Teoría de Autómatas y Lenguajes Formales

Ingeniería Técnica en Informática de Sistemas
Segundo curso, segundo cuatrimestre
Curso académico: 2010 – 2011
Departamento de Informática y Análisis Numérico
Escuela Politécnica Superior
Universidad de Córdoba

Hoja de ejercicios número 6: Gramáticas de contexto libre

1. Considérese la siguiente gramática:
   <oración> → <sujeto> <predicado>
   <sujeto> → <grupo_nominal>
   <sujeto> → <grupo_nominal> **adjetivo**
   <grupo_nominal> → **nombre** | **artículo nombre**
   <predicado> → **verbo** <complementos>
   <complementos> → <directo> <indirecto> <circunstanciales>
   <complementos> → <indirecto> <circunstanciales>
   <complementos> → <directo> <circunstanciales>
   <complementos> → <circunstanciales>
   <directo> → <grupo_nominal> | <grupo_nominal> **adjetivo**
   <indirecto> → **a** <grupo_nominal>
   <circunstanciales> → ε
   <circunstanciales> → <circunstancial> <circunstanciales>
   <circunstancial> → **preposición** <grupo_nominal>
   a) Obténgase la derivación por la izquierda de la cadena
      "**la mujer hermosa dio un beso cariñoso a Rodrigo en el parque**"
   b) Ídem por la derecha.
   c) Dibújese el árbol sintáctico asociado a la derivación por la izquierda

2. Una gramática de contexto libre posee el siguiente conjunto de reglas de producción

   P = {
   S → S D
   S → D
   D → T L **;**
   T → **int**
   T → **float**
   L → I R
   R → R **,** I
   R → ε
   I → P **identificador** C
   P → P ***
   P → ε
   C → C N
   C → ε
   N → N **[número]**

N → ε
}

Esta gramática permite generar declaraciones de variables en C como las siguientes

```
int a, *b, c[2];
float d[3][4], **e[10], f[], *g[];
```

- Elimina las producciones épsilon.
- Elimina la recursividad por la izquierda y factoriza por la izquierda las reglas de la gramática obtenidas en el apartado "a".
- Utiliza la gramática obtenida en el apartado "b" para mostrar una derivación por la izquierda de la declaración:

```
int * a; float x[3];
```

- Dibuja el árbol sintáctico de dicha derivación.

3. Comprueba si las siguientes gramáticas generan un lenguaje vacío o no:
   - P ={S → **a** A C B | A D | C B **a** A → **a** D | B C e | B **b**,
     B → B **a** | A B D **a** | **b**, C → D **c a,** D → A C **b** | **c** A S}
   - P = {E → T | E + T, T → T ***** P | P, P → F | F ^ P, F → **número** | **(** E **)** }

4. Dadas las siguientes gramáticas, construye otras equivalentes sin símbolos inútiles.
   - P ={S → A C B **d** | B **a** B, A → **a** A **d** | B C **a** | **a b**, B → **b** B **b** | **a**,
     C → C A C | A C **c**}
   - P ={S → A B | C A **d**, A → **a** A **b** | **b b** A | **a a**, B → **b** A C | **a** B | B A,
     C → **b a** C **a c** | **a b** D, D → **b** D **b c** | **c** C **a**}

5. Dadas las siguientes gramáticas:
   - P = {S → id **:=** A **;** , A →B E , B → B id **:=** | ε ,
     E → E + P | P , P → **id** | **(** A **)**}
   - P = {S → **a** S **a** | **b** S **b** | A, A → **a** B **b** D | **b** B **a**, B → **a** B | **b** B | ε}

   a) Obtén otras gramáticas equivalentes sin reglas lambda o ε - producciones.
   b) Suprime las reglas unitarias de las gramáticas obtenidas en el apartado anterior.

6. Dadas las siguientes gramáticas:
   - P = {E → T | E + T,
     T → T ***** P | P,
     P → F | F ^ P,
     F → **número** | **(** E **)** | **identificador**}
   - P = {S → ε | L I | L D **;** | D **;** | I **;** A | **;** A | **inicio** I **fin** | **inicio fin**
     L → L D **;** | D **;**
     D → **identificador** T
     T → **real** | **entero** | **carácter**
     I → I **;** A | **;** A | **inicio** I **fin** | **inicio fin**
     A → **bucle** | **sentencia_alternativa** | **llamada_a_rutina**
     }

   a) Elimina la recursividad a la izquierda.
   b) Factoriza por la izquierda las gramáticas obtenidas en el apartado anterior.

7. Dada la siguiente gramática
   P = {L → L T A **;** | T A **;**, T → **entero** | **carácter**, A → P **id** C **,** A | P **id** C,
   P → P ***** | ε, C → C **[número]** | **[número]**}
   a) Obtén una gramática equivalente si producciones épsilon (o reglas lambda).
   b) Elimina la recursividad a la izquierda y factorícese por la izquierda.
   c) Obtén una derivación por la izquierda de la cadena:
      carácter *mensaje[2], datos[3][4];
   d) Dibuja el árbol sintáctico asociado a la derivación anterior.

8. Dada la gramática:
   - P = {S → **id (** P **)**
     P → L | ε
     L → L **,** T **id** | T **id**
     T → **entero** | **real** | **carácter**}
   a) Obtén una gramática equivalente sin producciones épsilon (o reglas lambda).
   b) Elimina la recursividad a la izquierda y factorizar a la izquierda.
   c) Obtén una derivación por la izquierda de la cadena:
      feria (real toros, carácter gente)
   d) Dibuja el árbol sintáctico asociado a la derivación anterior.

9. Considera la siguiente gramática:
   - P = { <asignación_lógica> → **identificador :=** <predicado>
     <predicado> → <predicado> **or** <disyunción>
     <predicado> → <disyunción>
     <disyunción> → <disyuncion> **and** <conjunción>
     <disyunción> → <conjunción>
     <conjunción> → <simple> | **not (** <predicado> **)**
     <simple> → <operando> **operador_relacional** <operando>
     <simple> → **true** | **false** | **(** <predicado> **)**
     <operando> → **identificador** | **número**
     }
   a) Comprueba que esta gramática no genera un lenguaje vacío.
   b) Obtén las derivaciones a la izquierda y los árboles sintácticos correspondientes a las cadenas:
      1.- **X := Y > 17 and Z = 19**
      2.- **Z := not (Z>Y and Y<=9) or (Z=19 and not(Y=Z))**
   c) Suprime las reglas unitarias de esta gramática.
   d) Elimina la recursividad a la izquierda de la gramática obtenida en el apartado anterior.
   e) Factoriza por la izquierda la gramática obtenida en el apartado anterior.
   f) Utiliza la última gramática obtenida para derivar por la izquierda las cadenas indicadas en el segundo apartado.
   g) Dibuja los árboles sintácticos asociados a las derivaciones del apartado anterior.

10. Demuestra que la siguiente la gramática es ambigua:
    - P = {S → **a** | S **a** | **b** S S | S S **b** | S **b** S}
      a) Se deberá encontrar una cadena de terminales que tenga asociadas dos derivaciones por la izquierda diferentes.

b) Construye los árboles sintácticos asociados a esas derivaciones.
c) Obténganse dos derivaciones por la derecha distintas de dicha cadena de terminales y los árboles sintácticos correspondientes.

11. Muestra que la siguiente gramática es ambigua y encontrar otra equivalente que no lo sea.
    - P = { S → A | B, A → **a** A **b** | **a b** , B → **a b** B | ε}

12. Demuestra que si una gramática de contexto libre cumple la siguiente característica entonces ha de ser ambigua:
    "Existe un símbolo no terminal A que posee, simultáneamente, alguna producción recursiva por la izquierda (A → A α) y alguna producción recursiva por la derecha (A → β A)."

13. Dadas las siguientes gramáticas:
    - P = { E → T | E + T, T → T * P | P, P →F | F ^ P , F → **número** | **(** E **)**}
    - P = {S → A **a** A | C A | B **a** | B , A → **a a** B **a** | C A | **a a** C ,
      B → **b** B | **b b** | **b** A , C → C **a** | **b** C | ε}

    a) Obtén la forma normal de Chomsky.
    b) Obtén la forma normal de Greibach

14. Si G = (N,T,P,S) está en la forma normal de Chomsky y S deriva la cadena de terminales w en k pasos, siendo la longitud de w igual a n,
    $$(S \overset{k}{\Longrightarrow} w,\ \text{longitud}(w)=n,\ w \in T^*) \quad \text{¿Qué vale k?}$$

15. Diseña gramáticas de contexto libre que generen los siguientes lenguajes:
    - L = $\{w \# w^R \# \mid w \in (0+1)^*\}$
    - L = Declaraciones de variables en C de los tipos: int, short, long, float, double, char y puntero a alguno de los tipos anteriores o puntero a puntero. Obtener la derivación a la izquierda de la declaración:
      int uno, dos, tres;
      short corto, *d;
      float **flota, a; char ***astericos;
      double bed;
    - L = proposiciones lógicas en C, como por ejemplo:
      (a ==b) && (c != 10 || a>=1)
    - L = asignaciones en C.
    - L = Prototipos de funciones en C.
      - Por ejemplo, se deberá generar la declaración
        char * reserva (int n);