(B))={w})
(PRIMEROS(E D · SIGUIENTES(B))={x, y, z})
(PRIMEROS(z D · SIGUIENTES(C))={z})
(PRIMEROS(x D · SIGUIENTES(D))={x})
(PRIMEROS(SIGUIENTES(D))={z, $})
(PRIMEROS(y · SIGUIENTES(E))={y})
(PRIMEROS(SIGUIENTES(E))={x, z})

	w	**x**	**y**	**z**	**$**
A	r1	r1	r1	r1	
B	r2	r3	r3	r3	
C				r4	
D		r5		r6	r6
E		r8	r7	r8	

b)

A $	x z x $	–
B C $	x z z $	1
E D C $	x z x $	1 3
D C $	x z x $	1 3 8
x D C $	x z x $	1 3 8 5
D C $	z x $	1 3 8 5
C $	z x $	1 3 8 5 6
z D $	z x $	1 3 8 5 6 4
D $	x $	1 3 8 5 6 4
x D $	x $	1 3 8 5 6 4 5
D $	$	1 3 8 5 6 4 5
$	$	1 3 8 5 6 4 5 6

Ejercicio 43 La gramática tiene recursividad a izquierdas y problemas de factorización. Podemos proceder de dos maneras equivalentes:

1. Factorizando primero y eliminando después la recursividad a izquierdas.

$$\begin{array}{l} \mathrm{P}\rightarrow \mathrm{P\ x} \\ \mathrm{P}\rightarrow \mathrm{y\ P\ y} \\ \mathrm{P}\rightarrow \mathrm{y\ Q} \\ \mathrm{Q}\rightarrow \mathrm{x} \\ \mathrm{Q}\rightarrow \mathrm{z} \end{array} \Longrightarrow \begin{array}{l} \mathrm{P}\rightarrow \mathrm{P\ x} \\ \mathrm{P}\rightarrow \mathrm{y\ P^1} \\ \mathrm{P^1}\rightarrow \mathrm{P\ y} \\ \mathrm{P^1}\rightarrow \mathrm{Q} \\ \mathrm{Q}\rightarrow \mathrm{x} \\ \mathrm{Q}\rightarrow \mathrm{z} \end{array} \Longrightarrow \begin{array}{l} \mathrm{P}\rightarrow \mathrm{y\ P^1\ P^2} \\ \mathrm{P^2}\rightarrow \mathrm{x\ P^2} \\ \mathrm{P^2}\rightarrow \epsilon \\ \mathrm{P^1}\rightarrow \mathrm{P\ y} \\ \mathrm{P^1}\rightarrow \mathrm{Q} \\ \mathrm{Q}\rightarrow \mathrm{x} \\ \mathrm{Q}\rightarrow \mathrm{z} \end{array}$$

$\textsc{siguientes}(\mathrm{P}) = \textsc{siguientes}(\mathrm{P^2}) = \{\mathrm{y}, \$\};$
$\textsc{siguientes}(\mathrm{P^1}) = \textsc{siguientes}(\mathrm{Q}) = \{\mathrm{x}, \mathrm{y}, \$\};$

$\textsc{primeros}(\mathrm{P\ y} \cdot \textsc{siguientes}(\mathrm{P^1})) = \{\mathrm{y}\} \cap \textsc{primeros}(\mathrm{Q} \cdot \textsc{siguientes}(\mathrm{P^1})) = \{\mathrm{x}, \mathrm{y}\} = \emptyset$
$\textsc{primeros}(\mathrm{x\ P^2} \cdot \textsc{siguientes}(\mathrm{P^2})) = \{\mathrm{x}\} \cap \textsc{primeros}(\textsc{siguientes}(\mathrm{P^2})) = \{\mathrm{y}, \$\} = \emptyset$
$\textsc{primeros}(\mathrm{x} \cdot \textsc{siguientes}(\mathrm{Q})) = \{\mathrm{x}\} \cap \textsc{primeros}(\mathrm{z} \cdot \textsc{siguientes}(\mathrm{Q})) = \{\mathrm{z}\} = \emptyset$

La gramática resultante es LL(1)

2. Eliminando la recursividad primero y factorizando después.

$$\begin{array}{l} \mathrm{P}\rightarrow \mathrm{P\ x} \\ \mathrm{P}\rightarrow \mathrm{y\ P\ y} \\ \mathrm{P}\rightarrow \mathrm{y\ Q} \\ \mathrm{Q}\rightarrow \mathrm{x} \\ \mathrm{Q}\rightarrow \mathrm{z} \end{array} \Longrightarrow \begin{array}{l} \mathrm{P}\rightarrow \mathrm{y\ P\ y\ P^a} \\ \mathrm{P}\rightarrow \mathrm{y\ Q\ P^a} \\ \mathrm{P^a}\rightarrow \mathrm{x\ P^a} \\ \mathrm{P^a}\rightarrow \epsilon \\ \mathrm{Q}\rightarrow \mathrm{x} \\ \mathrm{Q}\rightarrow \mathrm{z} \end{array} \Longrightarrow \begin{array}{l} \mathrm{P}\rightarrow \mathrm{y\ P^b} \\ \mathrm{P^b}\rightarrow \mathrm{P\ y\ P^a} \\ \mathrm{P^b}\rightarrow \mathrm{Q\ P^a} \\ \mathrm{P^a}\rightarrow \mathrm{x\ P^a} \\ \mathrm{P^a}\rightarrow \epsilon \\ \mathrm{Q}\rightarrow \mathrm{x} \\ \mathrm{Q}\rightarrow \mathrm{z} \end{array}$$

$\textsc{siguientes}(\mathrm{P}) = \textsc{siguientes}(\mathrm{P^a}) = \textsc{siguientes}(\mathrm{P^b}) = \{\mathrm{y}, \$\};$
$\textsc{siguientes}(\mathrm{Q}) = \{\mathrm{x}, \mathrm{y}, \$\};$

$\textsc{primeros}(\mathrm{P\ y\ P^a} \cdot \textsc{siguientes}(\mathrm{P^b})) = \{\mathrm{y}\} \cap \textsc{primeros}(\mathrm{Q\ P^a} \cdot \textsc{siguientes}(\mathrm{P^b})) = \{\mathrm{x}, \mathrm{z}\} = \emptyset$
$\textsc{primeros}(\mathrm{x\ P^a} \cdot \textsc{siguientes}(\mathrm{P^a})) = \{\mathrm{x}\} \cap \textsc{primeros}(\textsc{siguientes}(\mathrm{P^a})) = \{\mathrm{y}, \$\} = \emptyset$
$\textsc{primeros}(\mathrm{x} \cdot \textsc{siguientes}(\mathrm{Q})) = \{\mathrm{x}\} \cap \textsc{primeros}(\mathrm{z} \cdot \textsc{siguientes}(\mathrm{Q})) = \{\mathrm{z}\} = \emptyset$

La gramática resultante es LL(1)

Ejercicio 44 La gramática no es LL(1) porque es recursiva a izquierdas. Aplicando la transformación queda:

$B \Rightarrow \{\ \mathrm{I}\ \}\ |\ \mathrm{i} \qquad I \Rightarrow \mathrm{B}\ \mathrm{I'} \qquad I' \Rightarrow ,\ \mathrm{B}\ \mathrm{I'}\ |\ \epsilon$

SIGUIENTES(B)={",", "}", $}; SIGUIENTES(I)=SIGUIENTES(I')={"}"};

	"{"	"}"	i	","	$
B	(1, "{")		(2, i)		
I	(3, BI')		(3, BI')		
I'		(5, ϵ)		(4, ,BI')	

La gramática es LL(1)

S$	{{i},{i,i}}$	–
{I}$	{{i},{i,i}}$	1
BI'}$	{i},{i,i}}$	1 3
{I}I'}$	{i},{i,i}}$	1 3 1
BI'}I'}$	i},{i,i}}$	1 3 1 3
iI'}I'}$	i},{i,i}}$	1 3 1 3 2
}I'}$	},{i,i}}$	1 3 1 3 2 5
,BI'}$	,{i,i}}$	1 3 1 3 2 5 4
{I}I'}$	{i,i}}$	1 3 1 3 2 5 4 1
BI'}I'}$	i,i}}$	1 3 1 3 2 5 4 1 3
iI'}I'}$	i,i}}$	1 3 1 3 2 5 4 1 3 2
,BI'}I'}$	,i}}$	1 3 1 3 2 5 4 1 3 2 4
iI'}I'}$	i}}$	1 3 1 3 2 5 4 1 3 2 4 2
}I'}$	}}$	1 3 1 3 2 5 4 1 3 2 4 2 5
}$	}}$	1 3 1 3 2 5 4 1 3 2 4 2 5 5
$	$	1 3 1 3 2 5 4 1 3 2 4 2 5 5

Ejercicio 46

PRIMEROS(A) $= \{a, b\} \cup \{c, d\}$ SIGUIENTES(A) $= \{\$\}$
PRIMEROS(B) $= \{a\} \cup \{\epsilon\}$ SIGUIENTES(A) $= \{b\}$
PRIMEROS(C) $= \{c\} \cup \{\epsilon\}$ SIGUIENTES(A) $= \{d\}$

(PRIMEROS(Bb · SIGUIENTES(A)) $= \{a, b\}) \cap$
(PRIMEROS(Cd · SIGUIENTES(A)) $= \{c, d\}) = \emptyset$

(PRIMEROS(aB · SIGUIENTES(B))$= \{a\}) \cap$
(SIGUIENTES(B)$= \{b\}) = \emptyset$

(PRIMEROS(cC · SIGUIENTES(C))$= \{c\}) \cap$
(SIGUIENTES(C)$= \{d\}) = \emptyset$

	a	b	c	d	$
A	(1, Bb)	(1, Bb)	(2, Cd)	(2, Cd)	
B	(3, aB)	(4, ϵ)			
C			(5, cC)	(6, ϵ)	

A$	aab$	–
Bb$	aab$	1
aBb$	aab$	1 3
Bb$	ab$	1 3
aBb$	ab$	1 3 3
Bb$	b$	1 3 3
b$	b$	1 3 3 4
$	$	1 3 3 4

Ejercicio 50 La TA LL(1) será:

	a	b	$
S	(1, AB)	(1, AB)	(1, AB)
A	(2, aA)	(3, ϵ)	(3, ϵ)
B		(4, bB)	(5, ϵ)

La gramática es LL(1)

S$	aabb$	–
AB$	aabb$	1
aAB$	aabb$	1 2
aAB$	abb$	1 2 2
B$	bb$	1 2 2 3
bB$	bb$	1 2 2 3 4
bB$	b$	1 2 2 3 4 4
$	$	1 2 2 3 4 4 5

Ejercicio 55 La gramática construida puede ser:

S ⇒ a S | A b X ⇒ c S | ϵ Z ⇒ e S
A ⇒ X Y Z | ϵ Y ⇒ d S | ϵ

Ejercicio 59 La gramática es LL(1) ya que cumple la condición LL(1):

(PRIMEROS(A x · SIGUIENTES(S)) = { x }) ∩ (PRIMEROS(A · SIGUIENTES(S)) = { $ }) = ∅
SIGUIENTES(A) = { x, $ })

Ejercicio 60

a)

	PRIMEROS	SIGUIENTES
S	$\{u, y, z, w, x\}$	$\{\$\}$
U	$\{u, y, z, \epsilon\}$	$\{w, x, y, z\}$
V	$\{w, x, \epsilon\}$	$\{y, z\}$
W	$\{y, z\}$	$\{v, \$\}$

b) La TA-LL(1) (solo para los no-terminales) correspondiente es:

	x	**y**	**z**	**u**	**v**	**w**	**$**
S	(UVW,1)	(UVW,1)	(UVW,1)	(UVW,1)		(UVW,1)	
U	(ϵ,4)	(ϵ,4)	(ϵ,4)	(u,2)		(ϵ,4)	
		(Wv,3)	(Wv,3)				
V	(xU,6)	(ϵ,7)	(ϵ,7)			(w,5)	
W		(y,8)	(z,9)				

c) NO, ya que la TA-LL(1) tiene entradas múltiples.

3.3. Análisis Sintáctico Ascendente

61. * Dada la gramática $\{S \rightarrow a\ S\ b \mid a\ b\}$, indicad cual será el *pivote* de las siguientes formas sentenciales: $aaabbb$ y $aaSbb$

62. * Para la misma gramática del ejercicio 61 describid, de una forma compacta, todos sus prefijos viables. Para el prefijo viable `a`, obtened todos sus ítems válidos LR(0).

63. Dado una gramática LR(0) con un ítem [$A \rightarrow \alpha\ .$], con $\alpha \neq \epsilon$ válido para algún prefijo viable γ, demostrad que ningún otro ítem puede ser válido para γ

64. Dada la siguiente gramática, calculad el conjunto de ítems LR(0) válidos para el prefijo viable c.

$$\text{A} \Rightarrow \text{a B} \mid \text{c C} \qquad \text{B} \Rightarrow \text{b B} \mid \text{f} \qquad \text{C} \Rightarrow \text{c C} \mid \text{f}$$

65. * Dada una gramática LR(0) con un ítem válido, [A $\rightarrow \alpha\ .$] con $\alpha \neq \epsilon$, para algún prefijo viable γ, demostrad que ningún otro ítem puede ser válido para γ.

66. * Demostrad si la siguiente afirmación es cierta o falsa:

    `Una gramática no es SLR(1) si existe un prefijo viable (para una forma sentencial a derechas) que tenga más de un ítem LR(0) válido`

67. Demostrad que no pueden existir conflictos *desplazamiento/desplazamiento* en la construcción de una tabla de análisis SLR(1).

68. * Dada una *derivación a derechas* $S' \overset{*}{\Rightarrow} \alpha A w \Rightarrow \alpha\beta w$ y siendo a el símbolo actual de la cadena de entrada. ¿Qué parte de la forma sentencial $\alpha\beta w$ puede estar en la pila de un analizador SLR(1) y que condición se debe cumplir para que la siguiente reducción a realizar es la que aparece explícitamente en la derivación?

69. * Justificar brevemente si las siguientes afirmaciones son ciertas o falsas:
    - `a)` El autómata de la Colección Canónica de Conjuntos e Ítems LR(0) puede ser indeterminista.
    - `b)` El estado inicial del autómata de la Colección Canónica de Conjuntos e Ítems LR(0) puede aparecer en alguna otra entrada (como estado siguiente) de la tabla de análisis SLR(1).
    - `c)` La tabla de análisis (acción) SLR(1) no podrá contener ninguna columna vacía (solo acciones error).
    - `d)` La tabla de análisis (acción) SLR(1) no podrá contener ninguna fila vacía (solo acciones error).
    - `e)` Si una gramática es recursiva a izquierdas entonces no es SRL(1).

70. Dada la siguiente gramática: a) Obtened las tablas de análisis LR(0) y SLR(1); b) Proporcionad la traza del análisis LR(0) y SLR(0) para la cadena $wyyz$

$$\text{E} \Rightarrow \text{w X} \mid \text{x Y} \qquad \text{X} \Rightarrow \text{y X} \mid \text{z} \qquad \text{Y} \Rightarrow \text{y Y} \mid \text{z}$$

71. Dada la gramática que se adjunta, construid la Colección Canónica de Conjuntos de ítems LR(0). ¿Es una gramática LR(0)? ¿Es SLR(1)?

$$\text{S} \Rightarrow \text{(A)} \qquad \text{A} \Rightarrow \text{A , D} \mid \text{D} \qquad \text{D} \Rightarrow \text{a} \mid \text{b} \mid \text{(A)}$$

72. Razonad por qué la sigiente gramática es LL(1) pero no SLR(1):

$$S \Rightarrow A\ a\ A\ b \mid B\ b\ B\ a \qquad A \Rightarrow \epsilon \qquad B \Rightarrow \epsilon$$

Avertid que no basta con demostrar que es LL(1) y no es SLR(1), es necesario justificar por qué es así.

73. * Dada la gramática que se adjunta: a) Construid la Tabla de Análisis SLR(1) a partir de la Colección Canónica de Conjuntos de ítems LR(0); b) Obtened la traza de analisis LALR(1) para la cadena *ab*.

$$S \Rightarrow A\ B \qquad A \Rightarrow a\ A \mid \epsilon \qquad B \Rightarrow b\ A \mid \epsilon$$

74. * Ídem para la siguiente gramática y la cadena *aaba*.

$$S \Rightarrow a\ S\ A \mid \epsilon \qquad A \Rightarrow B\ b \qquad B \Rightarrow A\ c \mid \epsilon$$

75. * Dada la gramática que se adjunta: a) Obtened Colección Canónica de Conjuntos de ítems LR(0); b) Construid la Tabla de Análisis LR(0) y SLR(1); c) ¿Es una gramática LR(0), y SLR(0)?;

$$S \Rightarrow A\ b \mid B\ c \qquad A \Rightarrow A\ a \mid \epsilon \qquad B \Rightarrow B\ a \mid \epsilon$$

76. Dada la siguiente gramática: a) Obtened la tabla de análisis SLR(1); b) Proponed una estrategia frente a los posibles conflictos, determinando la acción a seguir en cada caso, de forma que se mantenga la precedencia y la asociatividad usual entre los operadores (de mayor a menor precedencia: **no**, **y**, **o**; los operadores binarios son asociativos por la izquierda y el unario por la derecha).

$$S \Rightarrow S\ \mathbf{o}\ S \mid S\ \mathbf{y}\ S \mid \mathbf{no}\ S \mid \mathbf{cierto} \mid S\ \mathbf{falso} \mid \mathbf{id}$$

77. * `[J.Beaty, 1979]` Demostrad que la siguiente gramática es LL(1) pero no SLR(1):

$$\begin{array}{lll} S \Rightarrow a\ F \mid b\ G & F \Rightarrow X\ c \mid Y\ d & G \Rightarrow X\ d \mid Y\ c \\ X \Rightarrow I\ A & Y \Rightarrow I\ B & I \Rightarrow \epsilon \\ A \Rightarrow \epsilon & B \Rightarrow \epsilon & \end{array}$$

78. La clase de gramáticas LL(1) y SLR(1) son incomparables:

a) `[Aho et al., 1990)]` Demostrad que la siguiente gramática es LL(1) pero no SLR(1):

$$S \Rightarrow A\ a\ A\ b \mid B\ b\ A\ a \qquad A \Rightarrow \epsilon \qquad B \Rightarrow \epsilon$$

b) Demostrad que la siguiente gramática es SLR(1) pero no LL(1):

$$S \Rightarrow S\ a \mid a$$

3.3.1. Ejercicios resueltos

Ejercicio 61 `a a a b b b` `a a S b b`

Ejercicio 62 $S \mid a^{+}(b \mid S\ b?)$

$\{[S \to a\ .\ S\ b], [S \to a\ .\ b], [S \to .\ a\ S\ b], [S \to .\ a\ b],\}$

Ejercicio 65 Dado una gramática LR(0) con [A $\to \alpha$.] en un cierto estado, Si existiera otro ítem en ese estado, solo pueden pasar dos cosas:

- que sea un ítem [B $\to \beta$.], con lo cual se producirá un conflicto redución-reducción y no sería LR(0), o
- que sea un ítem [B $\to \beta$. ω], con lo cual se producirá un conflicto desplazamiento-reducción y no sería LR(0).

Ejercicio 66 Si se cumple la afirmación entonces la gramática no es LR(1) (de hecho es la definición de gramática LR(1)). Dado que las gramáticas SLR(1) son un subconjunto propio de las LR(1) entonces la gramática tampoco será SLR(1).

Ejercicio 68 $\alpha\ \beta$ deben estar en la pila y la condición para la reducción es que se cumpla que: a $\in$ `siguiente`(A).

Ejercicio 69

a) `FALSO`. Por construcción la operación `sucesor` solo permite una transición por símbolo.
b) `FALSO`. El estado inicial del autómata de la Colección Canónica de Conjuntos e Ítems LR(0) incorpora [S' $\to$. S] y dado que la gramática debe ser reducida (S' no puede aparecer en niguna parte derecha de ninguna regla), implica que [S' $\to$. S] no puede ser un ítem válido para ningún otro prefijo viable.
d) `FALSO`. Supondría que cuando ese símbolo terminal (válido) de la gramática aparezca como símbolo actual de la cadena de entrada, la gramática solo produce error.
d) `FALSO`. De ser cierta equivaldría a tener un estado sin ningún ítem válido.
e) `FALSO`. Contaejemplo, la gramática (A $\to$ A a | b) es recursiva a izquierdas y es SLR(1).

Ejercicio 73

I_0 | S' $\to$ · S
S $\to$ · A B
A $\to$ · a A
A $\to$ · |
I_1 | S' $\to$ S · |
I_2 | S $\to$ A · B
B $\to$ · b A
B $\to$ · |
I_3 | A $\to$ a · A
A $\to$ · a A
A $\to$ · |
I_4 | S $\to$ A B · |
I_5 | B $\to$ b · A
A $\to$ · a A
A $\to$ · |
I_6 | A $\to$ a A · |
I_7 | B $\to$ b A · |

$I_0 \xrightarrow{S} I_1$; $I_0 \xrightarrow{A} I_2$; $I_0 \xrightarrow{a} I_3$; $I_2 \xrightarrow{B} I_4$; $I_2 \xrightarrow{b} I_5$; $I_3 \xrightarrow{A} I_6$; $I_3 \xrightarrow{a} I_3$; $I_5 \xrightarrow{A} I_7$; $I_5 \xrightarrow{a} I_3$;

	a	b	$	S	A	B
0	D-3	R-3	R-3	1	2	
1			ACC			
2		D-5	R-5			4
3	D-3	R-3	R-3		6	
4			R-1			
5	D-3	R-3	R-3		7	
6		R-2	R-2			4
7			R-4			

0	ab$	–
0a3	b$	3
0a3A6	b$	3 2
0A2	b$	3 2
0A2b5	$	3 2 3
0A2b5A7	$	3 2 3 4
0A2B4	$	3 2 3 4 1
0S1	$	3 2 3 4 1

Ejercicio 74

I_0 | S' → · S
S → · a S A
S → · | I_1 | S' → S · | I_2 | S → a · S A
S → · a S A
S → · | I_3 | S → a S · A
A → · B b
B → · A c
B → ·

I_4 | S → a S A ·
B → A · c | I_5 | A → B · b | I_6 | B → A c · | I_7 | A → B b ·

$I_0 \xrightarrow{S} I_1$; $I_0 \xrightarrow{a} I_2$; $I_2 \xrightarrow{S} I_3$; $I_2 \xrightarrow{a} I_2$; $I_3 \xrightarrow{B} I_5$; $I_3 \xrightarrow{A} I_4$; $I_4 \xrightarrow{c} I_6$; $I_5 \xrightarrow{b} I_7$;

	a	b	c	$	S	A	B
0	D-2	R-2		R-2	1		
1				ACC			
2	D-2	R-2		R-2	3		
3		R-5				4	5
4		R-1	D-6	R-1			
5		D-7					
6		R-4					
7		R-3	R-3	R-3			

0	aabb$	–
0a2	abb$	–
0a2a2	bb$	2
0a2a2S3	bb$	2 5
0a2a2S3B5	bb$	2 5
0a2a2S3B5b7	b$	2 5 3
0a2a2S3A4	b$	2 5 3 1
0a2S3	b$	2 5 3 1 5
0a2S3B5	b$	2 5 3 1 5
0a2S3B5b7	$	2 5 3 1 5 3
0a2S3A4	$	2 5 3 1 5 3 1
0S1	$	2 5 3 1 5 3 1

Ejercicio 75

I_0 | S' → · S
S → · A b
S → · B c
A → · A a
A → ·
S → · B a
S → · | I_1 | S' → S · | I_2 | S → A · b
A → A · a | I_3 | S → B · c
B → B · a

I_4 | S → A b · | I_5 | A → A a · | I_6 | S → B c · | I_7 | B → B a ·

$I_0 \xrightarrow{S} I_1$; $I_0 \xrightarrow{A} I_2$; $I_0 \xrightarrow{B} I_3$; $I_2 \xrightarrow{b} I_4$; $I_2 \xrightarrow{a} I_5$; $I_3 \xrightarrow{c} I_6$; $I_3 \xrightarrow{a} I_7$;

	a	b	c	$	S	A	B
0	$\frac{R-4}{R-6}$	R-4	R-6		1	2	3
1				ACC			
2	D-5	D-4					
3	D-7		D-6				
4				R-1			
5	R-3	R-3					
6				R-2			
7	R-5		R-5				

- No es LR(0), conflicto desplazamiento/reducción en el estado I_0
- No es SLR(1), conflicto reducción/reducción en la TA SLR(1) para el par (0, a)

Ejercicio 77

I_0 | S' → · S
S → · a F
S → · b G | I_2 | S → a · F
F → · X c
F → · Y d
X → · I A
Y → · I B
I → · | I_7 | X → I · A
Y → I · B
A → ·
B → · | ...

$$I_0 \xrightarrow{a} I_2;\ I_2 \xrightarrow{I} I_7;\ \ldots$$

Dado que:

SIGUIENTES(A) = SIGUIENTES(X) = {c, d}; y SIGUIENTES(B) = SIGUIENTES(Y) = {d,c};

en el estado I_7 habrá dos conflictos reducción/reducción, por tanto NO es SLR(1).

Por otro lado, como se cumple:

(PRIMEROS(aF · SIGUIENTES(S))={a}) ∩ (PRIMEROS(bG · SIGUIENTES(S))={b}) = ∅
(PRIMEROS(Xc · SIGUIENTES(F))={c}) ∩ (PRIMEROS(Yd · SIGUIENTES(F))={d}) = ∅
(PRIMEROS(Xd · SIGUIENTES(G))={d}) ∩ (PRIMEROS(Yc · SIGUIENTES(G))={c}) = ∅

entonces SI que es LL(1).

4. Análisis Semántico

4.1. Gramáticas de Atributos

79. Dada la siguiente gramática: a) Construid una gramática S-atribuida que calcule el número de `a` y de `b` que tiene una frase cualquiera del lenguaje generado por dicha gramática; y b) obtened obtener la traza de análisis ascendente y evaluación de los atributos para la frase: (*a* , (*b*))

$$\text{S} \Rightarrow (\ \text{A}\ ) \qquad \text{A} \Rightarrow \text{A}\ ,\ \text{D} \mid \text{D} \qquad \text{D} \Rightarrow (\ \text{A}\ ) \mid \text{a} \mid \text{b}$$

80. * Explicad las restricciones que ha de cumplir una gramática de atributos para que sea L-Atribuida, y la relación de éstas con las gramáticas S-Atribuidas.

81. * Construid un Esquema de Traducción Dirigida por la Sintaxis que, para la siguiente gramática, devuelva la cantidad de `a`, `b` y `c` que se produce en el análisis de una cadena.

$$\text{S} \Rightarrow \text{a S A} \mid \epsilon \qquad \text{A} \Rightarrow \text{B b} \qquad \text{B} \Rightarrow \text{A c} \mid \epsilon$$

82. * La siguiente gramática corresponde a la especificación sintáctica de listas válidas de PROLOG:

$$\text{Lista} \Rightarrow [\ ] \mid [\ \text{T}\ ] \qquad \text{T} \Rightarrow \text{T}\ ,\ \text{T} \mid \text{Lista} \mid \mathit{id}$$

Diseñad una gramática atribuida capaz de procesar una lista y averiguar su longitud y extensión. Se define longitud de una lista o término, como el número de elementos que la componen. Análogamente, se define extensión de una lista o término, como la longitud máxima de cualquiera de sus elementos. Por ejemplo, la lista
`[ [ a1 , a2 ] , a3, [ a4 , [ a5 , a6 ] , [ a7 ] , a8 ] ]` tiene longitud 3 y extensión 4.

83. Construid una gramática S-atribuida que obtenga el número binario correspondiente al complemento a dos de otro número, basándose en la siguiente gramática. Las reglas semánticas deben definirse únicamente en función de los operadores aritméticos decimales básicos y de la operación de concatenación "."

$$\text{S} \Rightarrow \text{C} \qquad \text{C} \Rightarrow 0\ \text{C} \mid 1\ \text{C} \mid 0 \mid 1$$

84. Dada la siguiente gramática que genera árboles binarios de números enteros (en forma linealizada):

$$\begin{aligned} \text{S} &\rightarrow \text{A} \\ \text{A} &\rightarrow (\ \mathit{num}\ \text{A}\ \text{A}\ ) \mid (\ \mathit{num}\ ) \end{aligned}$$

Definamos el nivel de profundidad de un nodo en un árbol binario como el número mínimo de ramas que hay que recorrer para ir del nodo raíz hasta este nodo. Diseñad un ETDS para esta gramática que obtenga el nivel de profundidad que ocupa el número entero de mayor valor; en caso de que el valor máximo se repita, se debe indicar la profundidad mínima. Ejemplo, para la cadena `(2 (1 (0) (1)) (3 (2) (3)))` la contestación sería: valor máximo 3 y profundidad 1.

85. * Diseñad un ETDS para traducir declaraciones de objetos de un lenguaje (parecido al) C a un lenguaje (parecido al) PASCAL, mediante la siguiente gramática:

$$S \rightarrow T\ V \qquad V \rightarrow id\ D \qquad D \rightarrow [\ cte\ ]\ D \mid \epsilon \qquad T \rightarrow int \mid float \mid char$$

Ejemplos:

```
int A [7];          ⇒   var A: array[0..6] of integer;
float B [6] [5];    ⇒   var B: array[0..5,0..4] of real;
char C;             ⇒   var A: char;
```

86. * Dada la siguiente gramática, diseñad un ETDS que permita calcular el número de pares de paréntesis apertura-cierre y el número de *id* de cualquier cadena de entrada.

$$L \rightarrow (\ L\ ) \mid L\ B \mid \epsilon \qquad B \rightarrow id$$

87. * Diseñad un ETDS para traducir declaraciones de objetos de un lenguaje (parecido al) C a un lenguaje (parecido al) PASCAL, mediante la siguiente gramática:

$$D \rightarrow T\ id\ ; \qquad T \rightarrow int \mid float \mid char \mid struct\ \{\ C\ \} \qquad C \rightarrow C\ D \mid D$$

El resultado de la traducción puede acumularse en un atributo `trad` de tipo cadena. Por ejemplo:

```
int a;                              ⇒   a : integer;
struct { float b; char c; } r;      ⇒   r: record b: real; c: char; end;
```

88. * Para la siguiente gramática, construid un ETDS que calule el valor numérico (y lo asocie a un atributo del no-terminal E) de una expresión matemática definida por la gramática.

$$E \rightarrow T\ R \qquad T \rightarrow cte \mid (\ E\ ) \qquad R \rightarrow +\ T\ R \mid -\ T\ R \mid \epsilon$$

89. * Para la misma gramática del ejercicio 90, construid un ETDS que transforme una expresión de notación infija a postfija. Por ejemplo: `2 + 3 - 4` ⇒ `2 3 + 4 -` o este otro `2 + ( 3 - 4 )` ⇒ `2 3 4 - +`

90. * Para la siguiente gramática, construid un ETDS que calule el valor numérico decimal de un número binario. Por ejemplo, el valor decimal de 110 es 6.

$$S \rightarrow 0\ S \mid 1\ S \mid 0 \mid 1$$

91. * Dada la siguiente gramática, que genera cadenas que representan expresiones matemáticas, construid un ETDS que permita obtener como resultado una cadena que sea la derivada de la cadena de entrada. Por ejemplo: $x*x+\text{sen}(x)$ ⇒ $2*x*d_x+\cos(x)*d_x$

$$E \rightarrow E + T \mid T \qquad T \rightarrow T * F \mid F \qquad F \rightarrow \text{sen}\ (\ E\ ) \mid \cos\ (\ E\ ) \mid (\ E\ ) \mid x \mid cte$$

92. * Dada la siguiente gramática, que permite representar árboles binarios, se pide: a) construid un ETDS que obtenga el máximo y el mínimo valor numérico que aparezcan en el árbol; y b) sobre el ETDS del apartado anterior, añadid nuevas acciones semánticas para comprobar si el árbol está ordenado. Un árbol está ordenado si el valor numérico de cada nodo es mayor o igual que todos los valores numéricos del subárbol izquierdo y menor o igual que todos los del subárbol derecho. Por ejemplo, este el árbol está ordenado `(5 (2 nil nil) (7 nil nil))`; sin embargo, este otro no lo está `(4 (2 (1 nil nil) (7 nil nil)) (10 nil nil))`.

A→ (cte A A) | nil

4.1.1. Ejercicios resueltos

Ejercicio 80

Dado una gramática de atributos $GA = (G, A, R)$, donde G es una gramática incontextual, $A = \cup_{x \in (N \cup T)} A(x)$ el conjunto de atributos y $R = \cup_{k \in P} R(k)$ el conjunto de acciones semánticas, GA será **L-atribuida**, si $\forall (k : X_0 \to X_1 X_2 \dots X_n) \in P$, los atributos heredados $X_i.a$ $(1 \leq i \leq n \wedge a \in A(X_i))$ con $(X_i.a := f(X_r.b, \dots, X_s.c)) \in R(k)$ son evaluados en términos de:

- atributos de $X_1, X_2, \dots, X_{i-1}$ ó
- atributos heredados de X_0.

Mientras que, GA será **S-atribuida**, si todos los atributos empleados en todas las acciones semánticas son sintetizados. Atendiendo a la definición anterior, toda gramática S-atribuida es L-atribuida.

Ejercicio 81

S	⇒	a S A	S.a = S'.a + 1; S.b = S'.b + A.b; S.c = S'.c + A.c;
	⇒	ϵ	S.a = S.b =S.c = 0;
A	⇒	B b	A.b = B.b + 1; A.c = B.c
B	⇒	A c	B.b = A.b; B.c = A.c + 1
	⇒	ϵ	B.b = B.c = 0;

Ejercicio 82

L	⇒	[ ]	L.l = L.e = 0;
	⇒	[T]	L.l = T.l; L.e = T.e;
T	⇒	T , T	$T.l = T^1.l + T^2.l$; $T.e = \max\{T^1.e, T^2.e\}$;
	⇒	L	T.l = 1; T.e = L.l
	⇒	id	T.l = T.e = 1;

Ejercicio 85

S	$\Rightarrow$	T V	S.s := V.s $\odot$ T.s
V	$\Rightarrow$	*id* D	si D.s = *nil* entonces V.s := "var" $\odot$ id.nom $\odot$ ":" sino V.s := "var" $\odot$ id.nom $\odot$ " array[" $\odot$ D.s
D	$\Rightarrow$	[*cte*] D	si $D_1.s$ = *nil* entonces D.s := "0.." $\odot$ *cte.num* − 1 $\odot$ "] of" sino D.s := "0.." $\odot$ *cte.num* − 1 $\odot$ " , " $\odot$ $D_1.s$
	$\Rightarrow$	ϵ	D.s := *nil*;
T	$\Rightarrow$	int	T.s := "integer";
	$\Rightarrow$	float	T.s := "real";
	$\Rightarrow$	char	T.s := "char";

Ejercicio 86

L	$\rightarrow$	(L)	L.n $\leftarrow$ L_1.n; L.p $\leftarrow$ L_1.p + 1;
L	$\rightarrow$	LB	L.n $\leftarrow$ L_1.n + B.n; L.p $\leftarrow$ L_1.p;
L	$\rightarrow$	ϵ	L.n $\leftarrow$ 0; L.p $\leftarrow$ 0;
B	$\rightarrow$	*id*	B.n $\leftarrow$ 1;

Ejercicio 87

D	$\rightarrow$	T *id* ;	D.trad = id.nom $\oplus$ " : " $\oplus$ T.trad $\oplus$ ";"
T	$\rightarrow$	*int*	T.trad = "integer"
	$\rightarrow$	*float*	T.trad = "real"
	$\rightarrow$	*char*	T.trad = "char"
	$\rightarrow$	*struct* { C }	T.trad = "record" $\oplus$ C.trad $\oplus$ "end"
C	$\rightarrow$	C D	C.trad = C'.trad $\oplus$ D.trad
	$\rightarrow$	D	C.trad = D.trad

Ejercicio 88

E	$\rightarrow$	T R	R.h = T.s E.s = R.s
R	$\rightarrow$	+ T R	R^1.h = R.h + T.s R.s = R^1.s
R	$\rightarrow$	− T R	R^1.h = R.h − T.s R.s = R^1.s
R	$\rightarrow$	ϵ	R.s = R.h
T	$\rightarrow$	(E)	T.s = E.s
T	$\rightarrow$	*cte*	T.s = LEXVAL(*cte*)

Ejercicio 89

E	$\rightarrow$	T	R.h = T.s
		R	E.s = R.s
R	$\rightarrow$	+ T	R^1.h = R.h $\odot$ T.s $\odot$ +
		R	R.s = R^1.s
R	$\rightarrow$	− T	R^1.h = R.h $\odot$ T.s $\odot$ −
		R	R.s = R^1.s
R	$\rightarrow$	ϵ	R.s = R.h
T	$\rightarrow$	(E)	T.s = E.s
T	$\rightarrow$	*cte*	T.s = *cte*

Ejercicio 90

S	$\rightarrow$	0 S	S.n = S^1.n + 1; S.r = 0 + S^1.r;
S	$\rightarrow$	0 S	S.n = S^1.n + 1; S.r = $2^{S.n}$ + S^1.r;
S	$\rightarrow$	0	S.n = 0; S.r = 0;
S	$\rightarrow$	1	S.n = 0; S.r = 1;

Ejercicio 91

E	$\rightarrow$	E + T	E.f = E^1.f $\oplus$ T.f; E.d = E^1.d $\oplus$ T.d;
E	$\rightarrow$	T	E.f = T.f; E.d = T.d;
T	$\rightarrow$	T * F	T.f = T^1.f $\otimes$ F.f; T.d = T^1.d $\otimes$ F.f $\oplus$ T^1.f $\otimes$ F.d;
T	$\rightarrow$	F	T.f = F.f; T.d = F.d;
F	$\rightarrow$	sen(E)	F.f = sen(E.f); F.d = cos(E.f) $\otimes$ E.d;
F	$\rightarrow$	cos(E)	F.f = cos(E.f); F.d = − sen(E.f) $\otimes$ E.d;
F	$\rightarrow$	(E)	F.f = (E.f); F.d = (E.d);
F	$\rightarrow$	x	F.f = x; F.d = d_x;
F	$\rightarrow$	*cte*	F.f = *cte*; F.d = 0;

Ejercicio 92

A $\rightarrow$ (cte A A)	A.max = max { cte.val, A^1.max, A^2.max }
	A.min = min { cte.val, A^1.min, A^2.min }
	A.ord = (A^1.ord && A^2.ord && (A^1.max $\leq$ cte.val $\leq$ A^1.min))
A $\rightarrow$ nil	A.max = $-\infty$; A.min = ∞; A.ord = true;

4.2. Comprobación de Tipos

93. Explicad que se entiende por conversión de tipos explícita e implícita, quien la realiza y poned un ejemplo de cada una de ellas.

94. Explicad la diferencia entre la equivalencia de tipos por nombre y estructural.

95. Explicad las diferencias entre el análisis de tipos estático y el dinámico. Poned un ejemplo de error detectado por cada uno de ellos.

96. * Considerando que se ha completado la fase de declaración de los objetos, diseñad un ETDS para la comprobación semántica de tipos en el par de reglas:

$$\texttt{E} \rightarrow \texttt{* id} \qquad \text{y} \qquad \texttt{E} \rightarrow \texttt{id}$$

97. * Construid un ETDS para la siguiente gramática: **a)** Desarrollad la declaración de tipos, introduciendo en la tabla de símbolos cada identificador que aparezca en $L_{nombres}$ tantas veces como tipos distintos tenga asociados en L_{tipos}. El significado de la regla "D $\rightarrow$ L_{tipos} (D)" es que a todos los nombres mencionados en la declaración D, dentro del paréntesis, se les dan los tipos que aparecen en L_{tipos}, independientemente de cuantos niveles de anidamiento existan. **b)** Realizad la comprobación de tipos para las instrucciones y expresiones.

$$\begin{array}{lcl}
\text{S} & \rightarrow & \text{D ; I} \\
\text{D} & \rightarrow & L_{tipos}\ L_{nombres} \mid L_{tipos}\ (\ \text{D}\ ) \mid \text{D D} \\
L_{tipos} & \rightarrow & \text{T}\ L_{tipos} \mid \text{T} \\
\text{T} & \rightarrow & entero \mid real \mid complejo \\
L_{nombres} & \rightarrow & id \mid id\ L_{nombres} \\
\text{I} & \rightarrow & id := \text{E} \\
\text{E} & \rightarrow & id \mid \text{E} + \text{E} \mid (\ \text{E}\ ,\ \text{E}\ ) \mid cte \mid ctr
\end{array}$$

98. * Dada la siguiente gramática que genera declaraciones para un único identificador, obtened un ETDS que asigne a cada identificador su lista de opciones y que no permita opciones repetidas.

$$\begin{array}{lcl}
\text{S} & \rightarrow & id\ \text{Lopciones} \\
\text{Lopciones} & \rightarrow & \text{Lopciones Opciones} \mid \epsilon \\
\text{Opciones} & \rightarrow & \text{Modo} \mid \text{Escala} \mid \text{Precision} \mid \text{Base} \\
\text{Modo} & \rightarrow & real \mid complejo \\
\text{Escala} & \rightarrow & fija \mid flotante \\
\text{Precision} & \rightarrow & simple \mid doble \\
\text{Base} & \rightarrow & binaria \mid decimal
\end{array}$$

99. Dada la gramática, diseñad un ETDS que permita realizar la declaración de los objetos y las comprobaciones semáticas correspondientes, teniendo en cuenta que: **a)** el operador + está definido para *cadenas* (concatenación) y para *enteros* (suma); **b)** $a(p, f)$ es una subcadena de la cadena a determinada por los índices p y f; y **c)** $|a|$ es la longitud (entera) actual de la cadena a.

$$
\begin{array}{lcl}
\text{P} & \rightarrow & \text{D ; S} \\
\text{D} & \rightarrow & \text{D ; D} \mid id : \text{T} \\
\text{T} & \rightarrow & entero \mid cadena\ (\ num\ ) \\
\text{S} & \rightarrow & id := \text{E} \\
\text{E} & \rightarrow & \text{E} + \text{E} \mid id\ (\ \text{E}\ ,\ \text{E}\ ) \mid id
\end{array}
$$

100. * La siguiente gramática permite declarar identificadores de tipo *lista* y obtener expresiones de pertenencia de un elemento a una lista. Desarrollad un ETDS que permita hacer una comprobación de tipos de forma que: **a)** los elementos de una lista siempre sean del mismo tipo; y **b)** la relación de pertenencia sea compatible con la condición (a)

$$
\begin{array}{lcl}
\text{P} & \rightarrow & \text{D ; S} \\
\text{P} & \rightarrow & \text{D ; D} \mid id : \text{T} \\
\text{T} & \rightarrow & char \mid list\ of\ \text{T} \\
\text{S} & \rightarrow & \text{E}\ en\ \text{E} \\
\text{E} & \rightarrow & literal \mid id \mid (\ \text{L}\ ) \\
\text{L} & \rightarrow & \text{E}\ ,\ \text{L} \mid \text{E}
\end{array}
$$

4.2.1. Ejercicios resueltos

Ejercicio 96

E ⇒ id	Si ¬[ObtenerTDS(id.nom,E.t)] { E.t=terror; MenError(.); }
E ⇒ * id	Si ¬[ObtenerTDS(id.nom,id.t) ∧ id.t=tpuntero(E.t)] { E.t=terror; MenError(.); }

Ejercicio 97

S → D ; I	
D → L N	D.n = N.n; ∀ x ∈ N.n: `insertarConjTipos`(x, L.t);
→ L (D)	D.n = D'.n; ∀ x ∈ D'.n: `insertarConjTipos`(x, L.t);
→ D D	D.n = $D^1 \cup D^2$;
L → T L	L.t = L'.t ∪ T.t;
→ T	L.t = T.t;
T → entero	T.t = {tentero};
→ real	T.t = {treal};
→ complejo	T.t = {tcomplejo};
N → id N	N.n = N'.n ∪ {id.nom};
→ id	N.n = {id.nom};
I → id = E	id.lt = `buscaTipos`(id.nom); Si (id.lt = ∅) `MenError`(...);
→	Si (id.lt ∩ E.lt = ∅) `MenError`(...);
E → E + E	E.lt = E^1.lt ∩ E^2.lt; Si (E.lt = ∅) `MenError`(...);
→ id	E.lt = `buscaTipos`(id.nom); Si (E.lt = ∅) `MenError`(...);
→ (E , E)	E.lt = {tcomplejo};
→ cte	E.lt = {tentero};
→ ctr	E.lt = {treal};

Ejercicio 98

S → id L	`insertarListaOpciones`(id.nom, L.m, L.e, L.p, L.b);
L → L O	L.m = L'.m ∪ O.m; L.e = L'.e ∪ O.e; L.p = L'.p ∪ O.p; L.b = L'.b ∪ O.b;
→ ϵ	L.m = L.e = L.p = L.b = 0;
O → M	O.m = M.t; O.e = O.p = O.b = 0;
→ E	O.e = E.t; O.m = O.p = O.b = 0;
→ P	O.p = P.t; O.m = O.e = O.b = 0;
→ B	O.b = B.t; O.m = O.e = O.p = 0;
M → real	M.t = treal;
→ complejo	M.t = tcomplejo;
E → fija	E.t = tfija;
→ flotante	E.t = tflotante;
P → simple	P.t = tsimple;
→ doble	P.t =tdoble;
B → decimal	B.t = tdecimal;
→ binaria	B.t = tbinaria;

Ejercicio 100

P → D ; S	
→ D ; D	
→ id : T	`insertarTDS`(id.nom, T.t);
T → list_of T	T.t = tlista(T'.t);
→ char	T.t = tcarácter;
S → E en E	Si ¬ [(E^2.t = tlista(E^2.tel)) ∧ (E^1.t = E^2.tel)] `MenError`(...);
E → (L)	E.t = tlista(L.t);
→ literal	E.t = tcarácter;
→ id	Si ¬ [`obtenerTDS`(id.nom, E.t)] `MenError`(...);
L → E , L	L.t = L'.t; Si (L'.t ≠ E.t) `MenError`(...);
→ E	L.t = E.t;

5. Gestión de Memoria

101. Dado el siguiente fragmento de un programa en C:

```
int a[5] ;
real resultado ;
int imprime (int valor) {
  printf("El resultado es %d\n", valor);
}
int sum (int m) {
  int temporal                                    // MOSTRAR PILA
  if (m < 1) return 0;
  else {
    temporal = m+sum(m-1);
    return temporal ;
  }
}
int main() {                                      // MOSTRAR TDS
  a[1] = 2 ;
  resultado = sum (a[1]);
  imprime (resultado);
}
```

Se pide:

a) Suponiendo que la TDS está organizada en forma de pila, determinad su estado al llegar al punto marcado como `MOSTRAR TDS`.

b) Indicad el estado de la pila de ejecución, calculando el valor de los parámetros y de las variables locales, cada vez que la ejecución del programa pase por el punto marcado como `MOSTRAR PILA`.

102. Dado el siguiente fragmento de un programa en C, y suponiendo que la talla de los enteros es 1 y de los reales 2 :

```
|-----------------------------------------------------------|
|  int A (float p, int q, int r)                            |
|  { float x, y;                       // <== TDS           |
|    return p+q+r;  }                  // <== Pila          |
|                                                           |
|  int B (int p, float q)                                   |
|  { float x, y;                       // <== TDS           |
|    if (p == q) return A(q, p, 2);                         |
|    else B(p-1, q);  }                                     |
|                                                           |
|  int main()                                               |
|  { float x = 0.0, int y = 1;         // <== TDS           |
|    printf("%d", B(y, x)); }                               |
|-----------------------------------------------------------|
```

a) Mostrad el estado de la TDS, con todas sus tablas auxiliares, en el punto de control `TDS` durante el proceso de compilación.

b) Mostrad el estado de la pila de Registros de Activación (*frames*) cada vez que la ejecución del programa pase por el punto de control `Pila`.

103. * Dado el siguiente fragmento de un programa en C:

```
int fib[10] ;
int i= 0 ;
void printint (int x) {
  printf("%d\n", x);
}
int cal_fibo(int x) {
  if(x<2) return x;
  else                                          // MOSTRAR PILA
  return cal_fibo(x-1) + cal_fibo(x-2);
}
int main() {
  int num;
                                                // MOSTRAR TDS
  num = 3 ;
  fib[num] = cal_fibo(num);
  for(i=0; i<10 ; i++) printint(fib[i]);
}
```

Se pide:

a) Suponiendo que la Tabla de Símbolos está gestionada como una pila, determinad su contenido, indicando el valor de todos sus campos, al llegar al punto marcado como `MOSTRAR TDS`.

b) Indicad el contenido de la pila de ejecución, calculando el valor de los parámetros y de las variables locales, cada vez que la ejecución del programa pase por el punto marcado como `MOSTRAR PILA`. No olvidad indicar el valor del `frame pointer`.

104. * Dado el siguiente fragmento de un programa en C, y suponiendo que la talla de los enteros es 1, la de los reales 2 y la del segmento de gestión (enlaces de control y dirección de retorno) 2,

```
|-----------------------------------------------------|
|  float max[150];                                    |
|                                                     |
|  int f1 (int p1, int p2)                            |
|  { float v1, v2;                                    |
|    if (p1 <= p2) v1 = 1;           // <== Foto: TDS  |
|    else {                                           |
|      v2 = p1 -1;                   // <== Foto: Pila |
|      v1= f1(v2,p2) * p1;                            |
|    }                                                |
|    return v1 ; }                                    |
|                                                     |
|  int main()                                         |
|  { float n;                                         |
|    max[1] = n = 4;                 // <== Foto: TDS  |
|    printf("%d", f1(max[1], 2));                      |
|                                   // <== Foto: Pila  |
|  }                                                  |
|-----------------------------------------------------|
```

a) Mostrad el estado de la TDS, con todas sus tablas auxiliares, en el punto de control `Foto: TDS` durante el proceso de compilación.

b) Mostrad el estado de la pila de Registros de Activación (*frames*) cada vez que la ejecución del programa pase por el punto de control `Foto: Pila`.

105. Dado el siguiente fragmento de un programa en `PASCAL`:

```
program prueba;
var a: record a1: real; a2: real end;
    b: array [0..9, 0..99] of integer;
    c: integer;
function P (var x,y: real): integer;
var c: integer;
    fuction Q: integer;
    begin
      ....                  <=============== B
    end;
begin
    ...
    if x >= y then c := P(x,y+1)
    else c := Q;
    ...
end;
procedure R (var x,y: real);
var c: integer;
begin
    ...
    c := P(x,y);            <=============== A
    ...
end;
begin
  ...
  R(4,3);
  ...
end.
```

Mostrad el estado de la TDS en el momento de la compilación del punto de control **A**. Igualmente representad el estado de la pila de Registros de Activación en el proceso de ejecución del punto de control **B**. Considerad que la talla de los enteros es 2 y la de los reales es 4.

106. Contestad brevemente a las siguientes cuestiones:

a) ¿Qué acciones se deben realizar para comprobar que el tipo de los parámetros actuales de un llamada a una función coincide con el de los parámetros formales?

b) ¿Cómo se reserva espacio para el valor de retorno de una función?

c) Dónde se almacena la información relativa a la primera instrucción del segmento de código de una función?

d) ¿Cómo se reserva espacio en un Registro de Activación para las variables temporales?

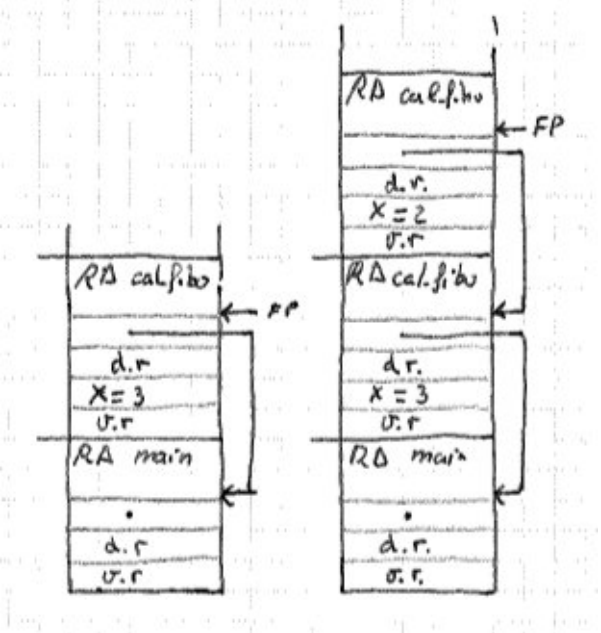
107. * Dado el siguiente programa `PASCAL` (lenguaje con estructura de bloques), y suponiendo un compilador con una Tabla de Símbolos gestionada como un pila apoyada en una Tabla de Bloques, indicad el contenido de la Tabla de Símbolos (lexema, categoría del objeto, tipo, posición, referencia), sus tablas auxiliares y la tabla de bloques en cada uno de los puntos señalados. Suponed que la *talla-enteros=2* y la *talla-reales=4*.

```
program tds;
var a: array[1..2,1..10] of integer;
    b: integer;                                          <---------- (1)
    function f1 (p1,p2: real): integer;
    var c: record c1: integer; c2: real; end;
        function f2 (p3: real): integer;
        var d: integer;
        begin                                            <---------- (2)
          f2:=p3;
        end;
        procedure f3;
        var d: integer;
        begin                                            <---------- (3)
          b:=5;
        end;
    begin
     f1:=int(p1*f2(p2));
    end;
begin                                                    <---------- (4)
  a[1,1]:=f1(b,b);  write(a[1,1]);
end.
```

5.1. Ejercicios resueltos

Ejercicio 103

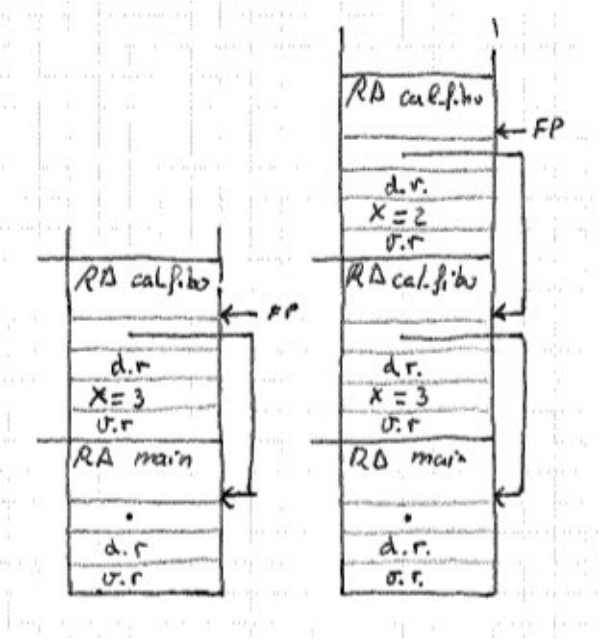
	nom		n	δ	tipo	Subtablas
1	fib	var_global	0	0	tvector	$\Rightarrow$ (10, tentero)
2	i	var_global	0	10	tentero	
3	printint	funcion	0	–	tfuncion	$\Rightarrow$ (tentero) $\rightarrow$ tvacio
4	cal_fibo	funcion	0	–	tfuncion	$\Rightarrow$ (tentero) $\rightarrow$ tentero
5	main	funcion	0	–	tfuncion	$\Rightarrow$ (tvacio) $\rightarrow$ tentero
6	num	var_local	1	0	tentero	



Ejercicio 104

	nom		n	δ	tipo	Subtablas
1	max	var_global	0	0	tvector	⇒ (150, treal)
2	f1	funcion	0	–	tfuncion	⇒ (tentero, tentero) → tentero
3	p2	parámetro	1	-3	tentero	
4	p1	parámetro	1	-4	tentero	
5	v1	var_local	1	0	treal	
6	v2	var_local	1	2	treal	

1	max	var_global	0	0	tvector	⇒ (150, treal)
2	f1	funcion	0	–	tfuncion	⇒ (tentero, tentero) → tentero
3	main	funcion	0	–	tfuncion	⇒ (tvacio) → tentero
4	n	var_local	1	0	treal	



Ejercicio 107

Punto-1[1]

TDB

0	1	2

TDS

1	a	variable	tarray	−21	→	2	tentero	→ 1	2	→ 1	10
2	b	variable	tentero	41							

[1]Se considera que el origen de todos los segmentos (variables, parámetros y código) comienza en 1.

Punto-2[2]

TDB

0	1	3
1	4	7
2	8	9

TDS

1	a	variable	tarray	−21	→	2	tentero	→	1	2	→ 1 | 10
2	b	variable	tentero	41							
3	f1	función	tentero	2	→	tentero		→	treal		
4	p1	parámetro	treal	1							
5	p2	parámetro	treal	5							
6	c	variable	trecord	1	→	c1	tentero	1	→	c2	treal | 3
7	f2	función	tentero	3	→	treal					
8	p3	parámetro	treal	1							
9	d	variable	tentero	1							

Punto-3

TDB

0	1	3
1	4	8
2	9	9

TDS

1	a	variable	tarray	−21	→	2	tentero	→	1	2	→ 1 | 10
2	b	variable	tentero	41							
3	f1	función	tentero	2	→	tentero		→	treal		
4	p1	parámetro	treal	1							
5	p2	parámetro	treal	5							
6	c	variable	trecord	1	→	c1	tentero	1	→	c2	treal | 3
7	f2	función	tentero	3	→	treal					
8	f3	función	tvacio	5							
9	d	variable	tentero	1							

Punto-4

TDB

0	1	3

TDS

1	a	variable	tarray	−21	→	2	tentero	→	1	2	→ 1 | 10
2	b	variable	tentero	41							
3	f1	función	tentero	2	→	tentero		→	treal		

[2]Se supone que la primera instrucción de cada bloque es un salto al comienzo de las instrucciones del bloque.

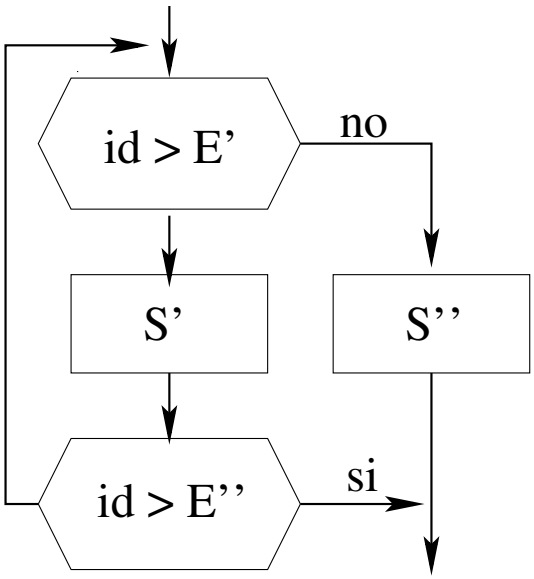
6. Generación de Código Intermedio

108. Dada la gramática, diseñad un ETDS que tenga en cuenta: 1) las comprobaciones semánticas de tipo; 2) la gestión de la TDS y la gestión de memoria para los objetos declarados; y 3) la generación de código intermedio para la instrucción propuesta.

D	→	T **id**	T	→	**union** { LC }	C	→	**id** . C
T	→	**int** | **double**	LC	→	LC ; T **id**	C	→	**id**
T	→	**struct** { LC }	LC	→	T **id**	S	→	**id** . C = E

109. Dada la instrucción *pseudowhile* cuya semántica viene definida por el diagrama de flujo que se adjunta. Diseñad un ETDS para la generación de código intermedio. Utilizad listas de referencias no satisfechas para el control de los saltos.

$$S \rightarrow \textit{pseudowhile id , } E' \textit{ until } E'' \textit{ do } S' \textit{ then } S''$$



110. * Diseñad un ETDS que genere código intermedio para la siguiente instrucción repetitiva:

I	→	`while` E `do` I R
R	→	`except` E' `do` I' | ϵ

La instrucción `while` tiene el comportamiento habitual: mientras que E sea cierto ejecuta I. La parte `except` permite ejecutar una instrucción alternativa, I', si su expresión, E', es cierta. La parte `except` se evalua ANTES de entrar en el cuerpo del bucle (I) del `while` y después de analizar la expresión E.

```
Ejemplo:    a=0;
            while a≤5 do
               { a++; write(a); }                ⇒      1 2 6
               except (a == 2) do { a=a+3; }
```

111. La siguiente gramática define una nueva instrucción que permite ejecutar (solo) las instrucciones de un determinado bloque de una lista de bloques. El número de bloque a ejecutar depende del valor entero de la expresión E: Si E vale 1 se ejecutará solo el primer bloque, si vale 2 el solo segundo, y así sucesivamente. Construid un ETDS que genere código intermedio para esta nueva instrucción:

$$
\begin{array}{lcll}
\text{I} & \rightarrow & \text{run E in LB} & \\
\text{LB} & \rightarrow & \{\ \text{LI}\ \}\ \text{LB} & \mid\ \epsilon \\
\text{LI} & \rightarrow & \text{I ; LI} & \mid\ \epsilon
\end{array}
$$

112. * Dada la siguiente gramática, diseñad un ETDS que genere código intermedio. La instrucción `yacase` (*yet another case*) es similar a la del PASCAL: si la `cte` coincide con la expresión `E` debe ejecutar la secuencia de instrucciones `I` asociada, y terminar la búsqueda en la lista de ítems. La instrucción `exit` supone la salida inmediata de la instrucción `yacase`.

$$
\begin{array}{lcl}
\text{I} & \rightarrow & \text{yacase E of L default I end} \\
\text{I} & \rightarrow & \text{I ; I} \\
\text{I} & \rightarrow & \text{exit}
\end{array}
\qquad
\begin{array}{lcl}
\text{L} & \rightarrow & \text{L ; cte : I} \\
\text{L} & \rightarrow & \text{cte : I} \\
\text{L} & \rightarrow & \text{else I}
\end{array}
$$

113. Diseñad un ETDS que genere código intermedio para el siguiente fragmento de una gramática:

$$
\begin{array}{lcl}
\text{I} & \rightarrow & \text{dependon E1 execute E2 times } \{\ \text{B} \\
\text{B} & \rightarrow & \text{num : LI ; B} \ \mid\ \} \\
\text{LI} & \rightarrow & \text{I ; LI} \ \mid\ \epsilon
\end{array}
$$

Donde la instrucción *dependon* ejecuta un bucle el número de veces indicado por $E2$. En cada iteración del bucle se ejecutarán todos los bloques de instrucciones precedidos del número cuyo valor coincide con el de la expresión $E1$. $E1$ y $E2$ no se modifican en el bucle.

Ejemplo La salida de este ejemplo será: `e e e`

```
dependon 5 execute 3 times {
   2: print(b);
   5: print(e);
   3: print(c); }
```

114. * Diseñad un ETDS que genere código intermedio para el siguiente fragmento de una gramática:

$$\text{S} \rightarrow \texttt{repeat-if}\ \text{E}\ \texttt{then}\ \text{S}\ \texttt{else}\ \text{S}\ \texttt{until}\ \text{E}$$

`repeat-if` es una instrucción repetitiva en la que, dependiendo del valor de la expresión E^1, se ejecutará S^1 en caso de que sea TRUE y S^2 en caso de FALSE. Este proceso se repetirá hasta que la expresión E^2 sea TRUE.

115. Diseñad un ETDS que genere código intermedio para el siguiente fragmento de una gramática:

$$
\begin{array}{lcl}
\text{I} & \rightarrow & \textbf{for}\ \text{id}\ \textbf{in}\ \text{id}\ \textbf{do}\ \text{I}\ \textbf{step}\ \text{E} \\
\text{I} & \rightarrow & \textbf{next}\ \mid \text{id} := \text{E}\ \mid \text{I ; I}
\end{array}
$$

donde `id-1` es una variable de tipo simple e `id-2` de tipo array.

La instrucción `for` es un bucle que en cada iteración asigna a la variable `id-1` un elemento del array `id-2`: desde el 0 (en la primera iteración) hasta el límite superior del array (en la última). La expresión `E` del `for` indica cuál será el próximo elemento del array a asignar al `id-1` y debe ser ≥ 1 (p.ej. un `step` 3 irá asignando a `id-1` los elementos 0, 3, 6 ... del array.

Cuando en el cuerpo del bucle se ejecuté la instrucción `next`, se abandonará inmediatamente la iteración en curso y se pasará al inicio de la siguiente.

116. Diseñad un ETDS que genere código intermedio para el bucle representado por la siguiente gramática:

$$\mathrm{I} \rightarrow \text{repeat if } \mathrm{E}_1 \text{ then } \mathrm{I}_1 \text{ else } \mathrm{I}_2 \text{ until } \mathrm{E}_2 \text{ updating } \mathrm{I}_3$$

Esta instrucción debe repetir el contenido del bucle hasta que $\mathrm{E}_2 = true$, en cuyo caso se debe salir del mismo. En el interior del blucle, si $\mathrm{E}_1 = true$ entonces se ejecutará la secuencia de instrucciones I_1 sino la I_2. Si $\mathrm{E}_2 = false$, y antes de repetir el bucle, se debe ejecutar la secuencia de instrucciones I_3.

117. * Diseñad un ETDS que realice las acciones semánticas necesarias para la comprobación de tipos y la generación de código intermedio para la siguiente gramática:

$$\begin{aligned} I &\rightarrow \texttt{select } \{ \ L \ \} \\ L &\rightarrow L \ ; \ E \ : \ I \ \mid \ E \ : \ I \end{aligned}$$

Donde I representa una secuencia de instrucciones y E una expresión lógica. La operación `select` debe evaluar la secuencia de expresiones lógicas en el orden que ocurran. Si alguna de ellas toma el valor `verdad` entonces se debe ejecutar la secuencia de instrucciones I que la acompaña y finalizar la evaluación de la instrucción `select`.

6.1. Ejercicios resueltos

Ejercicio 112

S ⇒ `yacase` E	Si (E.t ≠ tentero) MenError(.);
	L.pos = E.pos;
`of` L `default` S	CompletaLans(L.fin, Ω);
	CompletaLans(FusionaLans(L.b, S^1.b), Ω); S.b = `nil`;
⇒ S ; S	S.b = FusionaLans(S^1.b, S^2.b);
⇒ `exit`	S.b = CreaLans(Ω); Emite(goto ⊗);
L ⇒	L^1.pos = L.pos;
L ; cte	Si (cte.t ≠ tentero) MenError(.);
	L.sal = CreaLans(Ω); Emite(if L.pos ≠ cte.num goto ⊗);
: S	L.fin = CreaLans(Ω); Emite(goto ⊗);
	CompletaLans(L.sal, Ω);
	L.fin = FusionaLans(L.fin, L^1.fin);
	L.b = FusionaLans(L^1.b, S.b);
⇒ L ; cte	Si (cte.t ≠ tentero) MenError(.);
	L.sal = CreaLans(Ω); Emite(if L.pos ≠ cte.num goto ⊗);
: S	L.fin = CreaLans(Ω); Emite(goto ⊗);
	CompletaLans(L.sal, Ω); L.b = S.b;

La regla (L ⇒ `else` S) es redundante con la opción (`defaul` S) y no hace falta hacerla.

Ejercicio 114

S ⇒ `repeat-if`	S.l0 = Ω;
(E)	Si (E^1.t ≠ tlógico) MenError(.);
	S.l1 = CreaLans(Ω); Emite(if E^1.pos = 0 goto ⊗);
`then` S	S.l2 = CreaLans(Ω); Emite(goto ⊗);
	CompletaLans(S.l1, Ω);
`else` S	CompletaLans(S.l2, Ω);
`until` E	Si (E^2.t ≠ tlógico) MenError(.);
	Emite(if E^2.pos = 0 goto S.l0);

Ejercicio 117

I ⇒ `select` { L }	CompletaLans(L.fin, Ω);
L ⇒ L ; E :	Si (E.t ≠ tlógico) MenError(.);
	E.lf = CreaLans(Ω); Emite(if E.pos = 0 goto ⊗);
I	L.fin = FusionaLans(L^1.fin, CreaLans(Ω));
	Emite(goto ⊗); CompletaLans(E.lf, Ω);
L ⇒ E :	Si (E.t ≠ tlógico) MenError(.);
	E.lf = CreaLans(Ω); Emite(if E.pos = 0 goto ⊗);
I	L.fin = CreaLans(Ω); Emite(goto ⊗);
	CompletaLans(E.lf, Ω);

Ejercicio 110

I ⇒ `while`	I.ini = R.ini = Ω;
E	I.v = CreaLans (Ω); Emite (if E.pos = 1 goto ⊗);
	I.fin = CreaLans (Ω); Emite (goto ⊗);
	R.f = Ω;
I	Emite (goto I.ini); CompletaLans (I.v, Ω);
R	CompletaLans (I.fin, Ω);
R ⇒ E	Emite (if E.pos = 0 goto R.f);
I	Emite (goto R.ini);;

7. Optimización de Código Intermedio

118. Dado el siguiente fragmento de un bloque básico, aplicad las optimizaciones locales mediante la construcción de su GDA y la posterior reconstrucción del código.

$t_1 = 2$	$t_5 = t_4 + 1$	$z = t_8$
$t_2 = y + t_1$	$t_6 = x - t_5$	$t_9 = 10$
$x = t_2$	$y = t_6$	$t_{10} = x - t_9$
$t_3 = 1$	$t_7 = 2$	$w = t_{10}$
$t_4 = t_3 * 9$	$t_8 = y + t_7$	

119. Dado el siguiente fragmento de código intermedio, indicad los bloques básicos que forman el bucle y sus familias de variable de inducción (con sus ternas asociadas). A continuación, aplicad los algoritmos de extracción de código invariante, reducción de intensidad y de eliminación de variables de inducción.

100	$j = 1$	105	$t_3 = t_2 + j$	110	$a[t_4] = j$
101	$i = 0$	106	$a[t_1] = t_3$	111	$i = i + 1$
102	$if\ i > N\ goto\ 108$	107	$goto\ 111$	112	$if\ i \leq 100\ goto\ 102$
103	$t_1 = i * 4$	108	$t_4 = i * 4$	113	...
104	$t_2 = a[t_1]$	109	$j = 0$		

120. Dado el siguiente fragmento de código intermedio:

100	$i := 0$	109	$t_7 := t_6 + j$	118	$t_{14} := j * 20$
101	$j := 10$	110	$t_8 := t_7 * 4$	119	$t_{15} := t_{14} + i$
102	$if\ i > 5\ goto\ 113$	111	$a[t_8] := t_5$	120	$t_{16} := t_{15} * 4$
103	$t_1 := i * 20$	112	$goto\ 122$	121	$a[t_{16}] := t_{13}$
104	$t_2 := t_1 + j$	113	$t_9 := j * 20$	122	$t_{17} := 1$
105	$t_3 := t_2 * 4$	114	$t_{10} := t_9 + i$	123	$t_{18} := i + t_{17}$
106	$t_4 := a[t_3]$	115	$t_{11} := t_{10} * 4$	124	$i := t_{18}$
107	$t_5 := t_4 + j$	116	$t_{12} := a[t_{11}]$	125	$if\ i < 10\ goto\ 102$
108	$t_6 := i * 20$	117	$t_{13} := t_{12} + j$		...

a) Determinad los bloques básicos y el grafo de flujo. A partir de los GDAs, reconstruid el código aplicando las optimizaciones locales.

b) Indicad los bloques básicos que forman el bucle y sus familias de variables de inducción (con sus ternas asociadas). A continuación, aplicad los algoritmos de extracción de código invariante, reducción de intensidad y de eliminación de variables de inducción.

121. Dado el siguiente fragmento de código intermedio de un bloque básico, aplicad las optimizaciones locales a partir de su GDA. A la salida del bloque solo estarán activas las variables: `A i`.

(100)	$t_0 := 0$
(101)	$i := t_0$
(102)	$y := k$
(103)	$x := 0$
(104)	$t_1 := y + x$
(105)	$y := t_1$
(106)	$t_2 := i * 4$
(107)	$t_3 := t_2 + 2$
(108)	$t_4 := t_3 * y$
(109)	$t_5 := A[t_4]$
(110)	$x := x + t_4$
(111)	$i := y$
(112)	$if\ \ i < x\ \ goto\ \ 200$

122. Dado el siguiente fragmento de código intermedio:

(100)	$s := 0$
(101)	$m := 1$
(102)	$t_1 := m * 2$
(103)	$ini := size * 2$
(104)	$t_2 := t_1 - ini$
(105)	$if\ \ x > y\ \ goto\ \ 108$
(106)	$s := a[t_2]$
(107)	$goto\ \ 109$
(108)	$s := b[t_2]$
(109)	$tot := tot + s$
(110)	$m := m + 2$
(111)	$if\ \ m < 100\ \ goto\ \ 102$
(112)	$print(s)$

a) Determinad los bloques básicos que forman el/los bucle/s. Extrae el código invariante. Indicad las variables de inducción y sus ternas asociadas.

b) Aplicad el algoritmo de reducción de intensidad.

c) Aplicad el algoritmo de eliminación de variables de inducción.

123. Dado el siguiente fragmento de código intermedio de un bloque básico, aplicad las optimizaciones locales a partir de su GDA. A la salida del bloque solo estarán activas las variables: `a, b, x, y, z`.

(100)	$t_1 := 10$	(106)	$t_5 := c + x$	(112)	$z := x + y$
(101)	$t_2 := t_1 * 2$	(107)	$x := t_5$	(113)	$t_8 := 1$
(102)	$c := t_2$	(108)	$t_6 := d * y$	(114)	$a := z + t_9$
(103)	$t_3 := 5$	(109)	$b := c + x$	(115)	$t_9 := 1$
(104)	$t_4 := t_3 + 2$	(110)	$t_7 := d * y$	(116)	$z := x * y$
(105)	$d := t_4$	(111)	$y := t_7$	(117)	$a := b + 1$

124. Dado el siguiente fragmento de código intermedio:

(100)	$k := 0$	(105)	$t_4 := 5$	(110)	$t_9 := t_3 + t_8$
(101)	$m := 100$	(106)	$t_5 := t_4 * 2$	(111)	$A[t_2] := t_9$
(102)	$t_1 := k * 2$	(107)	$t_6 := t_1 + t_5$	(112)	$if\ \ k > m\ \ goto\ \ 115$
(103)	$t_2 := t_1 * 2$	(108)	$t_7 := t6 * 2$	(113)	$k := k + 1$
(104)	$t_3 := A[t_2]$	(109)	$t_8 := A[t_7]$	(114)	$goto\ \ 102$

a) Determinad los bloques básicos que forman el bucle. Extraed el código invariante e indicad las variables de inducción y sus ternas asociadas.

b) Aplicad el algoritmo de reducción de intensidad.

c) Aplicad el algoritmo de eliminación de variables de inducción.

125. Dado el siguiente fragmento de código intermedio de un bloque básico, aplicad las optimizaciones locales a partir de su GDA. A la salida del bloque, aparte de las variables `i, j, A, B,` solo estará activa la variable: `k`.

(100)	$t_1 = 7$	(103)	$t_4 = A[t_3]$	(106)	$t_7 = t_6$	(109)	$t_{10} = t_7 * t_9$
(101)	$t_2 = t_1 * 4$	(104)	$t_5 = 7$	(107)	$t_8 = 0$	(110)	$t_{11} = B[t_{10}]$
(102)	$t_3 = i * t_2$	(105)	$t_6 = t_5 * 4$	(108)	$t_9 = j + t_8$	(111)	$k = t_4 + t_{11}$

126. Dado el siguiente fragmento de código intermedio:

(100)	$i = 5$	(104)	$t_2 = 10$	(108)	$x = x + t_5$	(112)	$t_7 = A[t_6]$
(101)	$j = 20$	(105)	$t_3 = t_2 * 2$	(109)	$j = j - 2$	(113)	$x = x + t_7$
(102)	$x = 0$	(106)	$t_4 = t_1 + t_3$	(110)	$if\ \ i > 10\ goto\ 116$	(114)	$i = i + 2$
(103)	$t_1 = j * 4$	(107)	$t_5 = B[t_4]$	(111)	$t_6 = i * 2$	(115)	$goto\ \ 103$

a) Determinad los bloques básicos que forman el/los bucle/s. Extraed el código invariante. Indicad las variables de inducción y sus ternas asociadas.

b) Aplicad el algoritmo de reducción de intensidad.

c) Aplicad el algoritmo de eliminación de variables de inducción.

127. Dado el siguiente fragmento de código intermedio de un bloque básico, aplicad las optimizaciones locales a partir de su GDA. A la salida del bloque, aparte de las variables `i, j, A, B,` solo estará activa la variable: `k`.

(100)	$t_1 = 7$	(103)	$t_4 = A[t_3]$	(106)	$t_7 = t_6$	(109)	$t_{10} = t_7 * t_9$
(101)	$t_2 = t_1 * 4$	(104)	$t_5 = 7$	(107)	$t_8 = 0$	(110)	$t_{11} = B[t_{10}]$
(102)	$t_3 = i * t_2$	(105)	$t_6 = t_5 * 4$	(108)	$t_9 = j + t_8$	(111)	$k = t_4 + t_{11}$

128. Dado el siguiente fragmento de código intermedio:

(100)	$i = 5$	(104)	$t_2 = 10$	(108)	$x = x + t_5$	(112)	$t_7 = A[t_6]$
(101)	$j = 20$	(105)	$t_3 = t_2 * 2$	(109)	$j = j - 2$	(113)	$x = x + t_7$
(102)	$x = 0$	(106)	$t_4 = t_1 + t_3$	(110)	$if\ \ i > 10\ goto\ 116$	(114)	$i = i + 2$
(103)	$t_1 = j * 4$	(107)	$t_5 = B[t_4]$	(111)	$t_6 = i * 2$	(115)	$goto\ \ 103$

a) Determinad los bloques básicos que forman el/los bucle/s. Extraed el código invariante. Indicad las variables de inducción y sus ternas asociadas.

b) Aplicad el algoritmo de reducción de intensidad.

c) Aplicad el algoritmo de eliminación de variables de inducción.

129. Dado el siguiente fragmento de un bloque básico, aplicad las optimizaciones locales mediante la construcción de su GDA y la posterior reconstrucción del código. Considerad activas a la salida del bloque las variables: `x, y, k`

$t_2 = 3$	$y = t_6$	$t_{11} = 7$	$t_{15} = 40$
$t_3 = y + t_2$	$t_7 = 1$	$t_{12} = t_{11} + x$	$t_{16} = t_{15} + k$
$t_4 = 4$	$t_8 = t_7 * x$	$k = t_{12}$	$x = t_{16}$
$t_5 = t_4 * 10$	$t_9 = 0$	$t_{13} = 7$	
$t_6 = t_5 + k$	$t_{10} = t_9 + t_8$	$t_{14} = t_{13} + x$	

130. Dado el siguiente fragmento de código intermedio, indicad los bloques básicos que forman el bucle y sus variables de inducción (con sus ternas asociadas). A continuación, aplicad las optimizaciones globales de extracción de código invariante, reducción de intensidad y de eliminación de variables de inducción.

100	$j = 10$	104	$t_5 = a[t_4]$	108	$x = t_8 + t_5$	112	$j = j + 1$
101	$i = 1$	105	$t_6 = 10 + i$	109	$if\ x \geq M\ goto\ 112$	113	$if\ i > 20\ goto\ 115$
102	$t_3 = 70 + j$	106	$t_7 = t_6 * 2$	110	$i = i + 1$	114	$if\ j \leq 10\ goto\ 102$
103	$t_4 = t_3 * 2$	107	$t_8 = b[t_7]$	111	$goto\ 113$	115	...