Computabilidad y Complejidad:

Gramáticas Formales

Salvador Marti Roman
Ramon Ruiz Dolz
Laboratorio: 3CO21

1. Dada una gramática incontextual, diremos que un símbolo auxiliar es directamente generativo si aparece como antecedente de una producción cuyo consecuente es una cadena de símbolos terminales (incluida la cadena vacía). Implemente un módulo Mathematica que, tomando como entrada una gramática incontextual, obtenga como salida una lista con aquellos símbolos auxiliares de la gramática directamente generativos.

Este primer ejercicio nos plantea la búsqueda de símbolos directamente generativos en una gramática determinada. Para ello hemos decidido realizar el siguiente algoritmo, recorremos los elementos de las listas de la parte derecha de las producciones y, mediante el módulo MemberQ comprobamos que todos esos elementos sean terminales, es decir, que no existan en la lista de auxiliares. Esto lo manejamos con una variable booleana, si detecta un elemento consecuente perteneciente a la lista de auxiliares se vuelve falso y por lo tanto luego no se añade a la lista "sol", la solución donde tendremos únicamente los símbolos directamente generativos. Finalmente, tras realizar todas las iteraciones y haber recorrido los símbolos, devolvemos la lista "sol" con todos los símbolos directamente generativos.

```
(*Ejercicio 1*)

Ejercicio1[grammar_] :=
 Module[{aux, n, i,  p, t, pright, pleft, rightlist,  k, j, sol, element},
  aux = grammar;
  n = aux[[1]];(*N, auxiliares*)
  p = aux[[3]];(*P, producciones*)

  sol = {};
  For[k = 1, k <= Length[p], k++,

   pright = p[[k]][[2]];(*Parte derecha de la producción*)

   pleft = p[[k]][[1]];(*Parte izquierda de la producción*)

   For[j = 1, j <= Length[pright], j++,

    rightlist = pright[[j]];(*Una lista de la parte derecha*)

    t = True;

    For[i = 1, i <= Length[rightlist], i++,

     element = rightlist[[i]];
     (*Comprueba si algún elemento de la lista de la parte derecha pertenece a N*)
     If[MemberQ[n, element] == True, t = False];

     ];
    If[t == True, AppendTo[sol, pleft]];
    ];
   ];
  sol = Union[Flatten[sol]];
  Return[sol];
  ]
```

2. Dada una gramática incontextual, diremos que un símbolo auxiliar es directamente no generativo si en las producciones donde aparece como antecedente aparece también en el consecuente. Implemente un módulo Mathematica que, tomando como entrada una gramática incontextual, obtenga como salida una lista con aquellos símbolos auxiliares de la gramática directamente no generativos.

En este segundo ejercicio, debemos implementar un algoritmo para buscar los símbolos auxiliares directamente no generativos. En nuestro caso hemos decidido realizar la implementación de la siguiente forma, recorremos todas las listas de consecuentes de las producciones y realizamos una intersección de estas con el antecedente, ahorrando así otro bucle adicional. Si y sólo si intersectan, este símbolo es añadido a la lista "sol", que al igual que en el ejercicio anterior, contiene ya únicamente los símbolos directamente no generativos. Finalmente, realizamos la unión y flatten a la lista sol para limpiarla y eliminar símbolos repetidos y devolvemos esta lista como solución definitiva.

```
(*Ejercicio 2*)

Ejercicio2[grammar_] :=
 Module[{aux, n, p, pright, pleft, rightlist, j, k, sol},
  aux = grammar;
  n = aux[[1]];(*N, auxiliares*)
  p = aux[[3]];(*P, producciones*)

  sol = {};
  For[k = 1, k <= Length[p], k++,

   pright = p[[k]][[2]];(*Parte derecha de la producción*)

   pleft = p[[k]][[1]];(*Parte izquierda de la producción*)

   For[j = 1, j <= Length[pright], j++,
    rightlist = pright[[j]];(*Una lista de la parte derecha*)

    (*Comprueba si el símbolo auxiliar de la parte izquierda pertenece a la lista de la parte
derecha*)

    If[Intersection[rightlist, pleft] == pleft,
     AppendTo[sol, pleft]];

    ];
   ];
  sol = Union[Flatten[sol]];
  Return[sol];
  ]
```

3. Implemente un módulo en Mathematica que, tomando como entrada una gramática incontextual devuelva True si está en Forma Normal de Chomsky y False en caso contrario.

La forma normal de Chomsky es aquella en la cual las producciones sólo dan lugar a 2 símbolos auxiliares o a un símbolo terminal. Teniendo en cuenta esto hemos implementado el siguiente algoritmo, recorremos las listas de consecuentes y comprobamos el tamaño de éstas, en caso de ser diferente de 1 o 2 podemos descartar ya que la gramática se encuentre en forma normal de Chomsky, en caso contrario, se comprueba que en caso de tener tamaño 2, todos los símbolos de esta pertenezcan íntegramente a la lista de símbolos auxiliares, es decir que la intersección de ésta con la lista de terminales sea la lista vacía, en caso de diferir de este resultado, también podemos descartar que la gramática pertenezca a la forma normal. Finalmente solo queda comprobar, que en caso de tener tamaño 1, la intersección de esta lista con los símbolos auxiliares sea la lista vacía, determinando así que el elemento pertenece a los símbolos terminales. En caso de devolver algo diferente a la lista vacía, podemos descartar que la gramática esté en forma normal de Chomsky. Este proceso se realiza con todas las listas de consecuentes, y después de recorrerlos todos (ninguna condición se ha dado) podemos afirmar rotundamente que esta gramática pertenece al conjunto de gramáticas en forma normal de Chomsky.

```
(*Ejercicio 3*)

Ejercicio3[grammar_] :=
 Module[{aux, n, p, t, pright, pleft, rightlist, i, j, k, sol },
  aux = grammar;
  n = aux[[1]];(*N, auxiliares*)
  t = aux[[2]];(*T, terminales*)

  p = aux[[3]];(*P, producciones*)
  sol = True;
  For[k = 1, k <= Length[p], k++,

   pright = p[[k]][[2]];(*Parte derecha de la producción*)

   pleft = p[[k]][[1]];(*Parte izquierda de la producción*)

   For [j = 1, j <= Length[pright], j++,

    rightlist = pright[[j]];(*Una lista de la parte derecha*)

    (*Comprueba que el tamaño de la lista de la parte derecha respeta la forma normal de Chomsky*)

    If[Length[rightlist] > 2 || Length[rightlist] < 1, sol = False];

    (*Comprueba si la lista de la parte derecha de longitud 2 contiene un símbolo terminal*)

    If[Length[rightlist] == 2,
     If[Intersection[rightlist, t] != {}, sol = False]];
    (*Comprueba si la lista de la parte derecha de longitud 1 contiene un símbolo auxiliar*)

    If[Length[rightlist] == 1,
     If[Intersection[rightlist, n] != {}, sol = False]];

    If[sol == False, Break[]];
    ];
   If[sol == False, Break[]];
   ];
  Return[sol];
  ]
```

4. Implemente un módulo en Mathematica que, tomando como entrada una gramática cuyas producciones están en una de las dos formas que se indican al final, obtenga como salida una gramática equivalente lineal por la derecha.

Una gramática linear por la derecha es una donde las producciones se presentan de la siguiente forma:

$A \rightarrow \alpha B \quad \alpha \in T^*, A, B \in N$
$A \rightarrow \alpha \quad \alpha \in T^*, A \in N$

Esta posible implementación del módulo utiliza variables con tres subíndices para nombrar los estados auxiliares adicionales que se requieren para llevar a cabo la transformación en las producciones de la gramática, a fin de simplificar el código.

Los subíndices x,y,z corresponden a: el antecedente al que pertenecían originalmente, a cuál de las posibles producciones pertenece y a que altura de la producción está "m" respectivamente.

Las variables m y n se utilizan para seleccionar símbolos específicos de los consecuentes. Por ejemplo, dado una producción A→abc, si m=1, n=2 la transición traducida sería: $A \rightarrow aX_{111}$ y $X_{111} \rightarrow b\, X_{112}$

Para esta implementación podemos distinguir un total de 5 casos clave dependiendo de si estamos tratando con el primer símbolo de una producción, si este pertenece a los auxiliares o terminales y si finalmente el siguiente símbolo "n" es el último.

Adicionalmente, es necesario tener en consideración si el consecuente es de largura 1 y el símbolo es terminal ya que por lo tanto no requeriría ninguna modificación. Similar es el caso donde la longitud es 2 y tenemos un terminal seguido por un auxiliar, a fin de no modificarlo se hace uso del módulo "Continue" para saltar a la siguiente iteración del bucle. Es decir, no debemos modificar lo que está ya dado de forma linear por la derecha.

En los casos donde m=1 solo será necesario encadenar los consecuentes a la lista de producciones de "aux" ya que creamos el estado dentro del primer bucle "i". En cambio, para m>1 será necesario encadenar a "extras" una transición completa. Cada vez que un estado auxiliar es creado se añade inmediatamente a la lista de símbolos auxiliares de "aux". Finalmente añadimos "extras" a la lista de transiciones.

```
Ejercicio4[grammar_] :=
 Module[{p, N, t, aux, pright, pleft, i, m, n, production, extras},
  N = grammar[[1]]; (*N, auxiliares*)
  t = grammar[[2]]; (*T, terminales*)
  p = grammar[[3]]; (*P, producciones*)
  aux = grammar;
  extras = {};
  aux[[3]] = {};
  For[i = 1, i <= Length[p], i++,

   pright = p[[i]][[2]];
   pleft = p[[i]][[1]];
   AppendTo[aux[[3]], {pleft, {}}];
   n = 2;
   For[production = 1, production <= Length[pright], production++,
    n = 2;

   If[Length[pright[[production]]] == 1,
     AppendTo[aux[[3]][[i]][[2]], {pright[[production]][[1]]}];
     Continue[];
   ];

   If[Length[pright[[production]]] == 2 && MemberQ[t, pright[[production]][[1]]] &&
     MemberQ[N, pright[[production]][[2]]],

     AppendTo[aux[[3]][[i]][[2]], {pright[[production]]}];
     Continue[];
```

```
   ];

    For[m = 1, m < Length[pright[[production]]], m++,

     If[MemberQ[t, pright[[production]][[n]]],   (* Es “n” terminal? *)
      If[m == 1,
       (*True*)
       AppendTo[aux[[3]][[i]][[2]], {pright[[production]][[m]], Subscript[X, i, production, m +
1]}];
       AppendTo[aux[[1]], Subscript[X, i, production, m + 1]];,

       (*False*)
       AppendTo[extras, {{Subscript[X, i, production, m]}, {{pright[[production]][[m]],
Subscript[X, i, production, m + 1]}}}];
       AppendTo[aux[[1]], Subscript[X, i, production, m + 1]];
       ];
      If[n == Length[pright[[production]]],(*
       Comprobamos si el siguiente simbolo consecuente es el último *)

       AppendTo[extras, {{Subscript[X, i, production, m + 1]}, {{pright[[production]][[n]]}}}];
       ];,
      If[m == 1,
        (*True*)
        AppendTo[aux[[3]][[i]][[2]], {pright[[production]][[m]],pright[[production]][[n]]}];,

        (*False*)
        AppendTo[extras,{{Subscript[X, i, production, m]}, {{pright[[production]][[m]],
pright[[production]][[n]]}}}];
        ];
      ];
     n++;
     ];
    ];
   ];
  AppendTo[aux[[3]], extras];
  Return[aux];
  ]
```

Dada una gramática incontextual en Forma Normal de Chomsky y una cadena w, implemente un módulo Mathematica que devuelva True si w pertenece a L(G) y False en caso contrario.

La implementación de este algoritmo CYK la hemos realizado de una forma muy similar al pseudocódigo provisto en las diapositivas. Tenemos una matriz V (Vij donde i es la longitud de la entrada w y j es cada fase de agrupación de la entrada w, grupos de 1, de 2 etc. hasta llegar a la propia cadena w) a la cual le vamos añadiendo filas en función de los consecuentes de la anterior fila. Para ello realizamos una iteración inicial, en la cual para cada símbolo añadimos los antecesores que lo generan. Tras esta primera iteración, el algoritmo realiza las iteraciones restantes, empezando ya en j=2 ("y" en el código) y agrupando los símbolos en grupos separados por K. En las iteraciones comprobamos que los consecuentes existan en las listas anteriores y se añaden a la matriz como nuevas filas tras limpiarlas y quitar símbolos repetidos con unión y flatten como en la primera iteración. Una vez la matriz V está completa, simplemente comprobamos que el símbolo inicial exista en el último elemento, es decir cuando la agrupación es igual a la longitud de la cadena w (un solo grupo). Si existe el símbolo inicial, devolvemos true, en cualquier otro caso, la cadena w no pertenece a la gramática.

V tiene esta forma al final: {{{B},{A,C},{A,C},{B},{A,C}},{{A,S},{B},{C,S},{A,S}},{{},{B},{B}},{{},{A,C,S}},{{A,C,S}}}
(cada lista es una fila y cada lista de elementos dentro de la lista anterior corresponde a las casillas).

```
(*Algoritmo CYK*)

CYK[grammar_, word_] :=
  Module[{aux, w, n, p, t, q, pright, pleft, rightlist, y, i, j, k, sol, ant, V, v, v1, vaux, J, K,
I, conj, ref1, ref2, s},

   aux = grammar;
   n = aux[[1]];(*N, auxiliares*)
   t = aux[[2]];(*T, terminales*)
   p = aux[[3]];(*P, producciones*)
   s = aux[[4]];(*S, inicial*)

   w = word;
   V = {};
   v1 = {};
   sol = False;

   (*Primera iteración del algoritmo*)
   For[y = 1, y <=  Length[w], y++,

    ant = {};

    For[j = 1, j <= Length[p], j++,

     pright = p[[j]][[2]];   (*Parte derecha de la producción*)
     pleft = p[[j]][[1]];    (*Parte izquierda de la producción*)

     For[k = 1, k <= Length[pright], k++,

      rightlist = pright[[k]];   (*Una lista de la parte derecha*)

      For[i = 1, i <= Length[rightlist], i++,

       If[w[[y]] == rightlist[[i]], AppendTo[ant, pleft];];
       ant = Flatten[ant];

       ];
      ];
     ];

    AppendTo[v1, ant];

    ];
```

```
   (*Añadimos la primera fila de la matriz Vij*)
   AppendTo[V, v1]

    For[y = 2, y <= Length[w], y++,    (*Agrupa las letras de 2 a n*)

     v = {};

     For[i = 1, i <= Length[w] - y + 1,i++,     (*Recorre las listas de agrupaciones*)
      vaux = {};

      For[j = 1, j <= Length[p], j++,

       pright = p[[j]][[2]];   (*Parte derecha de la producción*)
       pleft = p[[j]][[1]];    (*Parte izquierda de la producción*)



        For[k = 1, k <= Length[pright], k++,

        rightlist = pright[[k]];(*Una lista de la parte derecha*)

            If[Length[rightlist] == 2, (*Comprueba forma normal*)

             For[K = 1, K <= Length[w] - 1, K++,

              (*Se comparan los simbolos auxiliares de la matriz
                y de la lista de la parte derecha*)

              If[MemberQ[V[[K]][[i]], rightlist[[1]]] && MemberQ[V[[y - K]][[i + K]],
rightlist[[2]]],

               AppendTo[vaux, pleft];
               vaux = Flatten[vaux];
               vaux = Union[vaux];
              ];
            ];
          ];
         ];
        ];

      AppendTo[v, vaux];

      ];

     (*Vamos añadiendo filas a la matriz Vij*)
     AppendTo[V, v];

     ];

   Print[V];(*Impresión de la matriz del vector Vij*)

   If[MemberQ[Flatten[V[[Length[w]]]], s],

    sol = True;];(*Si esta la S en la ultima lista de V aceptamos.*)
     Return[sol];

   ];
```