# Práctica 1 – Gramáticas formales

*Computabilidad y complejidad*

**Iñaki Diez Lambies y Aitana Menárguez Box**

## PARTE 1

### Ejercicio 1

A continuación, se muestra el algoritmo implementado con *Mathematica* y explicado paso a paso.

```
ej1[gram_] := Module[{n, t, p, s, res, prod, ant, cons, con, i, j, simbolos},
                     módulo
  n = gram[[1]]; t = gram[[2]]; p = gram[[3]]; s = gram[[4]]; (* Guardamos los elementos de la gramática en variables *)
  res = {}; (* Creamos la variable para almacenar el resultado *)
  simbolos = Union[t, {{}}]; (* Los símbolos a comprobar son los terminales + la cadena vacía *)
             unión
  For[i = 1, i <= Length[p], i++, (* Recorremos las producciones *)
  para cada          longitud
   prod = p[[i]]; (* prod será la producción actual *)
   ant = prod[[1]]; (* Separamos antecedente y lista de consecuentes *)
   cons = prod[[2]];
   For[j = 1, j <= Length[cons], j++, (* Recorremos los consecuentes de esa producción *)
   para cada          longitud
    If[ContainsAll[simbolos, cons[[j]]], (* Si simbolos contiene todos los elementos del consecuente *)
    si  contiene todos
     AppendTo[res, ant]; (* Se añade el antecedente al resultado *)
     añade al final
     Break;
     finaliza iteración
    ]
   ];
  ];
  res = Flatten[Intersection[res, res]]; (* Así evitamos posibles duplicados y devolvemos una sola lista *)
        aplana    intersección
  Return [res]
  retorna
]
```

En primer lugar, se guardan los elementos de la gramática en variables para que sea más fácil acceder a ellas. En la variable ***n*** habrá una lista con los símbolos auxiliares, en ***t*** una lista con los símbolos terminales, en ***p*** una lista con las producciones y en ***s*** se encontrará el símbolo inicial. Además, creamos las variables ***res*** para almacenar los símbolos auxiliares resultado y ***simbolos***, que será la lista con los símbolos auxiliares más la cadena vacía y que nos servirá más adelante.

A continuación, se recorren las producciones para comprobar todos los consecuentes de las mismas y poder determinar qué símbolos auxiliares son directamente generativos. Éstas se dividen en *ant*, donde estará el símbolo antecedente, y *cons* que será una lista con los consecuentes de la producción a analizar.

Después, debemos recorrer cada uno de los elementos de la lista *cons*, que significa recorrer todos los consecuentes de la producción actual. Si la lista *simbolos* contiene todos los elementos que se encuentran en el consecuente, todos

los símbolos que se encuentren en el consecuente serán símbolos terminales ergo su antecedente será directamente generativo. Por ello, si se cumple la condición, se añade el antecedente al resultado y se deja de comprobar esa producción, ya que se ha determinado que el antecedente es parte del resultado.

Por último, se hace una intersección de la lista resultado consigo misma para evitar símbolos auxiliares duplicados (este paso no es estrictamente necesario) y se utiliza la función *Flatten* para hacer que el resultado sea una lista plana con los símbolos auxiliares y no una lista con listas de símbolos auxiliares.

## Ejercicio 2

```
Ej2[g_] :=
 Module[{n, t, p, s, prod, ant, cons, gen, res, i, j},
  n = g[[1]]; t = g[[2]]; p = g[[3]]; s = g[[4]]; (* Inicializamos las variables a cada elemento de la gramática *)
  res = {}; (* Aquí introduciremos la lista de antecedentes directamente no generativos *)
  For[i = 1, i ≤ Length[p], i++,
   prod = p[[i]]; (* prod es la producción que observamos*)
   gen = False; (* Damos por supuesto que es directamente no generativo *)
   ant = prod[[1]][[1]]; cons = prod[[2]]; (* En ant colocamos el antecedente y en cons la lista de consecuentes *)
   For[j = 1, j ≤ Length[cons] ∧ ¬ gen, j++, (* Para cada consecuente que aún siga siendo directamente no generativo *)
    (* Si resulta que en algún consecuente no se encuentra el propio antecedente
     eso quiere decir que este no es directamente no generativo *)
    If[¬ MemberQ[cons[[j]], ant], gen = True]
   ];
   (* Si después de mirar sus consecuentes resulta que es no generativo, lo añadimos a la solución *)
   If[¬ gen, AppendTo[res, ant]];
  ];
  Return[res]
 ]
```

## Ejercicio 3

```
Ej3[g_] :=
 Module[{n, t, p, s, prod, ant, cons, con, res, i, j, k},
  n = g[[1]]; t = g[[2]]; p = g[[3]]; s = g[[4]];  (* Inicializamos las variables a cada elemento de la gramática *)
  res = True; (* Suponemos que si que estará en FNG *)
  For[i = 1, i ≤ Length[p] ∧ res, i++, (* Para todas las producciones *)
   prod = p[[i]]; (* Guardamos el conjunto de producciones a consultar *)
   ant = prod[[1]][[1]]; cons = prod[[2]]; (* Separamos los antecedentes de los consecuentes *)
   res = MemberQ[n, ant]; (* Comprobamos que el antecedente forma parte del conjunto de símbolos auxiliares *)
   For[j = 1, j ≤ Length[cons] ∧ res, j++, (* Para todas los consecuentes *)
    con = cons[[j]]; (* Guardamos el consecuente en una variable *)
    res = MemberQ[t, First[con]]; (* Compribamos que el primer símbolo forma parte de los símbolos terminales *)
    For[k = 2, k ≤ Length[con] ∧ res, k++, (* Para el resto de símbolos *)
     res = MemberQ[n, con[[k]]]; (* Comprobamos que forman parte de los símbolos auxiliares *)
    ];
   ];
  ];
  Return[res]; (* Devolvemos el resultado *)
 ]
```

# Ejercicio 4

```
Ej4[gram_] := Module[{n, t, p, s, aux, ter, pro, ini, cont, i, pAct, ant, cons, nuevosCons, j, cAct, lengthC, k, simbolo, siguienteS, nuevaGram},
              módulo
  n = gram[[1]]; t = gram[[2]]; p = gram[[3]]; s = gram[[4]];
  aux = n; ter = t; pro = {}; ini = s; (* Creamos las variables para la nueva gramática *)
  cont = 1; (* Contador de símbolos auxiliares *)
  For[i = 1, i <= Length[p], i++, (* Recorremos las producciones *)
  para cada          longitud
   pAct = p[[i]]; (* pAct será la producción actual *)
   cons = pAct[[2]]; (* Separamos la lista de consecuentes de la producción actual *)
   For[j = 1, j <= Length[cons], j++, (* Recorremos los consecuentes de esa producción *)
   para cada          longitud
    ant = Flatten[pAct[[1]]]; (* Indicamos cuál será el antecedente inicial de la producción *)
          aplana
    cAct = cons[[j]]; (* cAct es el consecuente actual *)
    lengthC = Length[cAct];
              longitud
    For[k = 1, k ≤ lengthC, k++, (* Recorremos el consecuente símbolo a símbolo *)
    para cada
     simbolo = cAct[[k]]; (* simbolo será el símbolo actual *)
     If[MemberQ[n, simbolo], (* Si el símbolo actual es de los auxiliares, estamos en el final *)
     si  ¿contenido en?
      AppendTo[pro, {ant, {simbolo}}]; (* Solamente se entrará aquí si no había símbolos terminales *)
      añade al final
      Break[] ];
      finaliza iteración
     If[k == lengthC, (* Si está al final el simbolo terminal, es decir, no hay auxiliar *)
     si
      AppendTo[pro, {ant, {{simbolo}}}]; (* Se añade a las producciones solo *)
      añade al final
      Break[] ];
      finaliza iteración
     siguienteS = cAct[[k + 1]]; (* Cuando nos hemos asegurado de que no estamos en el último símbolo *)
                                 (* Se guarda en siguienteS el símbolo a continuación del actual *)
     If[MemberQ[n, siguienteS], (* Si el siguiente símbolo es auxiliar, será el final *)
     si  ¿contenido en?
      AppendTo[pro, {ant, {{simbolo, siguienteS}}}]; (* Ese símbolo será el siguiente *)
      añade al final
      Break[] ]; (* No se crean más consecuentes, se acaba *)
      finaliza iteración
     (* Si no estamos en ninguno de los casos anteriores *)
     AppendTo[pro, {ant, {{simbolo, {auxiliar, cont}}}}]; (* Se crea una nueva producción con el antecedente y el nuevo auxiliar *)
     añade al final
     aux = Union[aux, {auxiliar, cont}]; (* Añadimos el nuevo símbolo creado a la lista de símbolos auxiliares*)
           unión
     ant = {auxiliar, cont}; (* Hacemos que ese nuevo auxiliar creado sea ahora el antecedente de la próxima producción *)
     cont++; (* Aumentamos el contador para poder crear un nuevo auxiliar si fuera necesario *)
    ]
    (* Acabamos de recorrer todos los símbolos *)
   ]
   (* Acabamos de recorrer todos los consecuentes de la producción *)
  ];
  (* Acabamos de recorrer todas las producciones *)

  nuevaGram = {aux, ter, pro, ini}; (* Creamos la nueva gramática con las variables definidas al principio *)
  Return[nuevaGram]
  retorna
]
```

# PARTE 2

## CYK

```
CYK[g_, w_] :=
 Module[{n, t, p, s, v, i, j, k, sim, pre, suf, ants},
 módulo
  n = g[[1]]; t = g[[2]]; p = g[[3]]; s = g[[4]]; (* Inicializamos las variables a cada elemento de la gramática *)
  (* Si la palabra tiene símbolos fuera de nuestro conjunto de símbolos terminales
   "t" es imposible que esta forme parte del lenguaje *)
  If[! ContainsAll[t, w], Return[False]];
  si   contiene todos        retorna  falso
  (* Inicializamos la matriz de tamaño "longitud de w" * "longitd de w" a todo 0 (equivalente a vacio) *)
  v = SparseArray[{}, {Length[w], Length[w]}];
      array disperso          longitud  longitud
  (* En primer lugar calculamos la primera fila *)
  For[i = 1, i ≤ Length[w], i++, (* Para cada símbolo *)
  para cada        longitud
   sim = w[[i]]; (* Guardamos el símbolo actual en "sim" *)
   (* Calculamos para cada símbolo de la cadena, lo símbolos auxíliares que generan
    el símbolo terminal en cuestión.
     En otras palabras, son aquellos antecedentes del conjunto de producciones
    que contienen en su consecuente el símbolo terminal de la cadena en cuestión *)
   v[[i]][[1]] = Flatten[Cases[p, {ant_, cons_List} /; Cases[cons, {sim}] ≠ {} → ant]]; 1
                 aplana   casos                        casos
  ];
  (* Seguidamente calculamos el resto de las celdas corresponndientes
     a partir de los resultados anteriores.
      Cada celda nos indica el símbolo auxiliar que nos puede llevar a la cadena
     conformada por wi y de longitud j tal que i+j < longitud de w *)
  For[j = 2, j ≤ Length[w], j++, (* Para todo número de longitudes *)
  para cada        longitud
   For[i = 1, i ≤ (Length[w] - j + 1), i++, (* Para los i ≤ a la longitud de w - la longitud de la cadena a consultar + 1 *)
   para cada          longitud
     v[[i]][[j]] = {}; (* Inicializamos la celda a lista vacía, aquí guardaremos los símbolos auxiliares correspondientes *)
     For[k = 1, k ≤ j - 1, k++, (* Consultamos todas las posibles subcadenas que puede generar la cadena en cuestión *)
     para cada
      pre = v[[i]][[k]]; suf = v[[i + k]][[j - k]]; (* Separamos la cadena en dos subcadenas *)
      (* Realizamos la intersección de la generación de la primera subcadena con la segunda subcadena,
      esto nos dará el conjunto de símbolos auxiliares que pueden generar el conjunto de la cadena *)
      ants = Flatten[Cases[p, {ant_, cons_List} /; Cases[cons, {con1_, con2_} /; MemberQ[pre, con1] ∧ MemberQ[suf, con2]] ≠ {} → ant]];
             aplana   casos                        casos                      ¿contenido en?            ¿contenido en?
      (* Lo añadimos a la celda de forma que no repitamos los símbolos *)
      v[[i]][[j]] = Union[v[[i]][[j]], ants];
                    unión
     ];
    ];
  ];
  (* Si la celda i=1 j="longitud de w" contiene el símbolo auxiliar inicial, eso significa que la cadena pertenece al lenguaje *)
  Return[MemberQ[v[[1]][[Length[w]]], s]]
  retorna  ¿contenido en?  longitud
]
```