Procesadores de Lenguajes

Memoria de proyecto - Hito 1: Analizador Léxico

Grupo 10
Sergio Colet García
Laura Martínez Tomás
Rodrigo Souto Santos
Li Jie Chen Chen

Grado en Ingeniería informática
Facultad de Informática
Universidad Complutense de Madrid

2023-2024

Índice general

1 | Tiny(0)

1.1. Introducción

Para realizar este apartado nos hemos fijado en todas las funcionalidades que aparecen en el "Apendice A" que aparecen en el archivo "fase1.pdf". En los siguientes apartados definimos todas las clases que hay, su correspondiente especificación y un diagrama de transiciones.

1.2. Clases léxicas

1.2.1. Palabras reservadas

Para poder analizar de manera correcta, será necesario establecer una clase léxica por cada palabra reservada. En el lenguaje de esta práctica, *Tiny(0)*, contamos con 6 palabras reservadas, 3 de ellas utilizadas para definir el tipo de las variables. Tendremos pues, una palabra para las variables de tipo booleano, otra para las de tipo entero y una última para las reales. Además de éstas tendremos 3 palabras utilizadas para las operaciones lógicas. Las palabras son las definidas a continuación, contando cada con una clase léxica.

- *bool* → Variables booleanas.
- *int* → Variables enteras.
- *real* → Variables reales.
- *and* → Conjunción lógica.
- *or* → Disyunción lógica.
- *not* → Negación lógica.

Además de éstas, hay palabras reservadas como *true* o *false*, las cuáles no poseen una clase léxica propia porque al detectarse se asignarán como literales booleanos (cómo veremos más adelante).

1.2.2. Literales

- **Literales booleanos.** Toma como valor las palabras reservadas *true* o *false*. Su clase léxica será *literalBooleano*.
- **Literales enteros.** Opcionalmente empiezan con un signo más (+) o menos (-), y después debe aparecer una secuencia (que empieza por un número distinto de 0) de 1 o más dígitos. Su clase léxica será *literalEntero*.
- **Literales reales.**Empieza con una parte entera seguida de una parte decimal, exponecial o parte decimal seguida de exponecial. La parte decimal comienza con el signo punto (.) seguido de una secuencia (que puede ser sólo un 0 o números que no acaben en 0) de 1 o más dígitos. Por último, y también opcionalmente, puede aparecer una parte exponencial que se indica con (e) o (E), seguida de una parte entera con o sin parte decimal. Su clase léxica será *literalReal*.

1.2.3. Identificadores

Los identificadores nos sirven para poder ponerle un nombre a las variables. Éstos deben comenzar por un subrayado (_) o una letra, seguida de una secuencia de 0 o más subrayados, dígitos o letras. Su clase léxica será *identificador*.

1.2.4. Símbolos de operación y puntuación

Cada uno de ellos tendrá su propia clase léxica. En el subconjunto del lenguaje en el que trabajamos, *Tiny(0)*, contamos con las siguientes clases:

- **Suma.** Se representa con el símbolo más (+). Su clase léxica será *operadorSuma*.
- **Resta.** Se representa con el símbolo símbolo menos (-). Su clase léxica será *operadorResta*.
- **Multiplicación.** Se representa con el símbolo asterisco (*). Su clase léxica será *operadorMul*.
- **División.** Se representa con el símbolo barra (/). Su clase léxica será *operadorDiv*.
- **Menor.** Se representa con el símbolo menor qué (<). Su clase léxica será *operadorMenor*.
- **Mayor.** Se representa con el símbolo mayor qué (>). Su clase léxica será *operadorMayor*.
- **Igual.** Se representa con el dos símbolos de igualdad seguidos (==). Su clase léxica será *operadorIgual*.
- **Menor o igual.** Se representa con el símbolo menor qué seguido del símbolo de igualdad (<=). Su clase léxica será *operadorMenIgual*.
- **Mayor o igual.** Se representa con el símbolo mayor qué seguido del símbolo de igualdad (>=). Su clase léxica será *operadorMayIgual*.
- **Asignación.** Se representa con el símbolo un símbolo de igualdad (=). Su clase léxica será *operadorAsig*.
- **Paréntesis de apertura.** Se representa con el símbolo del paréntesis de apertura ("(", sin comillas). Su clase léxica será *parentesisAp*.
- **Paréntesis de cierre.** Se representa con el símbolo del paréntesis de cierre (")", sin comillas). Su clase léxica será *parentesisCi*.
- **Punto y coma.** Se representa con el símbolo punto y coma (;). Su clase léxica será *puntoYComa*.
- **Coma.** Se representa con el símbolo coma (,). Su clase léxica será *coma*.

1.3. Especificación formal del léxico

1.3.1. Definiciones auxiliares.

$letra \longrightarrow \mathbf{A}|\mathbf{B}|...|\mathbf{Z}|\mathbf{a}|\mathbf{b}|...|\mathbf{z}$
$digitoPositivo \longrightarrow \mathbf{1}|...|\mathbf{9}$
$digito \longrightarrow digitoPositivo|0$
$parteEntera \longrightarrow digitoPositivo\ digito*$
$parteDecimal \longrightarrow (digito*\ digitoPositivo)|0$
$parteExponencial \longrightarrow (e|E)[\backslash+|-]parteEntera$

1.3.2. Definiciones de cadenas ignorables.

$separador \longrightarrow \mathbf{SP}|\mathbf{TAB}|\mathbf{NL}$
$comentario \longrightarrow \#\#(\overline{\mathbf{NL}|\mathbf{EOF}})$

1.3.3. Definiciones léxicas.

$bool \longrightarrow \mathbf{bool}$
$int \longrightarrow \mathbf{int}$
$real \longrightarrow \mathbf{real}$
$and \longrightarrow \mathbf{and}$
$or \longrightarrow \mathbf{or}$
$not \longrightarrow \mathbf{not}$
$literalBooleano \longrightarrow \mathbf{true}|\mathbf{false}$

$literalEntero \longrightarrow [\backslash+|-]parteEntera$
$literalReal \longrightarrow [\backslash+|-]parteEntera(.parteDecimal|parteExponencial|.parteDecimalparteExponencial)$
$identificador \longrightarrow (\_|letra)(letra|digito|\_)*$
$operadorSuma \longrightarrow \backslash+$
$operadorResta \longrightarrow -$
$operadorMul \longrightarrow \backslash*$
$operadorDiv \longrightarrow /$
$operadorMenor \longrightarrow <$
$operadorMayor \longrightarrow >$
$operadorIgual \longrightarrow ==$
$operadorMenIgual \longrightarrow <=$
$operadorMayIgual \longrightarrow >=$
$operadorAsig \longrightarrow =$
$parentesisAp \longrightarrow \backslash($
$parentesisCi \longrightarrow \backslash)$
$puntoYComa \longrightarrow ;$
$arroba \longrightarrow @$

1.4. Diseño de un analizador léxico

Se ha diseñado el analizador léxico del lenguaje mediante un diagrama de transiciones, como se observa en la figura **??**. Éste ha sido realizado usando la herramienta *JFLAP*. Hemos incluido todos los posibles síbolos que pueden haber en el subconjunto *Tiny(0)*, contando finalmente con un total de 33 estados.

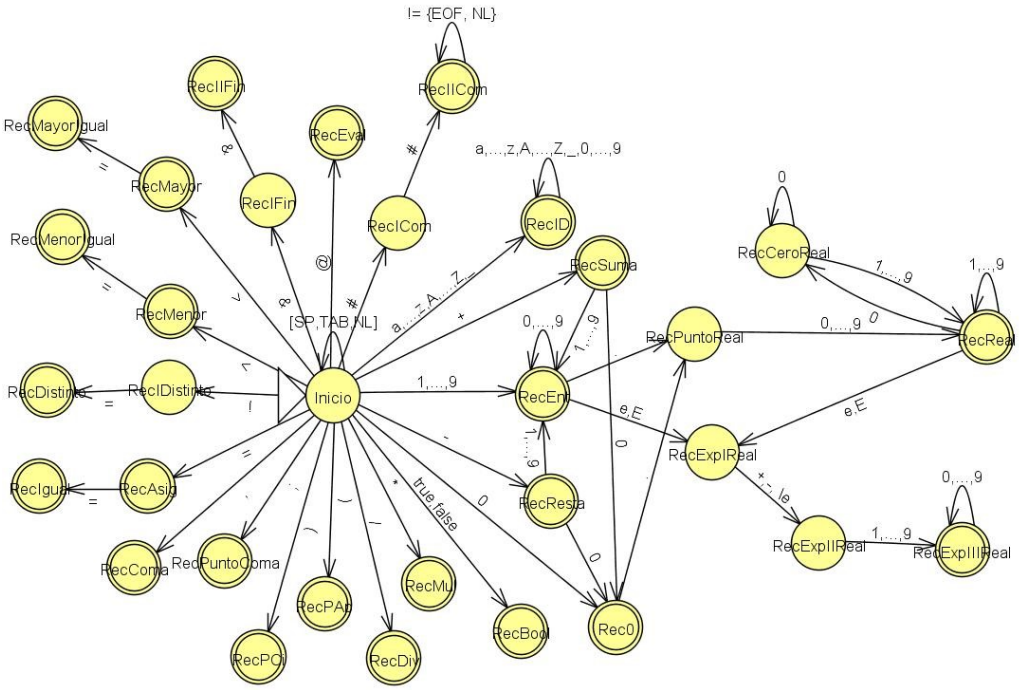

Figura 1.4.1: AFD para Tiny(0)

2 | Tiny

2.1. Introducción

Para realizar este apartado nos hemos basado en todas las funcionalidades que aparecen en el archivo "lenguaje.pdf" que se ha aportado en el campus. En los siguientes apartados definimos todas las clases que hay y su correspondiente especificación.

2.2. Clases léxicas

2.2.1. Palabras reservadas

Para poder analizar de manera correcta, será necesario establecer una clase léxica por cada palabra reservada. En el lenguaje de esta práctica, *Tiny(0)*, contamos con 3 palabras reservadas, utilizadas para definir el tipo de las variables. Tendremos pues, una palabra para las variables de tipo booleano, otra para las de tipo entero y una última para las reales.También contamos con 3 palabras reservadas para los operadores lógicos and, or y not, 1 palabra reservada para hacer referencia a la nada, 1 palabra reservada para referenciar una función, 3 palabras reservadas para control de flujo, 1 palabra reservada para la creación de un estructura, 1 palabra reservada para reserva de memoria, 1 palabra reservada para liberar la memoria, 1 palabra reservada para lectura, 1 palabra reservada para escritura, 1 palabra reservada para nueva linea, 1 palabra reservada para vínculos de los nombres de tipo y 1 palabra reservada para invocación a procedimiento. Las palabras son las definidas a continuación, contando cada con una clase léxica.

- *bool* → Variables booleanas.
- *int* → Variables enteras.
- *real* → Variables reales.
- *string* → Variables de cadena.
- *and* → Conjunción lógica.
- *or* → Disyunción lógica.
- *not* → Negación lógica.
- *null* → Referencia a la nada.
- *proc* → Función.
- *if* → Condición.
- *else* → Condición alternativa.
- *while* → Bucle con condición.
- *struct* → Estructura.
- *new* → Reserva de memoria.
- *delete* → Liberación de memoria.
- *read* → Lectura.
- *write* → Escritura.
- *nl* → Nueva línea.
- *type* → Vinculo de tipo.
- *call* → Invocación procedimiento.

Además de éstas, hay palabras reservadas como *true* o *false*, las cuáles no poseen una clase léxica propia porque al detectarse se asignarán como literales booleanos (cómo veremos más adelante).

2.2.2. Literales

- **Literales booleanos.** Toma como valor las palabras reservadas *true* o *false*. Su clase léxica será *literalBooleano*.
- **Literales enteros.** Opcionalmente empiezan con un signo más (+) o menos (-), y después debe aparecer una secuencia (que empieza por un número distinto de 0) de 1 o más dígitos. Su clase léxica será *literalEntero*.
- **Literales reales.** Empieza con una parte entera seguida de una parte decimal, exponecial o parte decimal seguida de exponecial. La parte decimal comienza con el signo punto (.) seguido de una secuencia (que puede ser sólo un 0 o números que no acaben en 0) de 1 o más dígitos. Por último, y también opcionalmente, puede aparecer una parte exponencial que se indica con (e) o (E), seguida de una parte entera con o sin parte decimal. Su clase léxica será *literalReal*.
- **Literales de cadena.** Secuencia de 0 o más caracteres distintos que estan entre comillas dobles(“ ”). Los caracteres pueden incluir las siguientes secuencias de escape: retroceso ($\backslash b$), retorno de carro ($\backslash r$), tabulador ($\backslash t$) y salto de línea ($\backslash n$). Su clase léxica será *literalCadena*.

2.2.3. Identificadores

Los identificadores nos sirven para poder ponerle un nombre a las variables. Éstos deben comenzar por un subrayado (_) o una letra, seguida de una secuencia de 0 o más subrayados, dígitos o letras. Su clase léxica será *identificador*.

2.2.4. Símbolos de operación y puntuación

Cada uno de ellos tendrá su propia clase léxica y son las siguientes clases:

- **Suma.** Se representa con el símbolo más (+). Su clase léxica será *operadorSuma*.
- **Resta.** Se representa con el símbolo símbolo menos (-). Su clase léxica será *operadorResta*.
- **Multiplicación.** Se representa con el símbolo asterisco (*). Su clase léxica será *operadorMul*.
- **División.** Se representa con el símbolo barra (/). Su clase léxica será *operadorDiv*.
- **Módulo.** Se representa con el símbolo barra (%). Su clase léxica será *operadorMod*.
- **Menor.** Se representa con el símbolo menor qué (<). Su clase léxica será *operadorMenor*.
- **Mayor.** Se representa con el símbolo mayor qué (>). Su clase léxica será *operadorMayor*.
- **Igual.** Se representa con el dos símbolos de igualdad seguidos (==). Su clase léxica será *operadorIgual*.
- **Menor o igual.** Se representa con el símbolo menor qué seguido del símbolo de igualdad (<=). Su clase léxica será *operadorMenIgual*.
- **Mayor o igual.** Se representa con el símbolo mayor qué seguido del símbolo de igualdad (>=). Su clase léxica será *operadorMayIgual*.
- **Asignación.** Se representa con el símbolo un símbolo de igualdad (=). Su clase léxica será *operadorAsig*.
- **Paréntesis de apertura.** Se representa con el símbolo del paréntesis de apertura (“(”, sin comillas). Su clase léxica será *parentesisAp*.
- **Paréntesis de cierre.** Se representa con el símbolo del paréntesis de cierre (“)”, sin comillas). Su clase léxica será *parentesisCi*.
- **Punto y coma.** Se representa con el símbolo punto y coma (;). Su clase léxica será *puntoYComa*.
- **Coma.** Se representa con el símbolo coma (,). Su clase léxica será *coma*.
- **Indirección.** Se representa con el símbolo del acento circumflejo (^). Su clase léxica será *indireccion*.
- **Final.** Se representa con el símbolo ampersand 2 veces consecutivas (&&). Su clase léxica será *final*.
- **Por Referencia.** Se representa con el símbolo ampersand una única vez (&). Su clase léxica será *porReferencia*.

- **Llave de apertura.** Se representa con el símbolo del corchete de apertura ({). Su clase léxica será *llaveAp*.
- **Llave de cierre.** Se representa con el símbolo del corchete de cierre (}). Su clase léxica será *llaveCi*.
- **Corchete de apertura.** Se representa con el símbolo del corchete de apertura ([). Su clase léxica será *corcheteAp*.
- **Corchete de cierre.** Se representa con el símbolo del corchete de cierre (]). Su clase léxica será *corcheteCi*.
- **Arroba.** Se representa con el símbolo coma (@). Su clase léxica será *arroba*.

2.3. Especificación formal del léxico

2.3.1. Definiciones auxiliares.

$letra \longrightarrow \mathbf{A}|\mathbf{B}|...|\mathbf{Z}|\mathbf{a}|\mathbf{b}|...|\mathbf{z}$
$digitoPositivo \longrightarrow \mathbf{1}|...|\mathbf{9}$
$digito \longrightarrow digitoPositivo|0$
$parteEntera \longrightarrow digitoPositivo\ digito*$
$parteDecimal \longrightarrow (digito*\ digitoPositivo)|0$
$parteExponencial \longrightarrow (e|E)[\backslash+|-]parteEntera$

2.3.2. Definiciones de cadenas ignorables.

$separador \longrightarrow \mathbf{SP}|\mathbf{TAB}|\mathbf{NL}$
$comentario \longrightarrow \#\#(\overline{\mathbf{NL}|\mathbf{EOF}})$

2.3.3. Definiciones léxicas.

$bool \longrightarrow \mathbf{bool}$
$int \longrightarrow \mathbf{int}$
$real \longrightarrow \mathbf{real}$
$string \longrightarrow \mathbf{string}$
$and \longrightarrow \mathbf{and}$
$or \longrightarrow \mathbf{or}$
$not \longrightarrow \mathbf{not}$
$null \longrightarrow \mathbf{null}$
$proc \longrightarrow \mathbf{proc}$
$if \longrightarrow \mathbf{if}$
$else \longrightarrow \mathbf{else}$
$while \longrightarrow \mathbf{while}$
$struct \longrightarrow \mathbf{struct}$
$new \longrightarrow \mathbf{new}$
$delete \longrightarrow \mathbf{delete}$
$read \longrightarrow \mathbf{read}$
$write \longrightarrow \mathbf{write}$
$nl \longrightarrow \mathbf{nl}$
$type \longrightarrow \mathbf{type}$
$call \longrightarrow \mathbf{call}$
$literalBooleano \longrightarrow \mathbf{true}|\mathbf{false}$
$literalEntero \longrightarrow [\backslash+|-]parteEntera$
$literalReal \longrightarrow [\backslash+|-]parteEntera(.parteDecimal|parteExponencial|.parteDecimalparteExponencial)$
$literalCadena$[1] $\longrightarrow$ “ $(\alpha|separador)*$ ”
$identificador \longrightarrow (\_|letra)(letra|digito|\_)*$
$operadorSuma \longrightarrow \backslash+$

[1]Definimos α como cualquier caracter perteneciente al código ASCII.

$operadorResta \longrightarrow -$
$operadorMul \longrightarrow \backslash*$
$operadorDiv \longrightarrow /$
$operadorMod \longrightarrow \%$
$operadorMenor \longrightarrow <$
$operadorMayor \longrightarrow >$
$operadorIgual \longrightarrow ==$
$operadorMenIgual \longrightarrow <=$
$operadorMayIgual \longrightarrow >=$
$operadorAsig \longrightarrow =$
$parentesisAp \longrightarrow \backslash($
$parentesisCi \longrightarrow \backslash)$
$puntoYComa \longrightarrow ;$
$arroba \longrightarrow @$
$coma \longrightarrow ,$
$indireccion \longrightarrow \backslash\hat{}$
$final \longrightarrow \&\&$
$porReferencia \longrightarrow \&$
$llaveAp \longrightarrow \{$
$llaveCi \longrightarrow \}$
$corcheteAp \longrightarrow [$
$corcheteCi \longrightarrow]$
$arroba \longrightarrow @$

Índice de figuras