Teoría de Lenguajes:
Práctica 8 – Parsers Descendentes

1er. cuatrimestre 2017

Parsers recursivos descendentes predictivos

1. Dadas las siguientes gramáticas, todas con $S = A$, construir los parsers recursivos descendentes predictivos correspondientes. Para cada caso, indicar si podría generarse algún conflicto o problema, y por qué. En caso afirmativo, ¿habría forma de solucionarlo sin modificar el lenguaje generado?

   (a) $P = \{\ A \longrightarrow A\texttt{\{}A\texttt{\}}A \mid \lambda\ \},\ V_T = \{\texttt{\{}, \texttt{\}}\},\ V_N = \{A\}$

   (b) $P = \{\ A \longrightarrow \texttt{+}AA \mid \texttt{-}AA \mid \texttt{a}\ \},\ V_T = \{\texttt{+}, \texttt{-}, \texttt{a}\},\ V_N = \{A\}$

   (c) $P = \{\ A \longrightarrow \texttt{0}A\texttt{1} \mid \texttt{01}\ \},\ V_T = \{\texttt{0}, \texttt{1}\},\ V_N = \{A\}$

2. Dada la siguiente gramática para la declaración de tipos en un lenguaje de programación:

   $G_1 = \langle \{S_0, T, S\}, \{\texttt{array}, \texttt{[}, \texttt{]}, \texttt{of}, \texttt{..}, \texttt{int}, \texttt{char}, \texttt{num}, \uparrow\}, S_0, P\rangle$, con P:

   $$\begin{array}{l} S_0 \longrightarrow T \\ T \longrightarrow S \mid \uparrow S \mid \texttt{array of } T \mid \texttt{array [}S\texttt{] of } T \\ S \longrightarrow \texttt{int} \mid \texttt{char} \mid \texttt{num..num} \end{array}$$

   (a) ¿Es posible crear un parser recursivo descendente predictivo para G_1? Si no, crear una gramática G_1' que genere el mismo lenguaje, y construir el parser para ella. Justificar.

   (b) Mostrar la derivación y el árbol correspondiente, paso a paso, para la siguiente cadena: `array [num..num] of` $\uparrow$ `char`

   (c) ¿Qué ocurriría con la cadena anterior si se la procesara con un parser descendente que no fuera predictivo?

Parsers LL(1)

3. Para cada G_i, decidir si es LL(1). En caso contrario, dar una gramática que sí sea LL(1) y genere $L(G_i)$. En ambos casos, indicar los símbolos directrices de cada producción de la gramática resultante.

   (a) $G_1 = \langle \{S, A\}, \{\texttt{a}, \texttt{b}\}, P_1, S\rangle$, con P_1:

   $$\begin{array}{l} S \longrightarrow \texttt{a}AS \mid \texttt{b} \\ A \longrightarrow \texttt{a} \mid \texttt{b}SA \end{array}$$

   (b) $G_2 = \langle \{S\}, \{\texttt{a}, \texttt{b}\}, P_2, S\rangle$, con P_2:

   $$S \longrightarrow \texttt{aa}S\texttt{bb} \mid \texttt{a} \mid \lambda$$

(c) $G_3 = \langle \{S, A, B\}, \{\texttt{a}, \texttt{b}\}, P_3, S \rangle$, con P_3:

$$\begin{array}{l} S \longrightarrow A \mid B \\ A \longrightarrow \texttt{a}A \mid \lambda \\ B \longrightarrow \texttt{b}B \mid \lambda \end{array}$$

(d) $G_4 = \langle \{S, A\}, \{\texttt{a}, \texttt{b}\}, P_4, S \rangle$, con P_4:

$$\begin{array}{l} S \longrightarrow \texttt{a}A\texttt{aa} \mid \texttt{b}A\texttt{ba} \\ A \longrightarrow \texttt{b} \mid \lambda \end{array}$$

(e) $G_5 = \langle \{S\}, \{\texttt{a}\}, P_5, S \rangle$, con P_5:

$$S \longrightarrow \texttt{a}S \mid \texttt{a}$$

(f) $G_6 = \langle \{S, A\}, \{\texttt{a}, \texttt{d}\}, P_6, S \rangle$, con P_6:

$$\begin{array}{l} S \longrightarrow A\texttt{a} \mid \texttt{a} \\ A \longrightarrow S\texttt{d} \mid \texttt{d} \end{array}$$

(g) $G_7 = \langle \{S, A\}, \{\texttt{a}, \texttt{c}\}, P_7, S \rangle$, con P_7:

$$\begin{array}{lcl} S & \longrightarrow & S\texttt{c} \mid \texttt{c}A \mid \lambda \\ A & \longrightarrow & \texttt{a}A \mid \texttt{a} \end{array}$$

(h) $G_8 = \langle \{S, A, L\}, \{\texttt{(}, \texttt{)}, \texttt{,}, \texttt{f}, \texttt{x}\}, P_8, S \rangle$, con P_8:

$$\begin{array}{lcl} S & \longrightarrow & \texttt{f}A \\ A & \longrightarrow & \texttt{(}L\texttt{)} \\ L & \longrightarrow & \texttt{x} \mid \texttt{x,}L \mid S \end{array}$$

(i) $G_9 = \langle \{S, A\}, \{\texttt{a}, \texttt{b}\}, P_9, S \rangle$, con P_9:

$$\begin{array}{lcl} S & \longrightarrow & S A\texttt{a} \mid A\texttt{a} \\ A & \longrightarrow & A\texttt{a} \mid \texttt{b} \end{array}$$

(j) $G_{10} = \langle \{S, T\}, \{\texttt{a}, \texttt{b}\}, P_{10}, S \rangle$, con P_{10}:

$$\begin{array}{lcl} S & \longrightarrow & \texttt{b} \mid S\texttt{b} \mid T\texttt{b} \\ T & \longrightarrow & \texttt{a}T\texttt{b} \mid \texttt{ab} \end{array}$$

4. Crear el parser LL(1) para la gramática del ejercicio 2.

Gramáticas con conflictos

5. A veces es posible utilizar un parser predictivo para reconocer el lenguaje de una gramática que tiene conflictos LL(1). Esto puede realizarse resolviendo los conflictos en favor de alguna de las producciones involucradas. Por ejemplo, la siguiente gramática representa el problema del $if - then - else$:
   $G_1 = \langle \{S, E, C\}; \{\texttt{if}, \texttt{then}, \texttt{sent}, \texttt{else}, \texttt{cond}\}; P_1; S \rangle$, con P_1:

$$\begin{array}{lcl} S & \longrightarrow & \texttt{if}\ C\ \texttt{then}\ S\ E \mid \texttt{sent} \\ E & \longrightarrow & \texttt{else}\ S \mid \lambda \\ C & \longrightarrow & \texttt{cond} \end{array}$$

(a) Mostrar que G_1 no es $LL(1)$.

(b) Indicar los símbolos directrices de cada producción.

(c) Para los terminales que sean símbolo directriz de más de una producción del mismo no terminal, asignárselos a una sola de las producciones de manera que en el árbol de derivación resultante cada $else$ quede asociado con el if más cercano.

(d) Mostrar un parseo top-down y el árbol de derivación resultante para: $if\ cond\ then\ if\ cond\ then\ sent\ else\ sent$

6. Dada la siguiente gramática que genera todas las expresiones regulares sobre los terminales $\{\texttt{a}, \texttt{b}, \texttt{c}\}$, indicar si es LL(1). Si no lo es, indicar qué cambios habría que realizarle para que lo sea, sin modificar el lenguaje generado, o si se podría resolver de otra manera (e.g., ajustando la tabla del parser).

   $G_2 = \langle \{E\}, \{\texttt{a}, \texttt{b}, \texttt{c}, \texttt{(}, \texttt{)}, \texttt{|}, \texttt{*}\}, E, P_2 \rangle$, con P_2:

$$E \longrightarrow E\,\texttt{|}\,E \mid EE \mid E\texttt{*} \mid \texttt{(}E\texttt{)} \mid \texttt{a} \mid \texttt{b} \mid \texttt{c}$$

7. Dada G_3 una gramática para expresiones de sumas y restas con la asociatividad habitual:

   $G_3 = \langle \{S\}, \{\texttt{+}, \texttt{-}, \texttt{num}, \texttt{var}\}, S, P_3 \rangle$, con P_3:

$$S \longrightarrow S\ \texttt{+}\ S \mid S\ \texttt{-}\ S \mid \texttt{num} \mid \texttt{var}$$

   (a) ¿Es G_3 LL(1)? Hacer la tabla correspondiente, indicando símbolos directrices.

   (b) En caso de haber conflictos, ¿estos podrían resolverse modificando la tabla sin cambiar el lenguaje ni la asociatividad de los operadores? Explicar cómo y por qué, si fuera posible, y dar ejemplos.

8. Dada la gramática G_4, que representa al operador condicional ternario (?), similar al existente en varios lenguajes de programación, pero con la segunda parte opcional[1],

   $G_4 = \langle \{S, C, E\}, \{\texttt{?}, \texttt{instr}, \texttt{:}, \texttt{and}, \texttt{or}, \texttt{!}, \texttt{(}, \texttt{)}, \texttt{bool}\}, S, P_4 \rangle$, con P_4:

$$\begin{array}{lcl} S & \longrightarrow & C\ \texttt{?}\ S\ E \mid \texttt{instr} \\ C & \longrightarrow & C\ \texttt{and}\ C \mid C\ \texttt{or}\ C \mid \texttt{!}\ C \mid \texttt{(}\ C\ \texttt{)} \mid \texttt{bool} \\ E & \longrightarrow & \texttt{:}\ S \mid \lambda \end{array}$$

   (a) Indicar si G_4 es LL(1), y explicar por qué.

   (b) En caso de haber conflictos, ¿estos podrían resolverse modificando la tabla y/o cambiando la gramática? Explicar cómo y por qué, y dar ejemplos.

[1] En ese caso, el valor se considera `null`

Gramáticas extendidas (ELL) y parsers recursivos-iterativos

9. Escribir gramáticas ELL(1) para:
   (a) lista de sentencias terminadas en `;`
   (b) lista de valores separados por `,`
   (c) expresiones con `+`, `-`, `*`, `(`, `)`, `id`
   (d) `if ... then ... elseif ... then ... else ... endif`, con las partes de `elseif` y `else` opcionales.

10. Dada la gramatica $G_1 = \langle\{L, R, A, D\}, \{\texttt{f}, \texttt{d}, \texttt{h}, \texttt{+}, \texttt{t}\}, P, L\rangle$, con P:

$$\begin{array}{rcl} L & \longrightarrow & R \mid R\texttt{+}L \\ R & \longrightarrow & A \mid RA \\ A & \longrightarrow & \texttt{d}L\texttt{h}D \mid \texttt{t} \mid \texttt{d}L\texttt{h} \\ D & \longrightarrow & \lambda \mid \texttt{f} \end{array}$$

   (a) Es G_1 ELL(1)? Justificar. Si no lo es, dar una gramática G_1' que sea ELL(1) y tal que L(G_1') = L(G_1) usando al menos una vez cada uno de los siguientes operadores: `*`, `+` y `?`.
   Describir cómo se llega a G_1' a partir de G_1 y justificar por qué G_1 es ELL(1).
   (b) Dar el parser recursivo-iterativo de la gramática G_1'.

11. La siguiente gramática extendida representa una versión (simplificada!) de declaraciones de variables y funciones en el lenguaje C.
    $G_2 = \langle\{Declaracion, Declarador, LParDecl, ParDecl, LId, Tipo\}, \{\texttt{(}, \texttt{)}, \texttt{[}, \texttt{]}, \texttt{,}, \texttt{*}, \texttt{int}, \texttt{char}, \texttt{ID}\}, P, Declaracion\rangle$, siendo P:

$$\begin{array}{rcl} Declaracion & \longrightarrow & Tipo\ Declarador\ (\ \texttt{,}\ Declarador\ )^* \\ Declarador & \longrightarrow & (\ \texttt{*}\ )^*\ (\ \texttt{ID} \mid \texttt{(}\ Declarador\ \texttt{)}\ ) \\ & & (\ \texttt{(}\ (\ LParDecl \mid LId\ )?\ \texttt{)} \mid \texttt{[}\ \texttt{NUM}\ ?\ \texttt{]}\ )^* \\ LParDecl & \longrightarrow & ParDecl\ (\ \texttt{,}\ ParDecl\ )^* \\ ParDecl & \longrightarrow & Tipo\ (\ Declarador\ )? \\ LId & \longrightarrow & \texttt{ID}\ (\ \texttt{,}\ \texttt{ID}\ )^* \\ Tipo & \longrightarrow & \texttt{int} \mid \texttt{char} \end{array}$$

    Escribir un parser recursivo-iterativo para este lenguaje, verificando que la gramática sea ELL(1).

12. Dada la gramática extendida G_3, que genera un directorio teléfonico con entradas delimitadas por punto y coma (`;`), donde cada entrada tiene un nombre y un número de teléfono con, opcionalmente, un carácter y dos puntos (`:`) al comienzo, que indican bajo qué letra debe agendarse esa entrada[2]
    $G_3 = \langle\{A\}, \{\texttt{a}, \texttt{num}, \texttt{,}, \texttt{:}, \texttt{;}\}, A, P_3\rangle$, con P_3:

$$A \longrightarrow (\ (\ \texttt{a}\ \texttt{:}\ \texttt{a}^*\texttt{,}\ \texttt{num} \mid \texttt{a}^*\texttt{,}\ \texttt{num}\ )\texttt{;}\ )^*$$

[2]Por defecto, cada entrada se agenda bajo la letra de comienzo del nombre, si lo hay.

(a) Mostrar que G_3 no es ELL(1).

(b) Crear una gramática de una sola producción para el lenguaje especificado, pero que sí sea ELL(1).

Por ejemplo:
la cadena original "Claudia Gaudio, 49543352; , 45763359; e:María Eugenia Numis, 1164478811;" que luego será convertida a la secuencia de tokens $w =$"`aaaaaaaaaaaaaa, num; , num; a:aaaaaaaaaaaaaaaaaaaa, num;`" $\in L(G_3)$.

Limitaciones de los parsers descendentes

13. Indicar por qué no se podría construir un parser descendente LL(1) en los siguiente casos. Justificar.

(a) La representación de las restas con $G_1 = \langle\{S\}, \{\texttt{num}, \texttt{-}\}, P, S\rangle$, manteniendo la asociatividad usual, y siendo P:
$S \longrightarrow S\texttt{-}S \mid \texttt{num}$

(b) $L_2 = \{w \mid w = a^i b^j, i \geq j\}$

(c) $L_3 = \{a^n b^m c^m \mid m, n \geq 1\} \cup \{a^n b^n c^m \mid m, n \geq 1\}$

(d) $L_4 = \{a^n b^m c^m d^n \mid m, n \geq 1\} \cup \{a^n b^n c^m d^m \mid m, n \geq 1\}$

(e) La parte de la definición y declaración de funciones, y el chequeo de múltiples definiciones de las mismas, para un lenguaje similar a C, representado por $G_2 = \langle\{F, \ldots\}, \{\texttt{type}, \texttt{id}, \texttt{args}, \texttt{body}, \texttt{(}, \texttt{)}, \texttt{\{}, \texttt{\}}, \ldots\}\rangle$, con P:

$$\ldots$$
$$F \longrightarrow \texttt{type id(args);}$$
$$F \longrightarrow \texttt{type id(args)\{body\};}$$
$$\ldots$$