Práctica PL

Elena Kaloyanova Popova y Álvaro Borja Velasco García

2018

Índice general

Capítulo 1

Introducción

Esta práctica consistirá en el desarrollo de un procesador de lenguajes sobre el siguiente lenguaje:

Capítulo 2

Fase 1: Analizador léxico

2.1. Clases Léxicas

Todo programa consta de dos secciones: una para las declaraciones y otra para las instrucciones, separadas por un token «&&». La sección de declaraciones está formada por una serie de declaraciones compuestas por el nombre de tipo y el de variable y separadas por un punto y coma. La sección de instrucciones, por su parte, consta de una serie de asignaciones (variable=expresión), separadas también por un punto y coma. Las clases léxicas que hemos considerado para representar los tokens del lenguaje son las siguientes:

- **SEC:** Representa el seccionador de las dos partes del programa («&&»).
- **NUM:** Palabra reservada «num».
- **BOOL:** Palabra reservada «bool».
- **VAR:** Representa el nombre de la variable. Comienza necesariamente por una letra, seguida por una secuencia de cero o más letras, dígitos o el símbolo «_».
- **ASIG:** Representa el signo igual de las asignaciones.
- **NXT:** Representa el signo «;» que marca el comienzo de la siguiente instrucción.
- **TRUE:** Palabra reservada «true».
- **FALSE:** Palabra reservada «false».

- **NUMR:** Representa un número real. Puede empezar opcionalmente con un signo seguido de una secuencia de uno o más digitos cualesquiera, pudiendo poner ceros no significativos a la izquierda. Puede opcionalmente estar seguido por una parte decimal y/o una parte exponencial.
- **MAS:** Operador suma (\+).
- **MENOS:** Operador resta (\-).
- **POR:** Operador multiplicación (*).
- **DIV:** Operador división (/).
- **AND:** Palabra reservada «and».
- **OR:** Palabra reservada «or».
- **NOT:** Palabra reservada «not».
- **MAY:** Operador mayor (>).
- **MEN:** Operador menor (<).
- **MAYI:** Operador mayor o igual (>=).
- **MENI:** Operador menor o igual (<=).
- **IGUAL:** Operador igual a (==).
- **DIST:** Operador distinto a (!=).
- **PAP:** Signo de apertura de paréntesis.
- **PCI:** Signo de cierre de paréntesis.
- **EOF:** Representa el final de fichero.

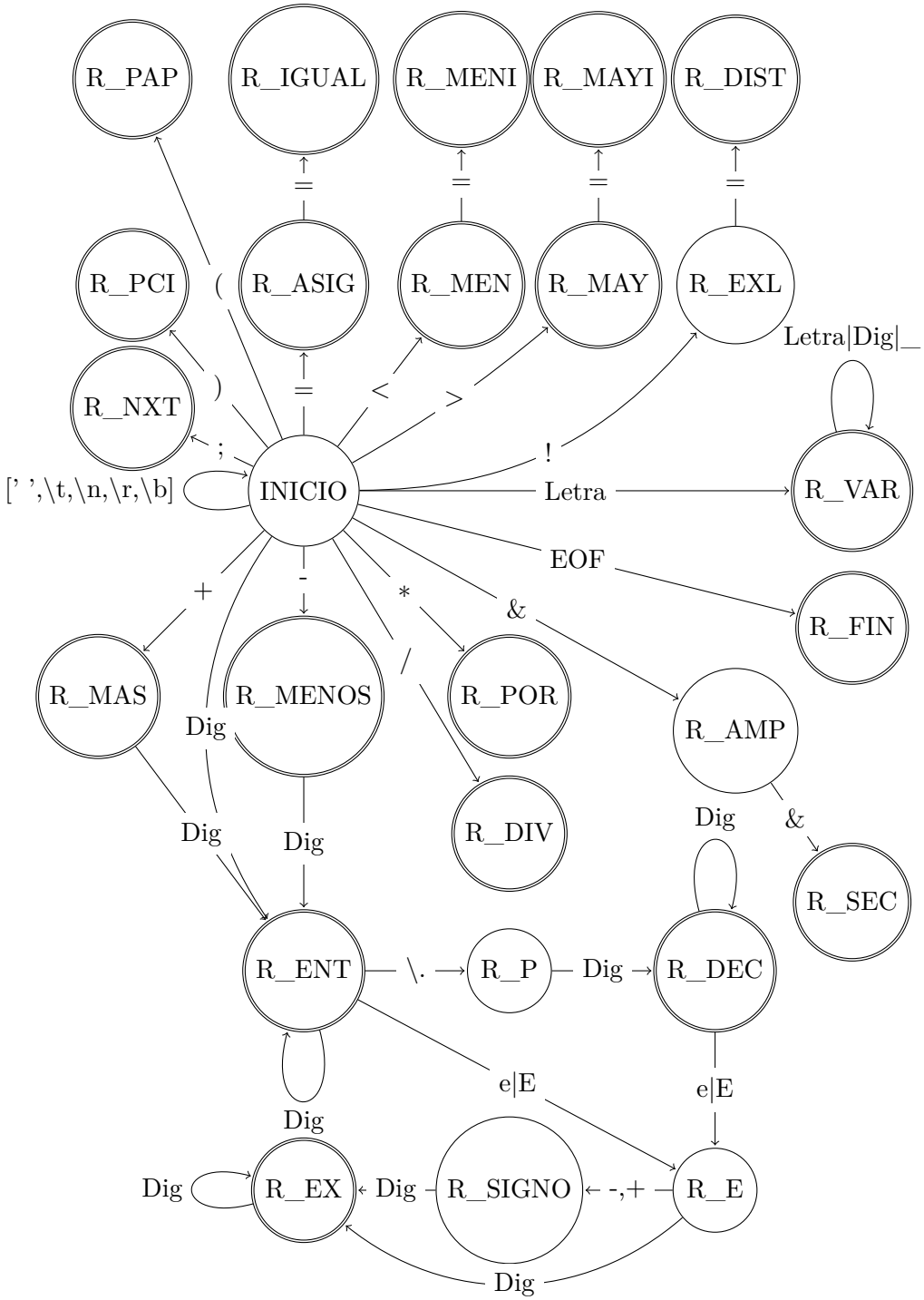
2.2. Especificación Formal

Las definiciones regulares correspondientes a las clases léxicas definidas son:

(⋆) **SEC:** [&][&]
(⋆) **VAR:** LETRA([LETRA|DIG|_]*)
LETRA: ([a-z,A-Z])

DIG: ([0-9])
(⋆) **NUM:** ([n][u][m])
(⋆) **BOOL:** ([b][o][o][l])
(⋆) **TRUE:** ([t][r][u][e])
(⋆) **FALSE:** ([f][a][l][s][e])
(⋆) **NUMR:** SIGNO?(DIG+(DEC)?(EX)?)
DEC: (\.)DIG+
EX: [e|E](SIGNO?DIG+)
SIGNO: [\+,\-]
DIG: [0-9]
(⋆) **AND:** ([a][n][d])
(⋆) **OR:** ([o][r])
(⋆) **NOT:** ([n][o][t])
(⋆) **MAS:** (\+)
(⋆) **MENOS:** (\-)
(⋆) **DIV:** (/)
(⋆) **POR:** (*)
(⋆) **MAY:** (>)
(⋆) **MEN:** (<)
(⋆) **MAYI:** ([>][=])
(⋆) **MENI:** ([<][=])
(⋆) **IGUAL:** ([=][=])
(⋆) **DIST:** ([!][=])
(⋆) **ASIG:** (=)
(⋆) **NXT:** (;)
(⋆) **PAP:** (\()
(⋆) **PCIERRE:** (\))
(⋆) **SEP:** [« »,\t,\n,\r,\b]

2.3. Diseño


R_PAP
R_IGUAL
R_MENI
R_MAYI
R_DIST
=
=
=
=
R_PCI
(
R_ASIG
R_MEN
R_MAY
R_EXL
Letra|Dig|_
R_NXT
)
=
<
>
;
!
[' ',\t,\n,\r,\b]
INICIO
Letra
R_VAR
EOF
R_FIN
+
-
*
&
R_MAS
R_MENOS
/
R_POR
R_AMP
Dig
Dig
Dig
R_DIV
Dig
&
R_SEC
R_ENT
\.
R_P
Dig
R_DEC
e|E
e|E
Dig
Dig
R_EX
Dig
R_SIGNO
-,+
R_E
Dig

Capítulo 3

Fase 2: Analizador sintáctico

En esta fase desarrollaremos el analizador sintáctico descendente predictivo para el lenguaje descrito en la primera fase.

3.1. Gramática incontextual

Empezaremos definiendo la gramática incontextual que define el lenguaje. Los operadores que utiliza nuestro lenguaje aparecen en la tabla 3.1.

Operador	Prioridad	Tipo	Asociatividad
+,-	0	Binario infijo	Asocia Izquierda
and or	1	Binarios infijos	Asocia Derecha No asocia
Relacionales	2	Binario infijo	No asocia
*,/	3	Binario infijo	Asocia Izquierda
- not	4	Unarios prefijos	Asocia No asocia

Cuadro 3.1: Operadores

La gramática incotextual obtenida apartir de la definición y los operadores es la siguiente:

Programa → LDsSECLIs
LDs → LDsNXTDeclaracion
LDs → Declaracion
Declaracion → NUMVAR
Declaracion → BOOLVAR
LIs → LIsNXTInstruccion

LIs → Instruccion
Instruccion → VAR ASIG EXP0
EXP0 → EXP0 OP0 EXP1
EXP0 → EXP1
EXP1 → EXP2 AND EXP1
EXP1 → EXP2 OR EXP2
EXP1 → EXP2
EXP2 → EXP3 OP2 EXP3
EXP2 → EXP3
EXP3 → EXP3 OP2 EXP4
EXP3 → EXP4
EXP4 → MENOS EXP4
EXP4 → NOT EXP5
EXP5 → NUMR
EXP5 → VAR
EXP5 → PAPEXP0PCIERRE
OP0 → MAS
OP0 → MENOS
OP2 → MAY
OP2 → MEN
OP2 → MAYI
OP2 → MENI
OP2 → IGUAL
OP2 → DIST
OP3 → POR
OP3 → DIV

Necesitamos transformar la gramática a una LL(1). Una vez transformada, la gramática queda de la siguiente manera:

Programa → LDsSECLIs
LDs → LDsNXTDeclaracion
LDs → Declaracion
Declaracion → NUMVAR
Declaracion → BOOLVAR
LIs → LIsNXTInstruccion
LIs → Instruccion
Instruccion → VAR ASIG EXP0
EXP0 → EXP0 OP0 EXP1
EXP0 → EXP1
EXP1 → EXP2 R1
R1 → AND EXP1

R1 → OR EXP2
R1 → ϵ
EXP2 → EXP3 R2
R2 → OP2 EXP3
R2 → ϵ
EXP3 → EXP3 OP2 EXP4
EXP3 → EXP4
EXP4 → MENOS EXP4
EXP4 → NOT EXP5
EXP5 → NUMR
EXP5 → VAR
EXP5 → PAPEXP0PCIERRE
OP0 → MAS
OP0 → MENOS
OP2 → MAY
OP2 → MEN
OP2 → MAYI
OP2 → MENI
OP2 → IGUAL
OP2 → DIST
OP3 → POR
OP3 → DIV