Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información
CI3725 – Traductores e Interpretadores
Enero – Marzo 2016

Etapa I
Análisis Lexicográfico

Esta primera etapa del proyecto corresponde al módulo de análisis lexicográfico del interpretador del lenguaje `BOT` que queremos construir. Específicamente, se desea que Ud. implemente el referido módulo utilizando una herramienta generadora de analizadores lexicográficos de las permitidas (mencionadas más adelante), y que además haga una breve revisión de los conceptos y métodos teórico-prácticos relevantes.

El analizador lexicográfico a ser construido en esta etapa deberá aceptar como entrada cualquier secuencia de caracteres y dar como salida los *tokens* relevantes reconocidos. Si se recibe caracteres que no corresponden a ningún *token* del lenguaje `BOT`, se debe dar un mensaje de error. Cada token debe ir acompañado de su número de fila y columna.

Los *tokens* relevantes serán:

- Cada una de las palabras claves utilizadas en la sintaxis de `BOT`, i.e. `create`, `while`, `bool`, `if`, etc. En este caso, los tokens deberán llamarse *"Tk⟨Palabra Clave⟩"*, donde ⟨*Palabra Clave*⟩ es la palabra clave a la que representa el token, con su primera letra en mayúscula. Por ejemplo, para la palabra clave `create`, su token sería `TkCreate`.

- Los identificadores de variables. A diferencia de las palabras claves, los identificadores corresponderán a un único *token* llamado `TkIdent`. Este *token* siempre tendrá asociado como atributo el identificador particular reconocido. Por ejemplo, al leer `contador`, se dará como salida `TkIdent("contador")`.

- Los literales numéricos, los cuales serán secuencias no-vacías de dígitos decimales. De manera análoga a los identificadores, éstos serán agrupados bajo el *token* `TkNum`. Este *token* tendrá como atributo el número reconocido, por ejemplo `TkNum(3000)`.

- Los literales booleanos, los cuales a diferencia de los literales numéricos, estarán representados por dos *tokens*. Uno de ellos para `true` (`TkTrue`) y otro para `false` (`TkFalse`).

- Los literales para caracteres, los cuales estarán envueltos en comillas simples. De manera análoga a los identificadores y los literales numéricos, éstos serán agrupados bajo el *token* `TkCaracter`. Este *token* tendrá como atributo el caracter reconocido, por ejemplo `TkCaracter('p')`.

- Cada uno de los símbolos que denotan separadores en `BOT`, los cuales se presentan a continuación:

  `","` – `TkComa`
  `"."` – `TkPunto`
  `":"` – `TkDosPuntos`
  `"("` – `TkParAbre`
  `")"` – `TkParCierra`

- Cada uno de los símbolos que denotan a operadores aritméticos, booleanos, relacionales, o de otro tipo en `BOT`, los cuales se presentan a continuación:

```
 "+"  —  TkSuma
 "-"  —  TkResta
 "*"  —  TkMult
 "/"  —  TkDiv
 "%"  —  TkMod
"/\"  —  TkConjuncion
"\/"  —  TkDisyuncion
 "~"  —  TkNegacion
 "<"  —  TkMenor
"<="  —  TkMenorIgual
 ">"  —  TkMayor
">="  —  TkMayorIgual
 "="  —  TkIgual
```

Los espacios en blanco, tabuladores y saltos de línea deben ser ignorados. Así mismo, los comentarios no deben ser reconocidos como *tokens*, sino ser ignorados igualmente. Cualquier otro caracter será reportado como error.

A continuación algunos ejemplos, donde primeramente se muestra la secuencia de caracteres de entrada que recibe el analizador lexicográfico y luego la secuencia de *tokens* de salida, todos acompañados de dos números (el primero de ellos la fila en donde se encuentra el token y el segundo la columna):

```
create
    int bot contador
        on activation:
            store 35.
        end
    end

execute
    $- Asignar al contador
    el valor 35 -$
    activate contador.
end

TkCreate 1 1, TkInt 2 5, TkBot 2 9, TkIdent("contador") 2 13,
TkOn 3 9, TkActivation 3 12, TkDosPuntos 3 22, TkStore 4 13,
TkNum(35) 4 19, TkPunto 4 21, TkEnd 5 9, TkEnd 6 5, TkExecute 8 1,
TkActivate 11 5, TkIdent("contador") 11 14, TkPunto 11 22, TkEnd 12 1
```

Como puede verse, la región del programa que está encerrada entre los símbolos "`$-`" y "`-$`" (y que por ende son comentarios en `BOT`), fueron ignorados completamente al presentar la salida.

Nótese que el analizador lexicográfico sólo puede ser capaz de reconocer secuencias <u>arbitrarias</u> de *tokens*, aunque tales secuencias sean sintácticamente incorrectas. Por ejemplo:

```
bot execute
    int 35
        activation: on
```

```
            store contador.
    end

end
    $- Asignar al 35
    el valor contador -$
    contador end.
activate create

TkBot 1 1, TkExecute 1 5, TkInt 2 5, TkNum(35) 2 9, TkActivation 3 9,
TkDosPuntos 3 19, TkOn 3 21, TkStore 4 13, TkIdent("contador") 4 19,
TkPunto 4 27, TkEnd 5 5, TkEnd 7 1, TkIdent("contador") 10 5,
TkEnd 10 14, TkPunto 10 17, TkActivate 11 1, TkCreate 11 10
```

Los *únicos* errores detectables a nivel lexicográfico corresponden a caracteres irrelevantes. Por ejemplo:

```
create
    int bot contador!
        on activation:
            store 35.
        end
    end

execute
    $- Asignar al contador
    el valor 35 -$
    activate? contador.
end

Error: Caracter inesperado "!" en la fila 2, columna 21
Error: Caracter inesperado "?" en la fila 11, columna 13
```

Su analizador lexicográfico debe reportar *todos* los errores léxicos, en caso de haberlos. Cuando algún error es encontrado, los *tokens* se hacen irrelevantes (ya que no corresponde a un programa correcto), por lo que no deberán ser mostrados. Los errores deben mostrarse con el mismo formato en que se mostró en el ejemplo anterior.

Es importante notar que cada *token* producido por su analizador lexicográfico debe corresponder a un tipo de datos y no simplemente ser impreso a la salida estándar. Esto a modo de poder suministrarlos luego como información para el analizador sintáctico (a implementar en la segunda parte de este proyecto). De esta forma, usted deberá implementar un programa principal que invoque al analizador lexicográfico y que sea luego el que imprima a la salida estándar la representación como texto de los *tokens* encontrados.

Tal como se indicó inicialmente, esta primera etapa consta tanto de la implementación del analizador lexicográfico utilizando alguno de los lenguajes y herramientas permitidos, como de una breve revisión teórico-práctica. A continuación se explica cuáles son los lenguajes y herramientas permitidas, los detalles de la entrega de la implementación y finalmente, en qué consiste la revisión teórico-práctica.

Lenguajes y Herramientas Permitidos Para la implementación de este proyecto se cuenta para escoger con cuatro (4) lenguajes de programación. Estos son:

- *Python:* Lenguaje de `scripting`, orientado a objetos. Es posible conseguir *Python* desde la siguiente dirección Web:

  (http://www.python.org/download/).

  Para *Python* la herramienta generadora de analizadores lexicográficos a utilizar es a la vez la misma que genera analizadores sintácticos. Por lo tanto, deberán manejarse algunas nociones de gramáticas libres de contexto antes de tiempo para poder trabajar con la misma. Esta herramienta se llama *PLY* y puede ser encontrada desde la siguiente dirección Web:

  (http://www.dabeaz.com/ply/).

- *Haskell:* Lenguaje funcional puro. Es posible conseguir *Haskell* desde la siguiente dirección Web:

  (http://www.haskell.org/ghc/download.html/).

  Si decide utilizar *Haskell*, su implementación deberá ser compatible con el compilador proporcionado (*GHC*).

  Para *Haskell* la herramienta generadora de analizadores lexicográficos a utilizar se llama *Alex* y puede ser encontrada en la siguiente dirección Web:

  (http://www.haskell.org/alex/).

- *Ruby:* Lenguaje de `scripting`, orientado a objetos. Es posible conseguir *Ruby* desde la siguiente dirección Web:

  (http://www.ruby-lang.org/en/downloads/).

  En el caso particular de *Ruby* no hay una buena herramienta generadora de analizadores lexicográficos, por lo que el trabajo deberá hacerse manualmente a través de las expresiones regulares que provee el lenguaje.

- *Java:* Lenguaje imperativo, orientado a objetos. Es recomendable utilizar versiones de *Java* iguales o posteriores a la 1.5, dependiendo también de los requisitos de las herramientas a utilizar. Es posible conseguir *Java* desde la siguiente dirección Web:

  (http://www.java.com/es/download/).

  Para *Java*, la herramienta generadora de analizadores lexicográficos a utilizar se llama *JFlex* y puede ser encontrada en la siguiente dirección Web:

  (http://jflex.de/download.html/).

Entrega de la Implementación Ud. debe entregar:

- Un correo electrónico con el código fuente en el lenguaje y la herramienta de su elección, entre los permitidos, de su analizador lexicográfico. Todo el código debe estar debidamente documentado. El analizador deberá ser ejecutado con el comando "`./LexBot` ⟨*Archivo*⟩", por lo que es posible que tenga que incorporar un script a su entrega que permita que la llamada a su programa se realice de esta forma. ⟨*Archivo*⟩ tendrá el programa escrito en `BOT` que se analizará lexicográficamente. Este archivo tendrá extensión "`.bot`", sin embargo no tiene que verificar esto. Deben abstenerse de imprimir nada más que lo pedido y tener cuidado con que la salida del programa corresponda con fidelidad a los requisitos mencionados anteriormente.

- Un *breve* informe explicando la formulación/implementación de su analizador lexicográfico y justificando todo aquello que Ud. considere necesario. Además el informe debe contener sus respuestas a la revisión teórico–práctica, la cual será explicada más adelante.

Nota: Es importante que su código pueda ejecutarse en las máquinas del LDC, pues es ahí y únicamente ahí donde se realizará su corrección.

Revisión Teórico-Práctica Al informe de la implementación, Ud. deberá agregar el desarrollo de las preguntas que se listan en esta sección.

Para el desarrollo de estas preguntas, utilice el algoritmo de la Sección 7.4.2 (página 252) del libro [WM95]. Nos referiremos a este algoritmo con el nombre A.

1. Dé tres expresiones regulares E_1, E_2 y E_3 que correspondan respectivamente al reconocimiento de la palabra clave `create`, de la palabra clave `char` y de identificadores.

2. Dé los diagramas de transición (i.e. representación gráfica) de tres autómatas finitos (posiblemente no-determinísticos) M_1, M_2 y M_3 que reconozcan respectivamente a los lenguajes denotados por E_1, E_2 y E_3. Esto corresponde al paso (1) del algoritmo A, con R'_i y R_i refiriéndose a nuestras E_i. (*Nota:* La relación entre las expresiones R'_i y R_i está especificada en la Sección 7.3.2 (páginas 249-250) del mismo libro [WM95]; sin embargo, ésta no es relevante para nuestros propósitos.)

3. Construya un autómata finito no-determinístico M que reconozca a la unión de los lenguajes $L(M_1)$, $L(M_2)$ y $L(M_3)$, tal como se indica en el paso (2) de A.

4. Note que, a efectos de implementar un analizador lexicográfico, es importante que el autómata M sepa reportar a cuál de los tres lenguajes pertenece cada palabra que él reconozca. Esto significa que M debe poder identificar a cuál de los tres lenguajes corresponde cada estado final. Indique a qué lenguaje corresponde cada uno de los estados finales de su autómata M.

5. La asignación de estados finales a lenguajes de su respuesta a la pregunta 4 debe crear conflictos, pues hay elementos que pertenecen a más de uno de los tres lenguajes $L(M_1)$, $L(M_2)$ y $L(M_3)$. Cada conflicto corresponde a una palabra w que pertenece a más de un lenguaje, digamos L_x y L_y, la cual tendrá, por lo tanto, más de una "vía de reconocimiento" en M. Estas vías deben terminar en estados finales asociados a los distintos lenguajes correspondientes, i.e. L_x y L_y. Indique cuáles son los conflictos de su autómata M, especificando las palabras que los generan, y los lenguajes y estados finales involucrados.

6. Construya un autómata finito determinístico M_{det} equivalente a M, i.e. que reconozca el mismo lenguaje que M (viz. $L(M_1) \cup L(M_2) \cup L(M_3)$). Esto corresponde al paso (3) de A. (La referencia no es obligatoria, sino una guía para seguir el flujo de las preguntas a continuación).

7. ¿Cómo se reflejan los conflictos de su respuesta a la pregunta 5 en su autómata M_{det}?

8. Los conflictos deben ser resueltos mediante un orden lineal que priorice a los lenguajes involucrados. En nuestro caso, establecemos que el orden de prioridad viene dado por la secuencia $\langle L(E_1), L(E_2), L(E_3) \rangle$ de manera decreciente, i.e. el primer lenguaje tiene más prioridad que los otros dos y el segundo más que el tercero. De acuerdo a esto, asocie un lenguaje (solo uno) a cada estado final de M_{det} tal como lo hizo en la pregunta 4 para M.

9. Tal como se indica en el paso (4) de A, construya un autómata finito determinístico mínimo M_{min} equivalente a M (y, por tanto, también equivalente a M_{det}). Explique por qué se debe usar la partición inicial de estados especificada en el paso (4) de A, e indique a cuál de los tres lenguajes corresponde cada estado final de M_{min}.

10. ¿Cómo relaciona Ud. el desarrollo de las preguntas 1–9 a la implementación de su analizador lexicográfico construido con la herramienta escogida?

Nota Importante: El algoritmo A (mencionado en las preguntas anteriores) difiere en algunos puntos con la forma en que se manejan los conceptos en el libro principal utilizado en el curso [S97]. Por ejemplo, las transiciones de los autómatas no-determinísticos son vistos como relaciones, mientras que en clase se verán como funciones que devuelven conjuntos. Sin embargo, teniendo en cuenta estos detalles, es posible transportar la información que se presenta en el algoritmo A para que sea consistente con lo acostumbrado en clase.

Referencia Bibliográfica

[WM95] R. Wilhelm & D. Maurer. *Compiler Design.* Addison-Wesley, 1995.

[S97] T. Sudkamp. *Languages and Machines.* Second Edition. Addison-Wesley, 1997.

Fecha de Entrega: Viernes 22 de Enero (Semana 5), hasta las 11:59 pm.
Valor: 5%.

S. Ancelmo, C. Ferreira y R. Monascal / Septiembre 2015