Memoria: Procesadores de Lenguaje - Lenguaje Tiny

Burgos Sosa, Rodrigo
Cassin, Gina Andrea
Estebán Velasco, Luis
Rabbia, Santiago Elias

Curso 2024

Grupo G03

1 Introducción

En el siguiente documento se expondrá una memoria sobre el desarrollo de analizadores léxicos aplicado sobre dos lenguajes de programación, Tiny y Tiny(0) - un subconjunto de Tiny. Se presentará una descripción de las clases léxicas y una especificación formal de ambos lenguajes, y un analizador léxico para Tiny(0).

2 Análisis léxico

La función de un analizador léxico es segmentar el programa de entrada en una secuencia de componentes léxicos o tokens. La primera fase en el desarrollo del analizador léxico (y la fase más importante) es llevar a cabo su especificación léxica. Esto se llevará a cabo a continuación:

2.1 Tiny(0)

2.1.1 Clases léxicas

A continuación se presentan las clases léxicas del lenguaje Tiny(0):

Clases léxicas

- **Identificador (variable)**: Comienzan necesariamente por una letra o subrayado (_), seguida de una secuencia de cero o más letras, dígitos, o subrayado (_).
- **Una clase léxica por cada tipo de variable**:
  - **int**: representa los números enteros.
  - **real**: representa los números reales.
  - **bool**: representa los valores booleanos (true o false).
- **Literal entero**
- **Literal real**
- **Literal booleano**
- **Una clase léxica por cada operador aritmético**:
  - +: suma.

  - **-**: resta.
  - *****: multiplicación.
  - /: división.

- **Una clase léxica por cada operador lógico**:
  - **and**: conjunción.
  - **or**: disyunción.
  - **not**: negación.

- **Una clase léxica por cada operador relacional**:
  - **<**: menor que.
  - **>**: mayor que.
  - **<=**: menor o igual que.
  - **>=**: mayor o igual que.
  - **==**: igual que.
  - **!=**: distinto que.

- **Una clase léxica por cada símbolo de puntuación**:
  - **(**: paréntesis izquierdo. Sirve para asociatividad.
  - **)**: paréntesis derecho. Sirve para asociatividad.
  - **;**: punto y coma. Sirve para separar declaraciones en la seccion de declaraciones, o separar instrucciones en la seccion de instrucciones.
  - **.**: punto. Para los decimales.
  - **{**: llave izquierda. Indica el inicio de un bloque de código.
  - **}**: llave derecha. Indica el fin de un bloque de código.
  - &&: doble signo et. Indica el fin de declaraciones.

- **Operador de asignación**: =
- **Operador de evaluación**: @

Cadenas ignorables

- **Espacios en blanco**.
- **Retroceso**: \b
- **Tabulador**: \t
- **Retorno de carro**: \r
- **Salto de línea**: \n
- **Comentarios**: comienzan con ## y terminan con un salto de línea.

2.1.2 Especificación formal

Definiciones auxiliares

- **letra** ≡ [a-z,A-Z]
- **digito** ≡ [0-9]
- **digitoSinCero** ≡ [1-9]
- **parteEntera** ≡ ({digitoSinCero} {digito}*) | 0
- **parteDecimal** ≡ ({digito}* {digitoSinCero}) | 0

Definiciones léxicas

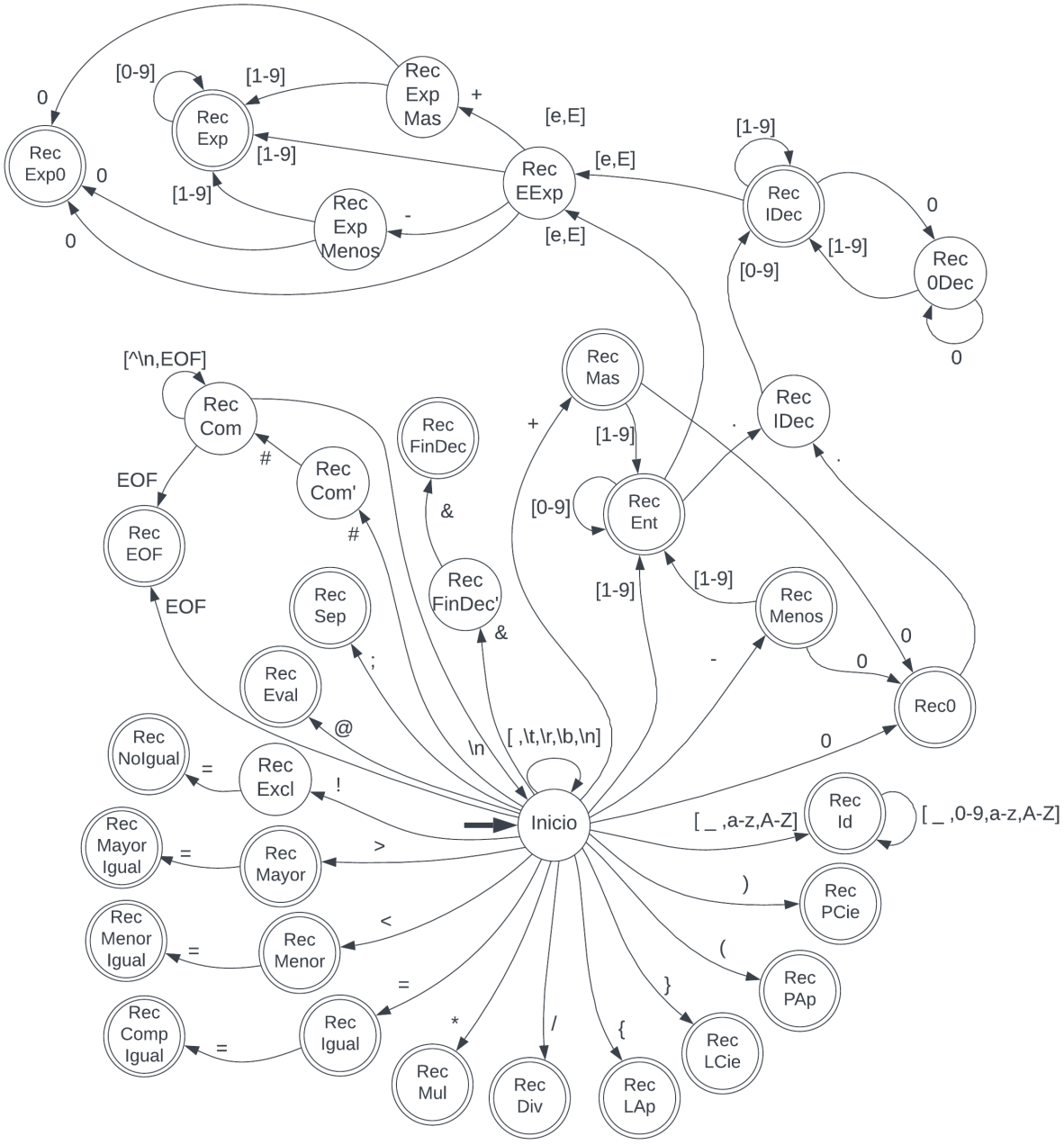
- **suma** ≡ \+
- **resta** ≡ \-
- **mul** ≡ *
- **div** ≡ /
- **parentesisAbrir** ≡ \(
- **parentesisCerrar** ≡ \)
- **abrirBloque** ≡ \{
- **cerrarBloque** ≡ \}
- **separadorDeclaraciones** ≡ ;

- **finDeclaraciones** ≡ &&
- **asignacion** ≡ \=
- **menor** ≡ <
- **mayor** ≡ >
- **menorIgual** ≡ <\=
- **mayorIgual** ≡ >\=
- **igual** ≡ \=\=
- **no igual** ≡ !\=
- **and** ≡ and
- **or** ≡ or
- **not** ≡ not
- **true** ≡ true
- **false** ≡ false
- **tipo entero** ≡ int
- **tipo real** ≡ real
- **tipo booleano** ≡ bool
- **eval** ≡ @
- **punto** ≡ \.
- **identificador** ≡ ({letra} | _) ({letra} | {dígito} | _)*
- **literalEntero** ≡ (\+ | \-)? {parteEntera}
- **literalReal** ≡ {literalEntero}((\.{parteDecimal})((e | E){literalEntero})) | (\.{parteDecimal}) | ((e | E){literalEntero})

Definiciones cadenas ignorables

- **separador** ≡ [,\t,\r,\b,\n]
- **comentario** ≡ ##([^\n, EOF])*

2.1.3 Diagrama de transiciones



2.2 Tiny

2.2.1 Clases léxicas

A continuación se presentan las clases léxicas del lenguaje Tiny:

Clases léxicas

- **Identificador (variable)**: Comienzan necesariamente por una letra o subrayado (_), seguida de una secuencia de cero o más letras, dígitos, o subrayado (_).
- **Una clase léxica por cada tipo de variable**:
  - **int**: representa los números enteros.
  - **real**: representa los números reales.
  - **bool**: representa los valores booleanos (true o false).
  - **string**: representa las cadenas de caracteres.
  - **array**: representa los arreglos.
- **Literal entero**
- **Literal real**
- **Literal booleano**
- **Literal cadena**
- **Una clase léxica por cada operador aritmético**:
  - +: suma.
  - -: resta.
  - *: multiplicación.
  - /: división.
- **Una clase léxica por cada operador lógico**:
  - **and**: conjunción.
  - **or**: disyunción.
  - **not**: negación.
- **Una clase léxica por cada operador relacional**:

- <: menor que.
- >: mayor que.
- <=: menor o igual que.
- >=: mayor o igual que.
- ==: igual que.
- **!**=: distinto que.

- **Una clase léxica por cada símbolo de puntuación**:
  - (: paréntesis izquierdo. Sirve para asociatividad. También indica el inicio de una lista de parámetros cuando se definen procedimientos y el inicio de argumentos cuando se llama una función.
  -): paréntesis derecho. Sirve para asociatividad. También indica el fin de una lista de parámetros cuando se definen procedimientos y el fin de argumentos cuando se llama una función.
  - **;**: punto y coma. Sirve para separar declaraciones en la sección de declaraciones, o separar instrucciones en la sección de instrucciones.
  - **,**: coma. Separa los campos dentro de la definición de un struct, los parámetros en la definición de un procedimiento, y los argumentos en la llamada a una función.
  - **.**: punto. Para los decimales, y es un operador de "acceso a registro".
  - **{**: llave izquierda. Indica el inicio de un bloque de código. También indica el inicio de la definición de un struct.
  - **}**: llave derecha. Indica el fin de un bloque de código. También indica el fin de la definición de un struct.
  - &: signo et simple. Indica que un parámetro de un procedimiento se pasa por referencia.
  - &&: doble signo et. Indica el fin de declaraciones.
  - [: corchete izquierdo. Operador de indexación.
  -]: corchete derecho. Operador de indexación.
  - %: operador módulo.
  - ^: acento circunflejo. Se usa para definir un puntero. También es el operador de indirección.

- **Operador de asignación**: =
- **Operador de evaluación**: @
- **Una clase léxica por cada palabra reservada**:
  - **null**: representa el valor nulo.
  - **proc**: palabra reservada para definir un procedimiento.
  - **if**: palabra reservada para definir una condición.
  - **else**: palabra reservada para definir una condición alternativa.
  - **while**: palabra reservada para definir un bucle.
  - **struct**: palabra reservada para definir una estructura.
  - **new**: palabra reservada para instrucción de reserva de memoria.
  - **delete**: palabra reservada para instrucción de liberación de memoria.
  - **read**: palabra reservada para instrucción de lectura.
  - **write**: palabra reservada para instrucción de escritura.
  - **nl**: palabra reservada para instrucción de nueva linea.
  - **type**: palabra reservada para declaración de tipo.
  - **call**: palabra reservada para instrucción de invocación a procedimiento.

Cadenas ignorables

- **Espacios en blanco**.
- **Retroceso**: \b
- **Tabulador**: \t
- **Retorno de carro**: \r
- **Salto de línea**: \n
- **Comentarios**: comienzan con ## y terminan con un salto de línea.

2.2.2 Especificación formal

Definiciones auxiliares

- **letra** ≡ [a-z,A-Z]
- **digito** ≡ [0-9]
- **digitoSinCero** ≡ [1-9]
- **parteEntera** ≡ ({digitoSinCero} {digito}*) | 0
- **parteDecimal** ≡ ({digito}* {digitoSinCero}) | 0

Definiciones léxicas

- **suma** ≡ \+
- **resta** ≡ \-
- **mul** ≡ *
- **div** ≡ /
- **parentesisAbrir** ≡ \(
- **parentesisCerrar** ≡ \)
- **abrirBloque** ≡ \{
- **cerrarBloque** ≡ \}
- **tamañoAbrir** ≡ [
- **tamañoCerrar** ≡]
- **finDeclaraciones** ≡ &&
- **asignacion** ≡ \=
- **menor** ≡ <
- **mayor** ≡ >
- **menorIgual** ≡ <\=
- **mayorIgual** ≡ >\=

- **igual** ≡ \=\=
- **no igual** ≡ !\=
- **and** ≡ and
- **or** ≡ or
- **not** ≡ not
- **true** ≡ true
- **false** ≡ false
- **modulo** ≡ %
- **puntero** ≡ ^
- **bitwiseAnd** ≡ &
- **tipo entero** ≡ int
- **tipo real** ≡ real
- **tipo booleano** ≡ bool
- **tipo string** ≡ string
- **tipo array** ≡ array
- **creacionTipo** ≡ type
- **null** ≡ null
- **procedimiento** ≡ proc
- **if** ≡ if
- **else** ≡ else
- **while** ≡ while
- **estructura** ≡ struct
- **new** ≡ new
- **delete** ≡ delete

- **read** ≡ read
- **write** ≡ write
- **nl** ≡ nl
- **call** ≡ call
- **eval** ≡ @
- **punto** ≡ \.
- **identificador** ≡ ({letra} | _) ({letra} | {dígito} | _)*
- **literalEntero** ≡ (\+ | \-)? {parteEntera}
- **literalReal** ≡ {literalEntero}((\.{parteDecimal})((e | E){literalEntero})) | (\.{parteDecimal}) | ((e | E){literalEntero})

Definiciones cadenas ignorables

- **separador** ≡ [,\t,\r,\b,\n]
- **comentario** ≡ ##([^\n, EOF])*