Procesadores de lenguajes. Práctica 2.

IMPLEMENTACIÓN DEL ANALIZADOR LÉXICO DEL LENGUAJE TINTO

1. Objetivos

- Describir las clases necesarias para desarrollar un analizador léxico genérico.
- Describir la implementación del analizador léxico del lenguaje de programación Tinto.

2. Código a utilizar

- Hay 7 ficheros en dos paquetes.
- *tinto.parser* -> clases dedicadas a desarrollar los analizadores léxicos.
  - Cuatro clases que permiten desarrollar analizadores léxicos de forma genérica
  - Dos clases que desarrollan el analizador léxico del lenguaje de programación *Tinto* como caso particular.
- Paquete *tinto* -> contiene *TintoCompiler* que en este caso sólo comprueba el funcionamiento del analizador léxico.
- Código práctica:
- https://uhu.es/francisco.moreno/gii_pl/practicas/practica02/practica02.rar

3. Analizador léxico genérico

- Cuatro clases que permiten la descripción genérica de los analizadores léxicos:
  - *tinto.parser.Token*
  - *tinto.parser.LexicalError*
  - *tinto.parser.BufferedCharStream*
  - *tinto.parser.Lexer*

3.1 tinto.parser.Token

- Describe un componente léxico.
- Atributos de los componentes léxicos como campos de la clase:
  - *kind*: categoría léxica a la que pertenece
  - *lexeme*: lexema del componente
  - *row*: fila en la que comienza el componente en el fichero de entrada
  - *column*: columna de comienzo en el fichero de entrada.
- El constructor de la clase recibe estos parámetros como argumentos.
- El resto de los métodos es para acceder a los atributos.

3.2 tinto.parser.LexicalError

- Describe un error léxico.
- Para identificar el error se utiliza el carácter que lo provoca y la fila y columna en la que se encuentra en el fichero de entrada.
- Con la información se crea el mensaje de error asociado a los errores de tipo léxico.

3.3 tinto.parser.BufferedCharStream

- Desarrolla un flujo de datos de entrada basado en un doble buffer que optimiza el acceso a un fichero de caracteres y permite retroceder en la lectura.
- El analizador léxico accede al fichero de entrada a través de esta clase, lo que le permite tanto avanzar como retroceder en el flujo de caracteres.
- La clase calcula la fila y la columna a la que pertenecen cada uno de los caracteres almacenados en cada momento.

3.3 tinto.parser.BufferedCharStream

- Métodos:
  - *getNextChar()* siguiente carácter de la cadena de entrada
  - *getRow()* fila del último carácter leído
  - *getColumn()* columna del último carácter leído
  - *retract(int)* retrocede un número de caracteres en el flujo de entrada
  - *close()* cierra el flujo de datos

3.3 tinto.parser.BufferedCharStream

- Internamente la clase dispone de los siguientes campos:
  - *stream*: almacena el flujo de datos del fichero de entrada
  - *buffer*: almacena los bytes leídos del flujo de entrada
  - *row*: contiene el número de fila asociado a cada carácter almacenado
  - *column*: contiene el número de columna asociado a cada carácter almacenado
  - *index*: posición del último carácter solicitado por el analizador léxico
  - *half*: indica si la última lectura del flujo de entrada se almacenó en la parte baja del buffer o en la parte alta

3.3 tinto.parser.BufferedCharStream

- El método interno *load()* lee un bloque de 1024 bytes del flujo de entrada y lo almacena en la parte alta o baja del buffer.
- El método *getNextChar()* además de devolver el siguiente carácter, se encarga de detectar cuando es necesario cargar otro bloque de datos.

3.4 tinto.parser.Lexer

- Clase abstracta que desarrolla un analizador léxico genérico basado en una máquina discriminadora determinista.
- Se requieren los siguientes métodos abstractos:
  - *transition(int, char):* contiene las transiciones del autómata finito determinista que describe el analizador léxico
  - *isFinal(int):* indica cuales son los estados finales del autómata
  - *getToken(int, String, int, int):* genera el componente léxico asociado a cada estado final
- El método más importante de la clase es *getNextToken()* que obtiene el siguiente componente léxico del fichero de entrada. Este método llama al método privado *tokenize()* hasta obtener un valor distinto de nulo.

3.4 tinto.parser.Lexer

- El método tokenize() desarrolla el comportamiento de una máquina discriminadora determinista:
  - Parte del estado inicial
  - Realiza todas las transiciones posibles del autómata hasta alcanzar un estado en el que no existan transiciones para el carácter correspondiente
  - Retrocede hasta el último estado final alcanzado en las transiciones devolviendo al flujo de entrada los caracteres leídos en estas últimas transiciones
  - Y devuelve el componente léxico asociado al estado final alcanzado y a la lista de caracteres leídos

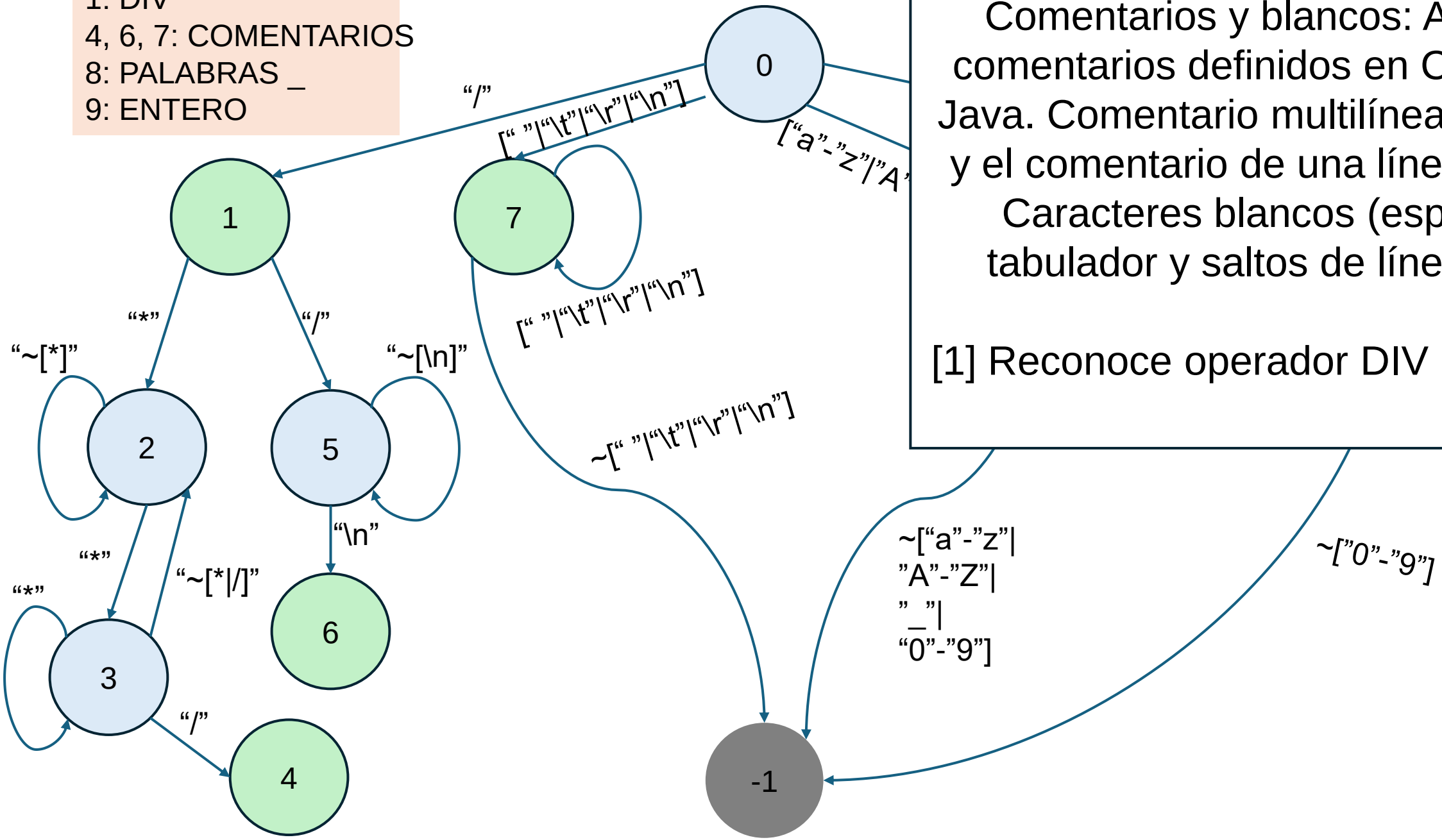
3.4 tinto.parser.Lexer

- Cuando el estado final corresponde a un comentario o a un blanco del lenguaje el método devuelve el valor nulo.
- El funcionamiento del método consiste en almacenar en la variable *lexeme* los caracteres que han permitido alcanzar un estado final
- Y en la variable *tainting* los caracteres leídos desde el último estado final hasta el estado actual.

4. El Analizador léxico de Tinto

- La implementación del analizador léxico del lenguaje de programación *Tinto* consta de dos archivos: *TokenConstants* y *TintoLexer.*
- *TokenConstants* interfaz que define las constantes asociadas a las diferentes categorías léxicas del lenguaje Tinto.
- *TintoLexer* extiende de la clase *Lexer* desarrollando el analizador léxico de Tinto. Para ello desarrolla los métodos *transition(int, char), isFinal(int)* y *getToken(int, String, int, int)* que describen el autómata finito determinista asociado a la especificación léxica del lenguaje *Tinto.*
- Describimos el autómata en las siguientes diapositivas.

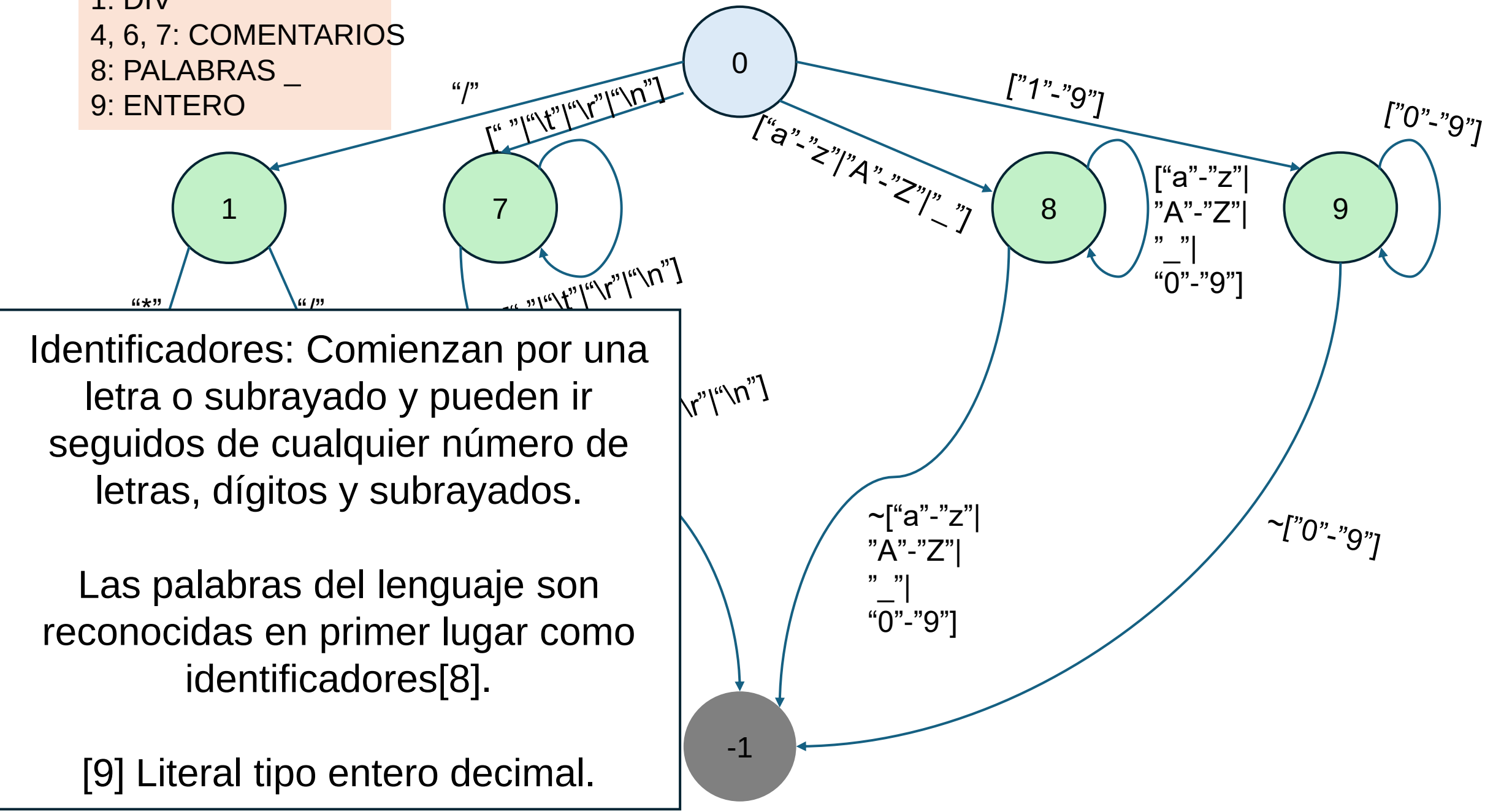
1: DIV
4, 6, 7: COMENTARIOS
8: PALABRAS _
9: ENTERO



Comentarios y blancos: Acepta comentarios definidos en C, C++ y Java. Comentario multilínea (/* */)[4] y el comentario de una línea (//)[6]. Caracteres blancos (espacio, tabulador y saltos de línea) [7].

[1] Reconoce operador DIV (división)

1: DIV
4, 6, 7: COMENTARIOS
8: PALABRAS _
9: ENTERO



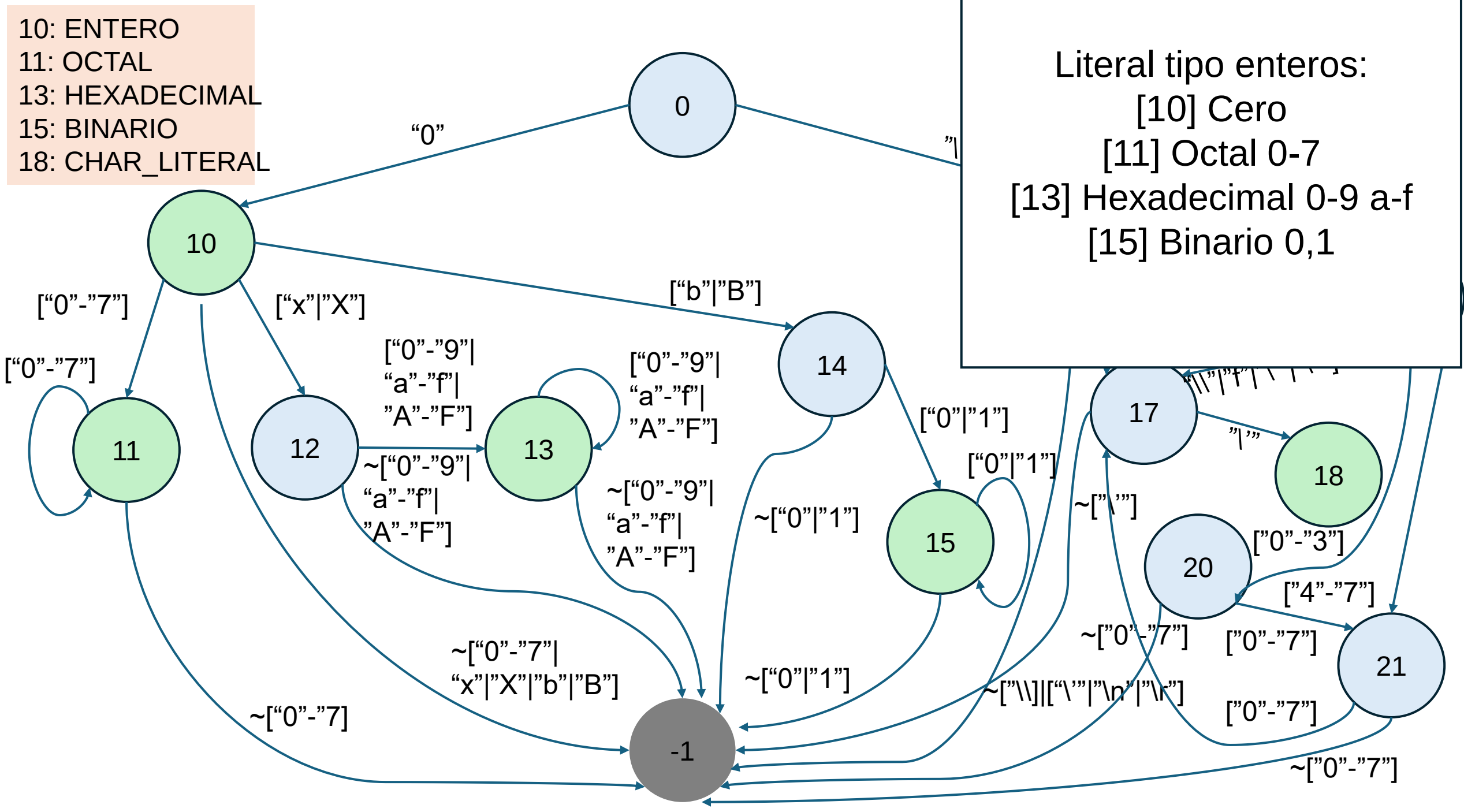
Identificadores: Comienzan por una letra o subrayado y pueden ir seguidos de cualquier número de letras, dígitos y subrayados.

Las palabras del lenguaje son reconocidas en primer lugar como identificadores[8].

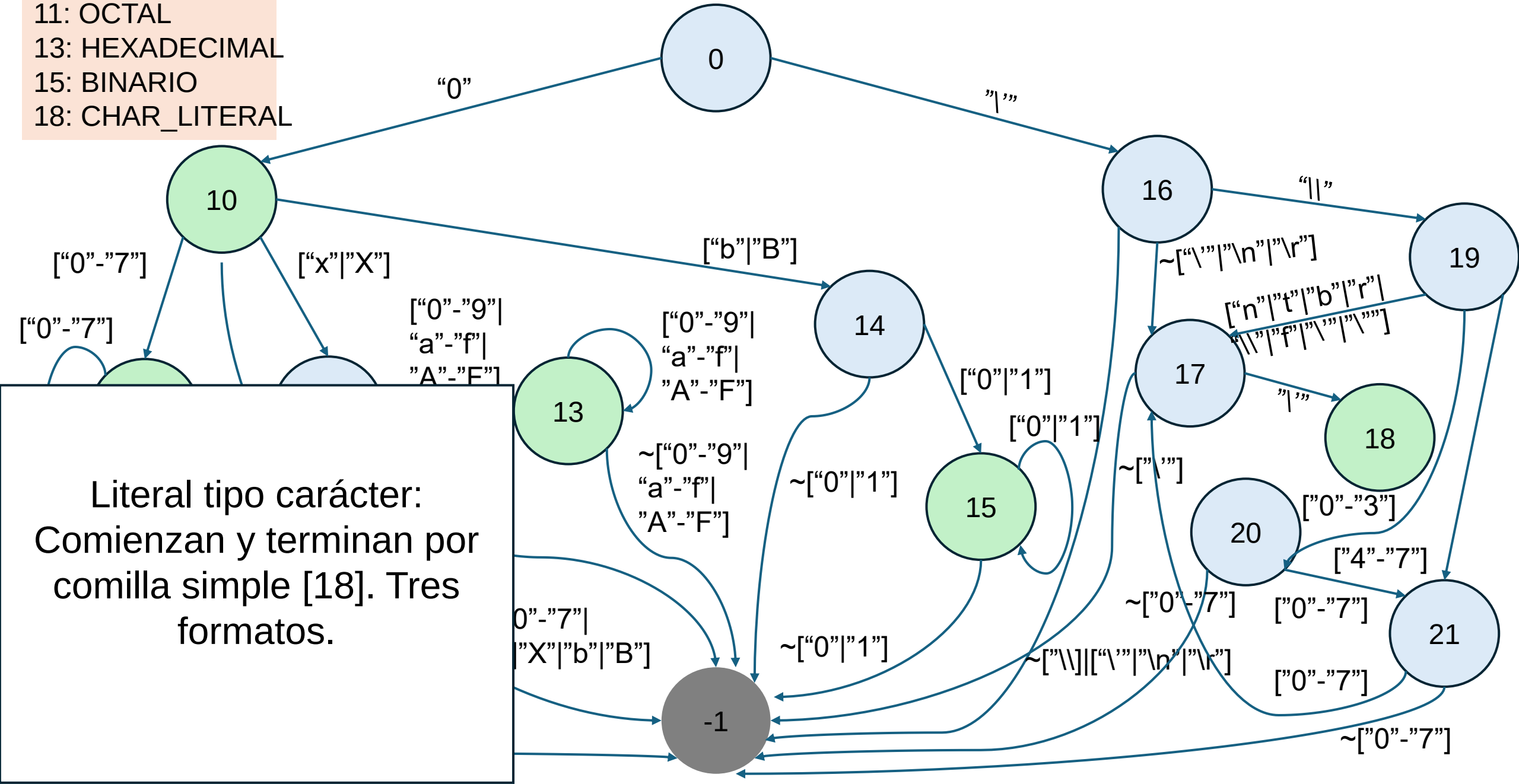
[9] Literal tipo entero decimal.

Palabras:

- BOOLEAN
- CHAR
- ELSE
- FALSE
- IF
- IMPORT
- INT
- LIBRARY
- NATIVE
- PRIVATE
- PUBLIC
- RETURN
- TRUE
- VOID
- WHILE
- IDENTIFIER

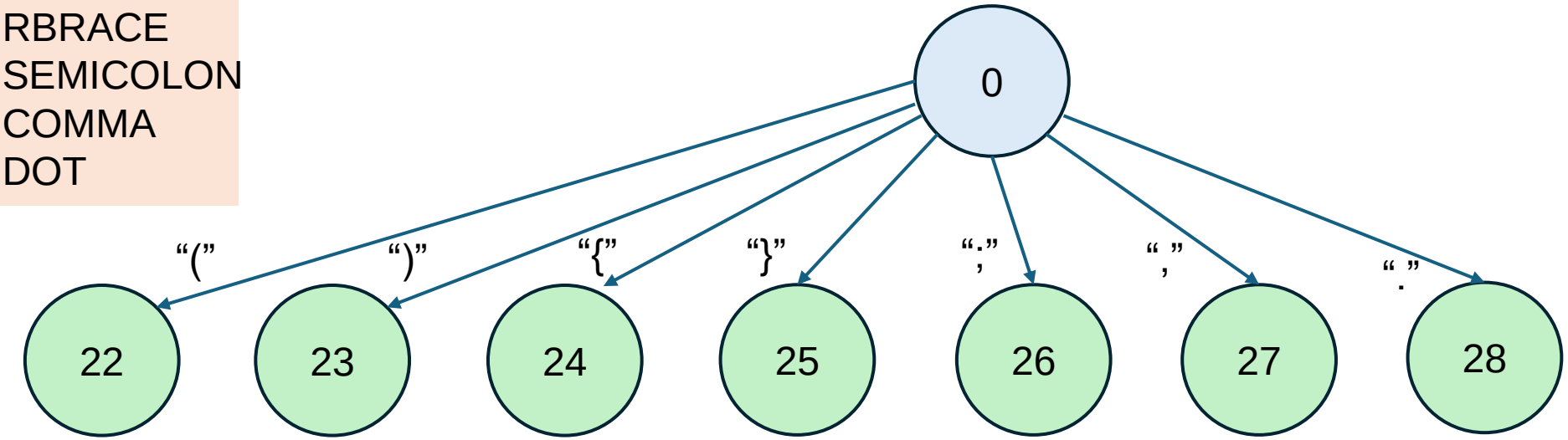

10: ENTERO
11: OCTAL
13: HEXADECIMAL
15: BINARIO
18: CHAR_LITERAL
Literal tipo enteros:
[10] Cero
[11] Octal 0-7
[13] Hexadecimal 0-9 a-f
[15] Binario 0,1
0
"0"
10
["0"-"7"]
["x"|"X"]
["b"|"B"]
["0"-"7"]
11
12
["0"-"9"|
"a"-"f"|
"A"-"F"]
13
["0"-"9"|
"a"-"f"|
"A"-"F"]
14
["0"|"1"]
~["0"-"9"|
"a"-"f"|
"A"-"F"]
~["0"-"9"|
"a"-"f"|
"A"-"F"]
~["0"|"1"]
["0"|"1"]
15
17
"|'"
18
~["\'"]
20
["0"-"3"]
["4"-"7"]
~["0"-"7"]
["0"-"7"]
21
~["\\]|["\'"|"\n"|"\r"]
["0"-"7"]
~["0"-"7"]
~["0"-"7"|
"x"|"X"|"b"|"B"]
~["0"|"1"]
~["0"-"7]
-1

10: ENTERO
11: OCTAL
13: HEXADECIMAL
15: BINARIO
18: CHAR_LITERAL

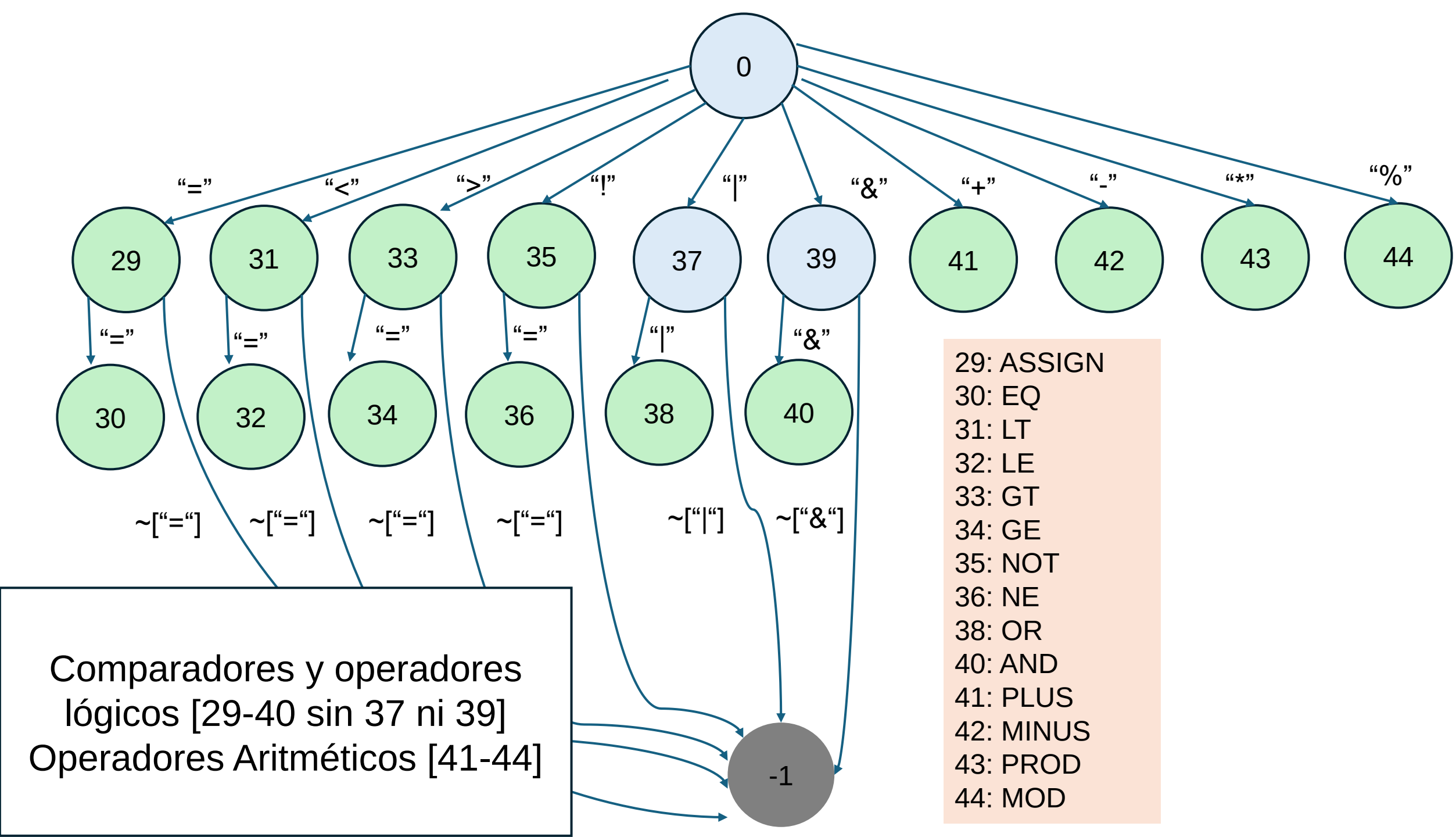


Literal tipo carácter: Comienzan y terminan por comilla simple [18]. Tres formatos.

22: LPAREN
23: RPAREN
24: LBRACE
25: RBRACE
26: SEMICOLON
27: COMMA
28: DOT



Separadores del lenguaje [22-28]


0
“=”
“<”
“>”
“!”
“|”
“&”
“+”
“-”
“*”
“%”
29
31
33
35
37
39
41
42
43
44
“=”
“=”
“=”
“=”
“|”
“&”
30
32
34
36
38
40
~[“=“]
~[“=“]
~[“=“]
~[“=“]
~[“|“]
~[“&“]
-1
29: ASSIGN
30: EQ
31: LT
32: LE
33: GT
34: GE
35: NOT
36: NE
38: OR
40: AND
41: PLUS
42: MINUS
43: PROD
44: MOD
Comparadores y operadores lógicos [29-40 sin 37 ni 39]
Operadores Aritméticos [41-44]

5. El Compilador de Tinto

- Clase principal del compilador de *Tinto* en el paquete tinto y se denomina *TintoCompiler*. Esta clase contiene el método *main()* que lanza la aplicación.
- En esta práctica el funcionamiento del compilador se limita a explorar el directorio de trabajo, abrir el fichero *Main.tinto* y ejecutar el analizador sobre él.
- En caso de error, el compilador genera el archivo *TintocErrors.txt* con la descripción del error detectado-
- Si el análisis léxico es correcto se genera el archivo *TintocOutput.txt* con el listado de tokens generados.